Docker & DevOps Basics

Containerizzazione e Deployment Moderno

15 novembre 2025

Indice

Prefazione

A chi si rivolge questo manuale

Questi appunti sono stati pensati per studenti di istituti tecnici e professionali, sviluppatori e sistemisti che vogliono apprendere le tecnologie di containerizzazione e DevOps moderne. Il percorso è strutturato per accompagnare progressivamente dalla teoria ai container fino alla gestione di infrastrutture complesse con Docker e strumenti DevOps.

Struttura del corso

Il corso è organizzato in 7 capitoli che coprono l'intero ecosistema Docker e le pratiche DevOps:

Parte I - Fondamenti Container (Capitoli 1-2)

- Introduzione ai container e differenze con le macchine virtuali
- Architettura Docker e componenti fondamentali
- Comandi base per gestire container e immagini
- Ciclo di vita di un container

Parte II - Creazione Immagini (Capitolo 3)

- Dockerfile: sintassi e istruzioni principali
- Best practices per immagini efficienti
- Multi-stage builds per ottimizzazione
- Layer caching e strategie di build

Parte III - Orchestrazione e Networking (Capitoli 4-5)

- Docker Compose per applicazioni multi-container
- Networking: bridge, host, overlay
- Gestione volumi e persistenza dati
- Service discovery e load balancing

Parte IV - Distribuzione (Capitolo 6)

- Docker Hub e registry pubblici
- Registry privati e sicurezza
- CI/CD con Docker
- Strategie di deployment

Prerequisiti

Per affrontare questo corso è consigliabile avere:

- Conoscenze base di Linux e comandi shell
- Familiarità con networking (IP, porte, protocolli)
- Conoscenza di almeno un linguaggio di programmazione
- Comprensione dei concetti di client-server
- (Opzionale) Esperienza con macchine virtuali

Strumenti necessari

Software consigliato:

- **Docker Engine**: Runtime per container Linux/Windows
- **Docker Desktop**: Applicazione GUI per macOS/Windows
- **Docker Compose**: Orchestrazione multi-container
- **Visual Studio Code**: Editor con estensioni Docker
- **Portainer**: Interfaccia web per gestione Docker

Ambienti di sviluppo:

- **Linux**: Ubuntu 20.04+, Debian, CentOS, Fedora
- **Windows**: Windows 10/11 Pro con WSL2
- **macOS**: macOS 10.15+ con Docker Desktop
- **Cloud**: AWS, Azure, Google Cloud (livello free tier)

Tool aggiuntivi:

- **Git**: Versioning del codice e Dockerfile
- **curl/wget**: Testing API e download
- **jq**: Parsing JSON per inspect e API
- **dive**: Analisi layer immagini Docker

Come studiare

Per ottenere il massimo da questi appunti:

1. **Installa Docker**: Configura l'ambiente sul tuo sistema
2. **Digita i comandi**: Non copiare/incollare, scrivi manualmente
3. **Sperimenta**: Modifica i Dockerfile e osserva i risultati
4. **Leggi i log**: Impara a debuggare container in errore

5. **Costruisci progetti**: Containerizza applicazioni reali
6. **Studia i layer**: Usa `docker history` e `dive`
7. **Pratica networking**: Testa comunicazione tra container
8. **Ottimizza**: Riduci dimensioni immagini e tempi di build

Nota

Questo manuale usa **Docker Engine 20.10+** e **Docker Compose V2**. La maggior parte dei comandi funziona anche su versioni precedenti, ma alcune funzionalità avanzate richiedono versioni recenti.

Convenzioni tipografiche

Nel testo vengono utilizzate le seguenti convenzioni:

- `Comandi shell`: Comando da eseguire in terminale
- **Parole chiave**: Concetti importanti (container, image, volume)
- *Nomi di file/path*: Riferimenti a file (Dockerfile, /var/lib/docker)
- Box colorati: Note, Attenzioni, Best Practices, Errori Comuni
- Diagrammi: Architetture e flussi con TikZ

Formato comandi:

```
# Commento esplicativo
$ docker comando [OPZIONI] ARGOMENTO
```

Output esempio:

```
CONTAINER ID   IMAGE    COMMAND   STATUS
a1b2c3d4e5f6   nginx    ...       Up 2 hours
```

Architettura del manuale

Struttura di ogni capitolo:

1. **Obiettivi**: Cosa imparerai
2. **Teoria**: Concetti fondamentali
3. **Pratica**: Esempi completi commentati
4. **Diagrammi**: Visualizzazione architetture
5. **Best Practices**: Consigli professionali
6. **Errori Comuni**: Problemi da evitare
7. **Debugging**: Troubleshooting e log analysis
8. **Esercizi**: Sfide pratiche graduate
9. **Caso di Studio**: Progetto completo
10. **Riepilogo**: Riassunto concetti chiave
11. **Riferimenti**: Documentazione ufficiale

Laboratorio pratico

Durante il corso costruirai:

- **Web app multi-tier**: Frontend + Backend + Database
- **Microservizi**: Architettura distribuita con API
- **CI/CD Pipeline**: Build automatizzata e deployment
- **Monitoring stack**: Prometheus + Grafana
- **Reverse proxy**: Nginx per load balancing

Certificazioni

Questo corso prepara per:

- **Docker Certified Associate (DCA)**
- **Kubernetes fundamentals** (passo successivo naturale)
- **Linux Foundation certifications** (DevOps track)

Sito web e risorse

Materiale aggiuntivo disponibile su:

- Repository GitHub: `https://github.com/campionluca/Appunti`
- Dockerfile di esempio scaricabili
- Docker Compose templates per progetti comuni
- Script di automazione e best practices
- Video tutorial e screencast
- Community Discord per supporto

Filosofia DevOps

Docker è uno strumento fondamentale nella cultura DevOps:

Principi DevOps

- **Automation**: Automatizza build, test, deployment
- **CI/CD**: Integrazione e consegna continue
- **Infrastructure as Code**: Infrastruttura versionata
- **Monitoring**: Osservabilità e metriche
- **Collaboration**: Dev e Ops lavorano insieme
- **Feedback rapido**: Cicli brevi di sviluppo

Container nel mondo reale

Docker è utilizzato da:

- **Startup**: Deployment rapido e scalabile
- **Enterprise**: Modernizzazione applicazioni legacy
- **Cloud providers**: AWS ECS/Fargate, Azure ACI, GCP Cloud Run
- **Kubernetes**: Orchestrazione container in produzione
- **Sviluppatori**: Ambienti consistenti dev/staging/prod

Roadmap di apprendimento

Percorso consigliato:

1. **Settimana 1-2**: Capitoli 1-2 (fondamenti e comandi base)
2. **Settimana 3-4**: Capitolo 3 (Dockerfile e build)
3. **Settimana 5-6**: Capitolo 4 (Docker Compose)
4. **Settimana 7-8**: Capitolo 5 (networking e volumes)
5. **Settimana 9-10**: Capitolo 6 (registry e deployment)
6. **Settimana 11-12**: Progetto finale completo

Progetto finale

Al termine del corso sarai in grado di:

- Containerizzare qualsiasi applicazione
- Creare Dockerfile ottimizzati e sicuri
- Orchestrare stack multi-container con Compose
- Configurare reti e volumi persistenti
- Distribuire su registry pubblici e privati
- Implementare CI/CD con GitHub Actions + Docker
- Debuggare problemi di container in produzione

Community e supporto

Dove trovare aiuto:

- **Docker Forums**: `https://forums.docker.com`
- **Stack Overflow**: Tag [docker] e [dockerfile]
- **Docker Community Slack**: Chat in tempo reale
- **Reddit**: r/docker per discussioni e best practices
- **GitHub Issues**: Report bug e feature requests

Sicurezza

Attenzione

La sicurezza dei container è fondamentale:

- Non eseguire container come root se evitabile
- Scansiona immagini per vulnerabilità (Trivy, Snyk)
- Usa immagini ufficiali da registry fidati
- Aggiorna regolarmente base images
- Limita risorse CPU/RAM per prevenire DoS
- Usa secrets manager per credenziali sensibili

Ringraziamenti

Si ringrazia:

- Docker Inc. per l'eccellente documentazione ufficiale
- La community open source per contributi e feedback
- L'Istituto Tecnico Antonio Scarpa per il supporto
- Gli studenti che hanno testato e migliorato questi materiali

Prof. Luca Campion
Novembre 2025

Note sulla versione

Versione 1.0 - Novembre 2025

- Prima release completa
- 7 capitoli + esempi pratici
- Coverage Docker: Engine, Compose, Networking, Registry
- Esempi testati su Docker 20.10+
- Diagrammi architettura con TikZ
- 100+ esempi di codice funzionanti
- Casi di studio reali da produzione

Prossimi aggiornamenti:

- Kubernetes fundamentals (orchestrazione avanzata)
- Docker Swarm per clustering

- Security scanning e hardening
- Monitoring con Prometheus/Grafana
- Service mesh con Istio

Feedback

Questo manuale è in continua evoluzione. Invia suggerimenti, correzioni o richieste a:

- Email: `luca.campion@example.com`
- GitHub Issues: `https://github.com/campionluca/Appunti/issues`
- Pull Requests benvenute!

Licenza

Questo materiale è distribuito con licenza Creative Commons BY-NC-SA 4.0:

- **BY**: Attribuzione obbligatoria
- **NC**: Uso non commerciale
- **SA**: Condivisione con stessa licenza

"Containers are the future of software deployment"
– Solomon Hykes, Docker Founder

Capitolo 1

Introduzione ai Container

Introduzione

I container rappresentano una rivoluzione nel modo in cui sviluppiamo, distribuiamo ed eseguiamo applicazioni. Questo capitolo introduce i concetti fondamentali della containerizzazione, le differenze con le macchine virtuali tradizionali e l'architettura di Docker.

Obiettivi di apprendimento

- Comprendere cosa sono i container e come funzionano
- Confrontare container e macchine virtuali
- Conoscere i vantaggi della containerizzazione
- Capire l'architettura di Docker e i suoi componenti
- Identificare i casi d'uso appropriati per i container

1.1 Cos'è un Container?

1.1.1 Definizione

Un **container** è un'unità software standardizzata che impacchetta il codice e tutte le sue dipendenze in modo che l'applicazione possa essere eseguita in modo rapido e affidabile da un ambiente di computing a un altro.

Analogia: Container di Spedizione

Come i container di spedizione standardizzano il trasporto merci, i container software standardizzano il deployment di applicazioni:

- **Dimensioni standard**: Formato uniforme e prevedibile
- **Portabilità**: Si spostano facilmente tra navi, treni, camion
- **Isolamento**: Il contenuto è separato dall'esterno
- **Efficienza**: Caricamento/scaricamento ottimizzato

1.1.2 Caratteristiche principali

1. **Isolamento**: Ogni container ha il proprio filesystem, processi, networking
2. **Portabilità**: "Build once, run anywhere" - funziona su qualsiasi sistema
3. **Leggerezza**: Condivide il kernel dell'host, avvio in secondi
4. **Immutabilità**: L'immagine non cambia, deployment consistenti
5. **Scalabilità**: Facilmente replicabile per gestire carico

1.2 Container vs Macchine Virtuali

1.2.1 Architettura a confronto

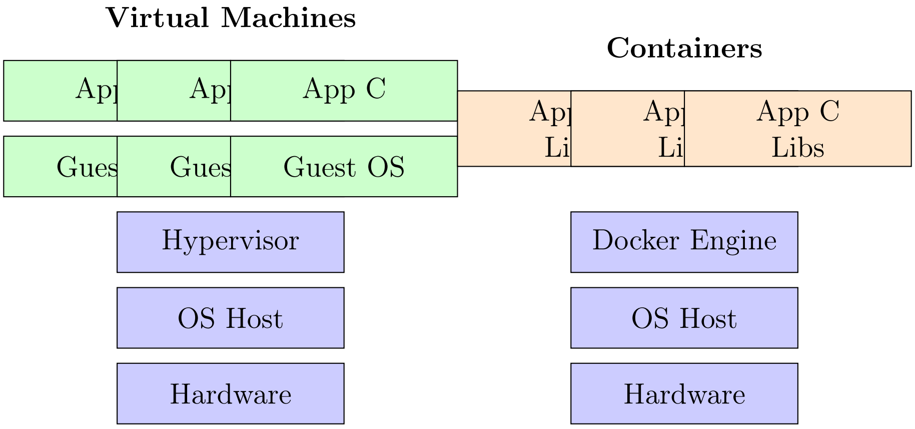

Figura 1.1: Architettura: Virtual Machines vs Containers

1.2.2 Differenze chiave

Caratteristica	Virtual Machine	Container
Dimensione	GB (include intero OS)	MB (solo app + dipendenze)
Avvio	Minuti	Secondi
Performance	Overhead hypervisor	Quasi native
Isolamento	Completo (hardware)	A livello processo
Portabilità	Limitata (formato VM)	Eccellente (standard OCI)
Densità	Decine per host	Centinaia per host

Tabella 1.1: Confronto VM vs Container

1.2.3 Virtual Machines

Vantaggi:

- Isolamento completo a livello hardware
- Esecuzione di OS diversi sullo stesso host
- Sicurezza superiore (separazione hypervisor)

- Supporto per applicazioni legacy

Svantaggi:

- Overhead significativo (ogni VM ha un OS completo)
- Avvio lento (boot del sistema operativo)
- Consumo elevato di risorse (RAM, CPU, disco)
- Portabilità limitata tra hypervisor diversi

1.2.4 Containers

Vantaggi:

- Leggerezza: condividono il kernel dell'host
- Avvio istantaneo (secondi)
- Alta densità: centinaia di container per server
- Portabilità: funzionano ovunque ci sia Docker
- Efficienza: minor spreco di risorse
- CI/CD: integrazione perfetta in pipeline DevOps

Svantaggi:

- Isolamento meno robusto delle VM
- Stesso kernel dell'host (no OS diversi)
- Sicurezza: vulnerabilità kernel colpisce tutti i container
- Non adatti per applicazioni che richiedono kernel diverso

Nota

Container e VM non sono mutualmente esclusivi. Molte architetture moderne usano **container dentro VM**: le VM forniscono isolamento hardware, i container portano portabilità e densità.

1.3 Vantaggi della Containerizzazione

1.3.1 1. Portabilità e Consistenza

Problema: "Works on my machine"

Scenario tradizionale:

- Sviluppo su macOS
- Staging su Ubuntu 20.04
- Produzione su CentOS 8
- Risultato: bug dipendenti dall'ambiente

Soluzione con container:

- Immagine Docker identica ovunque
- Stesso runtime, librerie, dipendenze
- Risultato: comportamento prevedibile

1.3.2 2. Microservizi e Scalabilità

I container sono ideali per architetture a microservizi:

- **Isolamento**: Ogni servizio in un container separato
- **Scalabilità indipendente**: Scala solo i servizi sotto carico
- **Deployment incrementale**: Aggiorna un servizio alla volta
- **Resilienza**: Fallimento di un container non compromette il sistema

Listing 1.1: Esempio: Stack microservizi

```
# Frontend
Container 1: React app (3 repliche)

# Backend API
Container 2: Node.js API (5 repliche)
Container 3: Python ML service (2 repliche)

# Database
Container 4: PostgreSQL (1 replica master)
Container 5: Redis cache (2 repliche)
```

1.3.3 3. DevOps e CI/CD

I container accelerano il ciclo di sviluppo:

1. **Sviluppo**: Ambiente identico per tutti i developer
2. **Testing**: Test automatici in container isolati
3. **Build**: Immagine Docker come artifact immutabile
4. **Deployment**: Push immagine su registry, pull in produzione
5. **Rollback**: Ritorna alla versione precedente in secondi

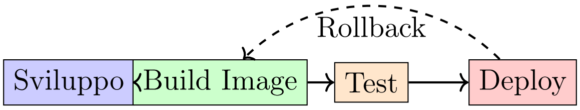

Figura 1.2: Pipeline CI/CD con Docker

Metrica	**VM**	**Container**	**Risparmio**
Memoria per istanza	2 GB	100 MB	95%
Tempo avvio	60 sec	2 sec	97%
Istanze per server	10	100	10x
Costo cloud mensile	$500	$50	90%

Tabella 1.2: Confronto efficienza risorse (valori medi)

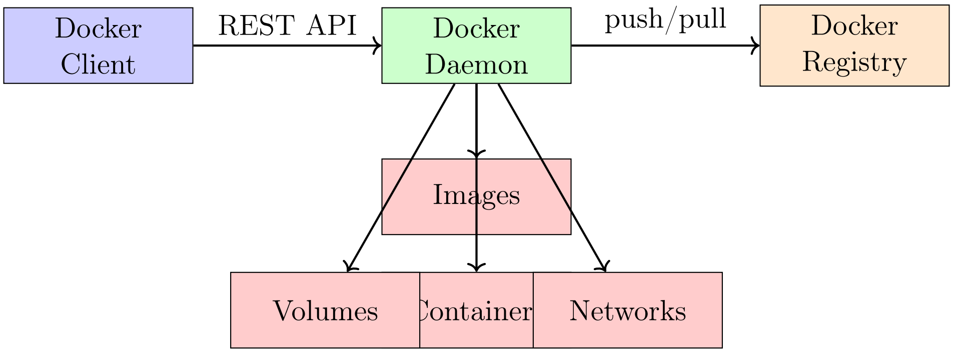

Figura 1.3: Architettura Docker

1.3.4 4. Efficienza delle Risorse

1.4 Architettura Docker

1.4.1 Componenti principali

Docker Client

Interfaccia utente per interagire con Docker:

- Comando `docker`: CLI principale
- Invia comandi al Docker Daemon via REST API
- Può connettersi a daemon remoti

```
# Esempi di comandi client
$ docker run nginx
$ docker ps
$ docker build -t myapp .
$ docker push myapp:latest
```

Docker Daemon (dockerd)

Il cuore di Docker che gestisce:

- Building, running, distributing container
- Gestione immagini, container, reti, volumi
- Comunicazione con registry per push/pull
- Esposizione REST API per client

Docker Registry

Repository per immagini Docker:

- **Docker Hub**: Registry pubblico ufficiale
- **Registry privati**: Harbor, Artifactory, AWS ECR, GCP GCR
- **Self-hosted**: Registry Docker open source

1.4.2 Oggetti Docker

Immagini

Template **read-only** per creare container:

- File system stratificato (layers)
- Definite da un Dockerfile
- Versionabili con tags (latest, v1.0, stable)
- Riutilizzabili e componibili

Listing 1.2: Struttura immagine a layer

```
Layer 5: App code (Python)         10 MB
Layer 4: pip install requirements   50 MB
Layer 3: Python 3.9                 100 MB
Layer 2: OS libraries (Ubuntu)      30 MB
Layer 1: Base layer                 5 MB
----------------------------------------
Total:                              195 MB
```

Container

Istanza **eseguibile** di un'immagine:

- Processo isolato con proprio filesystem
- Layer writable sopra l'immagine
- Effimero: può essere fermato, rimosso, ricreato
- Configurabile: variabili ambiente, porte, volumi

Volumi

Persistenza dati al di fuori del container:

- Sopravvivono alla cancellazione del container
- Condivisibili tra più container
- Gestiti da Docker (ottimizzazione I/O)

Reti

Comunicazione tra container e verso l'esterno:

- **Bridge**: Rete privata isolata (default)
- **Host**: Usa network stack dell'host
- **Overlay**: Multi-host networking (Swarm/Kubernetes)
- **None**: Nessun networking

1.5 Tecnologie Sottostanti

1.5.1 Namespace Linux

Isolamento delle risorse del sistema:

- **PID**: Albero processi isolato
- **Network**: Stack di rete separato
- **Mount**: Filesystem isolato
- **UTS**: Hostname e domain name
- **IPC**: Inter-process communication
- **User**: Mapping UID/GID

1.5.2 Control Groups (cgroups)

Limitazione e accounting delle risorse:

- CPU: Limiti di utilizzo processore
- Memoria: Limiti RAM e swap
- I/O: Bandwidth disco
- Network: Bandwidth rete

Listing 1.3: Esempio: Limitare risorse container

```
# Limita a 1 CPU e 512 MB RAM
$ docker run --cpus="1.0" --memory="512m" nginx
```

1.5.3 Union File System

Filesystem stratificato:

- **OverlayFS**: Default su Linux moderno
- **AUFS**: Legacy Ubuntu
- **Btrfs/ZFS**: COW filesystem avanzati

Vantaggi:

- Condivisione layer tra immagini (risparmio spazio)
- Build veloce (caching layer)
- Pull efficiente (solo layer mancanti)

1.6 Storia ed Evoluzione

1.6.1 Timeline

- **1979**: chroot (primi concetti di isolamento)
- **2000**: FreeBSD Jails
- **2005**: OpenVZ, Solaris Zones
- **2008**: LXC (Linux Containers)
- **2013**: Docker Inc. lancia Docker
- **2014**: Kubernetes (orchestrazione Google)
- **2015**: Docker Compose
- **2017**: Docker Swarm mode
- **2020**: Docker supporta Windows containers
- **2021**: containerd diventa CNCF graduated project

1.6.2 Open Container Initiative (OCI)

Standardizzazione del formato container:

- **Image spec**: Formato immagine universale
- **Runtime spec**: Specifiche esecuzione container
- **Distribution spec**: Distribuzione via registry

Implementazioni OCI:

- Docker
- containerd
- CRI-O (Kubernetes)
- Podman (Red Hat)

1.7 Casi d'Uso

1.7.1 Quando usare i container

Ideali per:

- Microservizi e API stateless
- Applicazioni web moderne (MERN, LAMP, MEAN)
- CI/CD e ambienti di sviluppo
- Batch processing e job worker
- Funzioni serverless (AWS Lambda usa container)

Non ideali per:

- Applicazioni GUI desktop
- Kernel modules e driver
- Applicazioni che richiedono hardware specifico
- Database con I/O intensivo (meglio VM o bare metal)

1.7.2 Esempi reali

Caso 1: E-commerce Platform

Architettura:

- 10 container frontend (React)
- 20 container backend (Node.js API)
- 5 container cart service (Python)
- 3 container payment gateway
- 2 database (PostgreSQL + Redis)

Risultati:

- Deploy 50 volte al giorno (vs 1 volta/settimana)
- Downtime ridotto 99%
- Costi cloud -60%

Caso 2: Machine Learning Pipeline

Setup:

- Container data ingestion (Kafka)
- Container preprocessing (Spark)
- Container training (TensorFlow GPU)
- Container model serving (Flask API)

Vantaggi:

- Riproducibilità esperimenti
- Scalabilità training parallelizzato
- Deployment modelli senza downtime

Best Practices

Best Practices Iniziali

1. **Un processo per container**: Non usare supervisord/systemd
2. **Immutabilità**: Mai modificare container in esecuzione
3. **Stateless**: Stato persistente su volumi esterni
4. **Logging**: Output su stdout/stderr, non file
5. **Configurazione**: Usa variabili ambiente, non file config
6. **Sicurezza**: Non eseguire come root se evitabile

Errori Comuni

Attenzione

Errori da evitare:

- Trattare container come VM (ssh, multiple process)
- Salvare dati importanti nel container filesystem
- Immagini enormi (GB) con software inutile
- Eseguire tutto come root
- Hardcodare configurazione nel Dockerfile
- Non versionare immagini (usare sempre tags)

Esercizi

1. Disegna un diagramma che confronta l'architettura di VM e container, evidenziando le differenze di layer.

2. Spiega con un esempio concreto come i container risolvono il problema "works on my machine".

3. Identifica 3 applicazioni nella tua scuola/azienda che potrebbero beneficiare della containerizzazione. Motiva la scelta.

4. Calcola il risparmio teorico: hai 50 applicazioni che richiedono 1GB RAM ciascuna. Confronta il costo di:
   - VM (overhead 2GB per VM)
   - Container (overhead 100MB per container)

5. Ricerca: trova 3 aziende famose che usano Docker in produzione e scopri come lo utilizzano.

Quiz di Verifica

1. **Vero/Falso**: I container condividono il kernel dell'host.

2. **Vero/Falso**: Un container può eseguire Windows su un host Linux.
3. Quale componente Docker gestisce la comunicazione tra client e daemon?
    - a) Registry
    - b) REST API
    - c) Dockerfile
    - d) Namespace
4. Qual è il vantaggio principale dei layer nelle immagini Docker?
5. Quando preferiresti una VM a un container?

Riepilogo Concetti Chiave

Concetti Fondamentali

- I **container** sono unità software leggere e portabili
- **Vantaggi vs VM**: Più leggeri, avvio rapido, alta densità
- **Docker** è la piattaforma leader per containerizzazione
- **Architettura**: Client, Daemon, Registry, Objects
- **Tecnologie**: Namespace, cgroups, Union FS
- **Portabilità**: Build once, run anywhere
- **DevOps**: CI/CD, microservizi, scalabilità

Prossimi Passi

Nel prossimo capitolo esploreremo:

- Installazione Docker su diversi sistemi operativi
- Comandi base: run, ps, images, stop, rm
- Gestione del ciclo di vita dei container
- Debugging e troubleshooting

Riferimenti

- Docker Official Docs: `https://docs.docker.com`
- OCI Specifications: `https://opencontainers.org`
- Linux Namespaces: `https://man7.org/linux/man-pages/man7/namespaces.7.html`
- cgroups: `https://www.kernel.org/doc/Documentation/cgroup-v2.txt`
- Docker Blog: `https://www.docker.com/blog`
- CNCF: `https://www.cncf.io`

Capitolo 2

Docker Basics: Comandi Fondamentali

Introduzione

Questo capitolo copre i comandi essenziali di Docker per gestire container e immagini. Imparerai a installare Docker, eseguire container, gestire il loro ciclo di vita e risolvere i problemi più comuni.

Obiettivi di apprendimento

- Installare Docker su Linux, macOS e Windows
- Eseguire container con `docker run`
- Ispezionare container e immagini
- Gestire il ciclo di vita dei container
- Visualizzare log e debuggare problemi
- Pulire risorse inutilizzate

2.1 Installazione Docker

2.1.1 Linux (Ubuntu/Debian)

Listing 2.1: Installazione su Ubuntu 20.04+

```
# Aggiorna repository
$ sudo apt-get update

# Installa dipendenze
$ sudo apt-get install ca-certificates curl gnupg lsb-release

# Aggiungi GPG key ufficiale Docker
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Configura repository
$ echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```

# Installa Docker Engine
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io

# Verifica installazione
$ sudo docker --version
Docker version 20.10.17, build 100c701

# Test con hello-world
$ sudo docker run hello-world
```

Esegui Docker senza sudo

```
# Crea gruppo docker
$ sudo groupadd docker

# Aggiungi utente al gruppo
$ sudo usermod -aG docker $USER

# Applica cambiamenti (logout/login oppure)
$ newgrp docker

# Test senza sudo
$ docker run hello-world
```

2.1.2 macOS

1. Scarica **Docker Desktop** da https://www.docker.com/products/docker-desktop
2. Apri il file `Docker.dmg` e trascina Docker in Applications
3. Avvia Docker Desktop dalla cartella Applicazioni
4. Attendi l'icona Docker nella menu bar (whale)
5. Apri terminale e verifica:

```
$ docker --version
Docker version 20.10.17, build 100c701
```

2.1.3 Windows

Requisiti:

- Windows 10/11 Pro, Enterprise o Education
- WSL 2 (Windows Subsystem for Linux)
- Virtualizzazione abilitata nel BIOS

1. Abilita WSL 2:

```
# PowerShell come Amministratore
> wsl --install
> wsl --set-default-version 2
```

2. Scarica Docker Desktop per Windows
3. Installa e riavvia
4. Configura: Settings -> General -> Use WSL 2 based engine
5. Verifica in PowerShell:

```
> docker --version
```

2.2 Docker Run: Eseguire Container

2.2.1 Sintassi base

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

2.2.2 Primo container

Listing 2.2: Hello World

```
$ docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:7d246653d0511db2a6b2e0436cfd0e52ac8c066000264b3ce63331ac66dca625
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Cosa è successo?

1. Docker cerca l'immagine `hello-world` localmente
2. Non trovandola, la scarica da Docker Hub
3. Crea un container dall'immagine
4. Esegue il container (stampa il messaggio)
5. Il container termina (processo completato)

2.2.3 Container interattivo

Listing 2.3: Ubuntu shell interattiva

```
$ docker run -it ubuntu bash

# Ora sei dentro il container Ubuntu
root@a1b2c3d4e5f6:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="22.04 LTS (Jammy Jellyfish)"

root@a1b2c3d4e5f6:/# ls /
```

```
bin boot dev etc home lib media mnt opt proc root run sbin srv sys tmp usr var

root@a1b2c3d4e5f6:/# exit
```

Opzioni:

- `-i`: Interactive (mantieni stdin aperto)
- `-t`: TTY (alloca un pseudo-terminale)
- `bash`: Comando da eseguire nel container

2.2.4 Container in background (detached)

Listing 2.4: Web server Nginx

```
$ docker run -d -p 8080:80 --name webserver nginx

# -d: Detached mode (background)
# -p 8080:80: Mappa porta host:container
# --name: Assegna nome al container
# nginx: Immagine da usare

# Output: container ID
a1b2c3d4e5f67890abcdef1234567890

# Testa nel browser: http://localhost:8080
# Oppure con curl
$ curl http://localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

2.2.5 Opzioni comuni di docker run

Opzione	**Descrizione**
`-d`	Detached mode (background)
`-it`	Interactive + TTY
`-p 8080:80`	Pubblica porta host:container
`-name myapp`	Nome personalizzato
`-v /host:/container`	Monta volume
`-e VAR=value`	Variabile ambiente
`-rm`	Rimuovi container quando termina
`-network net1`	Connetti a rete specifica
`-restart always`	Policy di restart
`-cpus="1.5"`	Limita CPU
`-memory="512m"`	Limita RAM

Tabella 2.1: Opzioni principali di docker run

2.2.6 Esempi pratici

Listing 2.5: Database MySQL

```
$ docker run -d \
  --name mysql-db \
  -e MYSQL_ROOT_PASSWORD=secret \
  -e MYSQL_DATABASE=myapp \
  -p 3306:3306 \
  -v mysql-data:/var/lib/mysql \
  mysql:8.0

# Connetti al database
$ docker exec -it mysql-db mysql -u root -p
Enter password: secret
mysql> SHOW DATABASES;
```

Listing 2.6: Redis cache

```
$ docker run -d \
  --name redis-cache \
  -p 6379:6379 \
  redis:alpine

# Test connessione
$ docker exec -it redis-cache redis-cli
127.0.0.1:6379> PING
PONG
127.0.0.1:6379> SET mykey "Hello Docker"
OK
127.0.0.1:6379> GET mykey
"Hello Docker"
```

2.3 Docker PS: Ispezionare Container

2.3.1 Listare container in esecuzione

```
$ docker ps

CONTAINER ID   IMAGE     COMMAND                    CREATED          STATUS
            PORTS                        NAMES
a1b2c3d4e5f6   nginx     "/docker-entrypoint...."   5 minutes ago    Up 5
    minutes    0.0.0.0:8080->80/tcp      webserver
f6e5d4c3b2a1   redis     "docker-entrypoint.s..."   2 hours ago      Up 2
    hours      0.0.0.0:6379->6379/tcp    redis-cache
```

2.3.2 Listare tutti i container (anche fermati)

```
$ docker ps -a

CONTAINER ID   IMAGE       COMMAND    CREATED           STATUS
                        PORTS    NAMES
a1b2c3d4e5f6   nginx         "..."     5 minutes ago     Up 5 minutes
                   8080:80    webserver
9876543210ab   ubuntu        "bash"     10 minutes ago    Exited (0) 8
   minutes ago               clever_einstein
```

```
5432167890cd   hello-world   "/hello"   1 hour ago        Exited (0) 1 hour ago                   stoic_tesla
```

2.3.3 Formattazione output

```
# Solo ID container
$ docker ps -q
a1b2c3d4e5f6
f6e5d4c3b2a1

# Custom format
$ docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Status}}"
CONTAINER ID   NAMES           STATUS
a1b2c3d4e5f6   webserver       Up 10 minutes
f6e5d4c3b2a1   redis-cache     Up 2 hours

# JSON output
$ docker ps --format json
{"Command":"\"/docker-entrypoint....\"","CreatedAt":"2025-11-15 10:00:00","ID":"a1b2c3d4e5f6",...}
```

2.3.4 Filtri

```
# Container per nome
$ docker ps --filter "name=web"

# Container per status
$ docker ps -a --filter "status=exited"

# Container per label
$ docker ps --filter "label=env=production"

# Container per ancestor (immagine)
$ docker ps --filter "ancestor=nginx"
```

2.4 Docker Images: Gestire Immagini

2.4.1 Listare immagini locali

```
$ docker images

REPOSITORY     TAG        IMAGE ID       CREATED         SIZE
nginx          latest     605c77e624dd   2 weeks ago     141MB
redis          alpine     a49ff3e0d85f   3 weeks ago     32.3MB
mysql          8.0        3218b38490ce   1 month ago     516MB
ubuntu         22.04      216c552ea5ba   2 months ago    77.8MB
hello-world    latest     feb5d9fea6a5   14 months ago   13.3kB
```

2.4.2 Cercare immagini su Docker Hub

```
$ docker search python

```

```
NAME                                DESCRIPTION
                                          STARS      OFFICIAL
python                              Python is an interpreted, interactive,
   objec...   9876       [OK]
pypy                                PyPy is a fast, compliant alternative
   implem...   345        [OK]
circleci/python                     Python is an interpreted, interactive,
   objec...   89
```

2.4.3 Scaricare immagini (pull)

```
# Ultima versione (tag latest)
$ docker pull python
Using default tag: latest
latest: Pulling from library/python
...

# Versione specifica
$ docker pull python:3.9-slim
3.9-slim: Pulling from library/python
...

# Da registry privato
$ docker pull myregistry.com:5000/myapp:v1.0
```

2.4.4 Rimuovere immagini

```
# Per ID
$ docker rmi 605c77e624dd

# Per nome:tag
$ docker rmi nginx:latest

# Forza rimozione (anche se usata)
$ docker rmi -f nginx

# Rimuovi immagini dangling (senza tag)
$ docker image prune

# Rimuovi tutte le immagini non usate
$ docker image prune -a
```

2.4.5 Ispezionare immagini

```
# Informazioni dettagliate
$ docker inspect nginx
[
    {
        "Id": "sha256:605c77e624dd...",
        "RepoTags": ["nginx:latest"],
        "Created": "2025-10-28T10:15:30.123456789Z",
        "Size": 141234567,
        ...
    }
]

# Estrai campo specifico con jq
$ docker inspect nginx | jq '.[0].Config.ExposedPorts'
{
  "80/tcp": {}
}

# History dei layer
$ docker history nginx
IMAGE          CREATED        CREATED BY
                                         SIZE
605c77e624dd   2 weeks ago    CMD ["nginx" "-g" "daemon off;"]
                  0B
<missing>      2 weeks ago    STOPSIGNAL SIGQUIT
                                0B
<missing>      2 weeks ago    EXPOSE 80
                                         0B
<missing>      2 weeks ago    COPY file:abc123... /etc/nginx/nginx.conf
         4.5kB
...
```

2.5 Gestione Ciclo di Vita Container

2.5.1 Stati del container

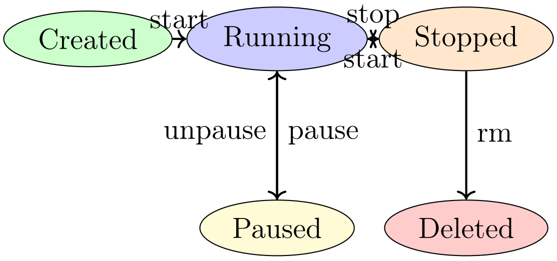

Figura 2.1: Stati del ciclo di vita di un container

2.5.2 Stop e Start

```
# Ferma container (graceful shutdown, SIGTERM poi SIGKILL)
$ docker stop webserver
webserver

# Ferma con timeout custom (default 10s)
$ docker stop -t 30 webserver

# Ferma forzatamente (SIGKILL immediato)
$ docker kill webserver

# Riavvia container fermo
$ docker start webserver

# Riavvia container in esecuzione
$ docker restart webserver
```

2.5.3 Pause e Unpause

```
# Congela processi del container (cgroup freezer)
$ docker pause webserver

# Riprendi esecuzione
$ docker unpause webserver
```

2.5.4 Rimuovere container

```
# Rimuovi container fermo
$ docker rm webserver

# Rimuovi container in esecuzione (forza)
$ docker rm -f webserver

# Rimuovi più container
$ docker rm container1 container2 container3

# Rimuovi tutti container fermati
$ docker container prune

# Rimuovi tutti container (anche in esecuzione)
$ docker rm -f $(docker ps -aq)
```

2.6 Logs e Debugging

2.6.1 Visualizzare log

```
# Log completi
$ docker logs webserver

# Segui log in real-time (come tail -f)
$ docker logs -f webserver

# Ultime N righe
$ docker logs --tail 100 webserver

# Log con timestamp
$ docker logs -t webserver
2025-11-15T10:30:15.123456789Z 172.17.0.1 - - [15/Nov/2025:10:30:15
    +0000] "GET / HTTP/1.1" 200

# Log da un certo tempo
$ docker logs --since 10m webserver
$ docker logs --since 2025-11-15T10:00:00 webserver
```

2.6.2 Eseguire comandi in container running

```
# Comando singolo
$ docker exec webserver ls /etc/nginx
conf.d
fastcgi.conf
mime.types
```

```
nginx.conf

# Shell interattiva
$ docker exec -it webserver bash
root@a1b2c3d4e5f6:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   8892  5432 ?        Ss   10:00   0:00 nginx: master
nginx       29  0.0  0.0   9316  2876 ?        S    10:00   0:00 nginx: worker

root@a1b2c3d4e5f6:/# exit
```

2.6.3 Inspect: Informazioni dettagliate

```
# Tutte le informazioni
$ docker inspect webserver

# Estrai IP address
$ docker inspect webserver | jq '.[0].NetworkSettings.IPAddress'
"172.17.0.2"

# Estrai variabili ambiente
$ docker inspect webserver | jq '.[0].Config.Env'
[
  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
  "NGINX_VERSION=1.23.1"
]

# Template Go (built-in)
$ docker inspect --format='{{.State.Status}}' webserver
running

$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' webserver
172.17.0.2
```

2.6.4 Stats: Monitoraggio risorse

```
# Statistiche real-time (come top)
$ docker stats

CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
a1b2c3d4e5f6   webserver     0.01%     5.234MiB / 7.775GiB   0.07%     1.23kB / 656B     12.3MB / 0B       3
f6e5d4c3b2a1   redis-cache   0.15%     12.45MiB / 7.775GiB   0.16%     5.67kB / 2.34kB   45.6MB / 1.23MB   5

# Stats di un singolo container
$ docker stats webserver

# Formato custom
$ docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

2.6.5 Events: Monitorare eventi Docker

```
# Stream eventi real-time
$ docker events

2025-11-15T10:35:20.123456789+00:00 container create a1b2c3d4e5f6 (image
    =nginx, name=webserver)
2025-11-15T10:35:20.456789123+00:00 container start a1b2c3d4e5f6 (image=
    nginx, name=webserver)
2025-11-15T10:40:15.789123456+00:00 container stop a1b2c3d4e5f6 (image=
    nginx, name=webserver)

# Filtro per tipo
$ docker events --filter 'type=container'

# Filtro per evento
$ docker events --filter 'event=start'
```

2.7 Pulizia Risorse

2.7.1 Disk usage

```
$ docker system df

TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          15        5         2.5GB     1.8GB (72%)
Containers      20        3         150MB     145MB (96%)
Local Volumes   10        2         500MB     450MB (90%)
Build Cache     50        0         1.2GB     1.2GB (100%)
```

2.7.2 Prune: Pulizia automatica

```
# Rimuovi container fermati
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure? [y/N] y
Deleted Containers:
9876543210ab
5432167890cd
Total reclaimed space: 125MB

# Rimuovi immagini non usate
$ docker image prune -a

# Rimuovi volumi non usati
$ docker volume prune

# Rimuovi reti non usate
$ docker network prune

# ATTENZIONE: Pulizia totale
$ docker system prune -a --volumes
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all volumes not used by at least one container
  - all images without at least one container associated to them
  - all build cache
Are you sure? [y/N]
```

Best Practices

Best Practices

1. **Nomi significativi**: Usa `-name` per identificare facilmente i container
2. **Tag espliciti**: Evita `latest`, specifica versione (`nginx:1.23`)
3. **Cleanup regolare**: Esegui `docker system prune` periodicamente
4. **Limita risorse**: Usa `-cpus` e `-memory` in produzione
5. **Health checks**: Configura controlli di salute per monitoring
6. **Logging centralizzato**: Usa driver di log (syslog, json-file, fluentd)
7. **Restart policy**: Configura `-restart` per alta disponibilità
8. **Non usare exec per deployment**: Usa docker-compose o orchestratori

Errori Comuni

Attenzione

Problemi frequenti:

1. **Porta già in uso**

```
Error: Bind for 0.0.0.0:8080 failed: port is already allocated
```

   Soluzione: Cambia porta host o ferma processo che la occupa

2. **Permessi negati**

```
Got permission denied while trying to connect to the Docker
    daemon socket
```

   Soluzione: Aggiungi utente al gruppo docker o usa sudo

3. **Container esce immediatamente**

```
$ docker ps    # Container non appare
```

   Soluzione: Controlla log con `docker logs`, il processo principale è terminato

4. **Immagine non trovata**

```
Unable to find image 'myapp:latest' locally
Error: pull access denied, repository does not exist
```

   Soluzione: Verifica nome immagine/tag o fai pull esplicito

Esercizi

1. Installa Docker sul tuo sistema e verifica con `docker -version`
2. Esegui un container Nginx:
   - Mappa porta 8080 -> 80
   - Assegna nome "mio-nginx"
   - Verifica accesso con browser
3. Crea un container MySQL:
   - Password root: "mysecret"
   - Database: "testdb"
   - Connetti con `docker exec` e crea una tabella
4. Esegui container Ubuntu interattivo:
   - Installa `curl` dentro il container
   - Testa connessione a un sito esterno
   - Esci senza fermarlo (Ctrl+P, Ctrl+Q)
   - Riconnettiti con `docker attach`
5. Monitoring:
   - Lancia 5 container nginx
   - Monitora con `docker stats`
   - Identifica quello che usa più RAM
   - Ferma tutti e rimuovili
6. Cleanup:
   - Controlla spazio con `docker system df`
   - Rimuovi container e immagini inutilizzate
   - Verifica spazio recuperato

Quiz di Verifica

1. Quale flag di `docker run` esegue il container in background?
   - a) -b
   - b) -d
   - c) –background
   - d) -detach
2. Come vedere i log di un container in real-time?
3. Qual è la differenza tra `docker stop` e `docker kill`?
4. Come rimuovere tutti i container fermati con un solo comando?
5. **Vero/Falso**: `docker ps` mostra anche i container fermati.

Riepilogo Concetti Chiave

Concetti Fondamentali

- `docker run`: Crea ed esegue container da immagine
- `docker ps`: Lista container (aggiungere -a per tutti)
- `docker images`: Mostra immagini locali
- `docker stop/start/restart`: Gestione ciclo di vita
- `docker logs`: Visualizza output container
- `docker exec`: Esegui comandi in container running
- `docker inspect`: Informazioni dettagliate JSON
- `docker stats`: Monitoraggio risorse real-time
- `docker system prune`: Pulizia risorse inutilizzate

Prossimi Passi

Nel prossimo capitolo esploreremo:

- Dockerfile: creare immagini custom
- Istruzioni FROM, RUN, COPY, CMD, ENTRYPOINT
- Multi-stage builds per ottimizzazione
- Best practices per immagini efficienti e sicure

Riferimenti

- Docker CLI Reference: https://docs.docker.com/engine/reference/commandline/cli/
- Docker Run Reference: https://docs.docker.com/engine/reference/run/
- Dockerfile Best Practices: https://docs.docker.com/develop/dev-best-practices/
- Docker Hub: https://hub.docker.com/

Capitolo 3

Dockerfile: Creare Immagini Custom

Introduzione

Il Dockerfile è un file di testo che contiene le istruzioni per costruire un'immagine Docker. Questo capitolo copre la sintassi, le istruzioni principali, best practices e tecniche avanzate come multi-stage builds.

Obiettivi di apprendimento

- Scrivere Dockerfile per diverse applicazioni
- Comprendere FROM, RUN, COPY, CMD, ENTRYPOINT, ENV
- Ottimizzare immagini con multi-stage builds
- Applicare best practices per efficienza e sicurezza
- Gestire cache dei layer per build veloci

3.1 Cos'è un Dockerfile

3.1.1 Definizione

Un **Dockerfile** è uno script testuale che automatizza la creazione di un'immagine Docker. Ogni istruzione crea un nuovo layer nell'immagine finale.

3.1.2 Struttura base

Listing 3.1: Dockerfile minimale

```
# Commento: immagine base
FROM ubuntu:22.04

# Installa software
RUN apt-get update && apt-get install -y python3

# Copia applicazione
COPY app.py /app/app.py

# Comando di avvio
CMD ["python3", "/app/app.py"]
```

3.1.3 Build dell'immagine

```
# Build con tag
$ docker build -t myapp:v1.0 .

# Build con nome e path specifico
$ docker build -t myapp:latest -f Dockerfile.prod .

# Build senza cache
$ docker build --no-cache -t myapp .
```

3.2 Istruzioni Fondamentali

3.2.1 FROM: Immagine Base

La prima istruzione di ogni Dockerfile. Specifica l'immagine di partenza.

Listing 3.2: Esempi di FROM

```
# Immagine ufficiale Ubuntu
FROM ubuntu:22.04

# Immagine Alpine (minimalista, 5MB)
FROM alpine:3.18

# Immagine specifica per linguaggio
FROM python:3.11-slim
FROM node:18-alpine
FROM openjdk:17-jdk-slim

# Multi-stage: usa alias
FROM golang:1.21 AS builder
FROM nginx:alpine AS production

# Scratch: immagine vuota (per binari statici)
FROM scratch
```

Nota

Alpine Linux è popolare per container perché:

- Dimensione minima: 5MB vs 70MB Ubuntu
- Sicurezza: superficie d'attacco ridotta
- Performance: avvio rapido
- Attenzione: usa musl libc invece di glibc (possibili incompatibilità)

3.2.2 RUN: Eseguire Comandi

Esegue comandi durante la build dell'immagine. Ogni RUN crea un nuovo layer.

Listing 3.3: Sintassi RUN

```
# Shell form (eseguita in /bin/sh -c)
RUN apt-get update && apt-get install -y curl

```

```
# Exec form (preferita, no shell processing)
RUN ["apt-get", "update"]
RUN ["apt-get", "install", "-y", "nginx"]

# Multi-line con backslash
RUN apt-get update && \
    apt-get install -y \
        curl \
        vim \
        git && \
    rm -rf /var/lib/apt/lists/*
```

Best practice: Minimizzare layer

Listing 3.4: Esempio SBAGLIATO (3 layer)

```
# Anti-pattern: ogni RUN crea un layer
RUN apt-get update
RUN apt-get install -y curl
RUN rm -rf /var/lib/apt/lists/*
```

Listing 3.5: Esempio CORRETTO (1 layer)

```
# Best practice: combina in un singolo RUN
RUN apt-get update && \
    apt-get install -y curl && \
    rm -rf /var/lib/apt/lists/*
```

3.2.3 COPY e ADD: Copiare File

COPY: Copia Semplice

Listing 3.6: Esempi COPY

```
# Copia file singolo
COPY app.py /app/app.py

# Copia directory
COPY src/ /app/src/

# Copia multipli file
COPY package.json package-lock.json /app/

# Copia con pattern
COPY *.py /app/

# Cambia ownership
COPY --chown=user:group app.py /app/
```

ADD: Copia Avanzata

Listing 3.7: ADD con funzionalità extra

```
# Come COPY ma con auto-extract di tar
ADD archive.tar.gz /app/

# Download da URL (SCONSIGLIATO, preferire RUN curl)
ADD https://example.com/file.txt /app/
```

Attenzione

Preferisci COPY a ADD tranne quando serve auto-extraction di archivi tar. ADD ha comportamenti impliciti che possono confondere.

3.2.4 CMD: Comando Predefinito

Specifica il comando di default quando il container viene eseguito.

Listing 3.8: Forme di CMD

```
# Exec form (preferita)
CMD ["python3", "app.py"]
CMD ["nginx", "-g", "daemon off;"]

# Shell form
CMD python3 app.py

# Come parametri a ENTRYPOINT
CMD ["--help"]
```

Caratteristiche:

- Solo l'ultimo CMD nel Dockerfile è effettivo
- Può essere sovrascritto da `docker run`
- Non eseguito durante build, solo a runtime

Listing 3.9: Override CMD

```
# Usa CMD del Dockerfile
$ docker run myapp

# Sovrascrive CMD
$ docker run myapp python3 script2.py
```

3.2.5 ENTRYPOINT: Punto di Ingresso

Configura il container come eseguibile.

Listing 3.10: ENTRYPOINT vs CMD

```
# Solo ENTRYPOINT
FROM alpine
ENTRYPOINT ["ping"]
CMD ["localhost"]

# Build e run
$ docker build -t pinger .
$ docker run pinger              # ping localhost
$ docker run pinger google.com   # ping google.com
```

Differenze CMD vs ENTRYPOINT:

Listing 3.11: Pattern comune: ENTRYPOINT + CMD

```
FROM python:3.11-slim
WORKDIR /app
COPY app.py .
```

Aspetto	**CMD**	**ENTRYPOINT**
Override	Facile (`docker run img cmd`)	Richiede `-entrypoint`
Scopo	Comando di default	Eseguibile fisso
Combinazione	Parametri per ENTRYPOINT	Comando principale

Tabella 3.1: CMD vs ENTRYPOINT

```

ENTRYPOINT ["python3", "app.py"]
CMD ["--help"]

# docker run myapp           -> python3 app.py --help
# docker run myapp --serve   -> python3 app.py --serve
```

3.2.6 ENV: Variabili d'Ambiente

Listing 3.12: Definire variabili ambiente

```
# Sintassi key=value
ENV NODE_ENV=production
ENV APP_PORT=3000

# Sintassi vecchia (deprecata)
ENV NODE_ENV production

# Multiple env vars
ENV PORT=8080 \
    DEBUG=false \
    LOG_LEVEL=info

# Usare in RUN
ENV APP_DIR=/app
RUN mkdir -p $APP_DIR
WORKDIR $APP_DIR
```

ENV vs ARG:

- **ENV**: Persiste nel container runtime
- **ARG**: Solo durante build

3.2.7 ARG: Argomenti di Build

Listing 3.13: Usare ARG per build parametrizzata

```
# Definisci ARG con default
ARG PYTHON_VERSION=3.11
FROM python:${PYTHON_VERSION}-slim

ARG APP_ENV=development
RUN if [ "$APP_ENV" = "production" ]; then \
        pip install --no-cache-dir gunicorn; \
    fi

# Build con override
# $ docker build --build-arg PYTHON_VERSION=3.9 .
# $ docker build --build-arg APP_ENV=production .
```

3.2.8 WORKDIR: Directory di Lavoro

Listing 3.14: Impostare working directory

```
# Crea directory se non esiste
WORKDIR /app

# Path relativo (relativi al WORKDIR precedente)
WORKDIR /usr
WORKDIR local
WORKDIR bin
RUN pwd  # Output: /usr/local/bin

# Best practice: usa WORKDIR invece di RUN cd
# SBAGLIATO
RUN cd /app && python app.py

# CORRETTO
WORKDIR /app
RUN python app.py
```

3.2.9 EXPOSE: Documentare Porte

Listing 3.15: Dichiarare porte

```
# Documenta porte usate
EXPOSE 80
EXPOSE 443
EXPOSE 3000/tcp
EXPOSE 53/udp

# EXPOSE è solo documentazione!
# Devi comunque fare -p al run
# $ docker run -p 8080:80 myapp
```

3.2.10 VOLUME: Punti di Montaggio

Listing 3.16: Definire volumi

```
# Crea mount point
VOLUME /data
VOLUME ["/var/log", "/var/db"]

# Esempio: database
FROM mysql:8.0
VOLUME /var/lib/mysql

# Al run, Docker crea volume anonimo se non specificato
# $ docker run -v mydata:/var/lib/mysql mysql
```

3.2.11 USER: Cambiare Utente

Listing 3.17: Eseguire come utente non-root

```
# Crea utente
RUN groupadd -r appuser && \
    useradd -r -g appuser appuser

# Crea directory con ownership corretta
RUN mkdir -p /app && chown -R appuser:appuser /app

# Cambia utente per istruzioni successive
USER appuser

WORKDIR /app
COPY --chown=appuser:appuser . .

CMD ["python3", "app.py"]
```

Nota

Sicurezza: Eseguire container come root è un rischio. Usa sempre USER per applicazioni in produzione.

3.2.12 LABEL: Metadati

Listing 3.18: Aggiungere metadata

```
LABEL maintainer="luca.campion@example.com"
LABEL version="1.0"
LABEL description="My awesome app"

# Multiple labels
LABEL org.opencontainers.image.title="MyApp" \
      org.opencontainers.image.version="1.0.0" \
      org.opencontainers.image.vendor="MyCompany"
```

3.3 Esempi Completi di Dockerfile

3.3.1 Applicazione Python Flask

Listing 3.19: Dockerfile per Flask app

```
FROM python:3.11-slim

# Metadata
LABEL maintainer="dev@example.com"

# Variabili ambiente
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    APP_HOME=/app

# Crea user non-root
RUN groupadd -r appuser && useradd -r -g appuser appuser

# Working directory
WORKDIR $APP_HOME

# Installa dipendenze sistema
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        curl \
        && rm -rf /var/lib/apt/lists/*

# Copia requirements e installa dipendenze Python
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copia applicazione
COPY --chown=appuser:appuser . .

# Cambia a utente non-root
USER appuser

# Esponi porta
EXPOSE 5000

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:5000/health || exit 1

# Comando di avvio
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "app:app"]
```

3.3.2 Applicazione Node.js

Listing 3.20: Dockerfile per Node.js app

```
FROM node:18-alpine

# Installa dumb-init per signal handling
RUN apk add --no-cache dumb-init

# Crea app directory
WORKDIR /usr/src/app

# Copia package files
COPY package*.json ./

# Installa dipendenze (production only)
RUN npm ci --only=production && npm cache clean --force

# Copia codice app
COPY . .

# Usa utente node built-in
USER node

# Esponi porta
EXPOSE 3000

# Usa dumb-init per gestire segnali
ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "server.js"]
```

3.3.3 Applicazione Go (Static Binary)

Listing 3.21: Dockerfile per Go app

```
FROM golang:1.21-alpine AS builder

WORKDIR /build

# Copia go mod files
COPY go.mod go.sum ./
RUN go mod download

# Copia source code
COPY . .

# Build static binary
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

# Final stage: usa scratch (immagine vuota)
FROM scratch

# Copia solo binary
COPY --from=builder /build/app /app

# Copia CA certificates per HTTPS
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/

EXPOSE 8080

ENTRYPOINT ["/app"]
```

3.4 Multi-Stage Builds

3.4.1 Concetto

I **multi-stage builds** permettono di usare più FROM in un Dockerfile, copiando solo gli artifact necessari nell'immagine finale.

Vantaggi:

- Immagini finali molto più piccole
- Separazione build tools da runtime
- Sicurezza: no source code in produzione
- Un solo Dockerfile per dev e prod

3.4.2 Esempio: Java Application

Listing 3.22: Multi-stage: Maven build + JRE runtime

```
# Stage 1: Build con Maven
FROM maven:3.9-eclipse-temurin-17 AS builder

WORKDIR /build

# Copia pom.xml e scarica dipendenze (layer cacheable)
COPY pom.xml .
RUN mvn dependency:go-offline

```

```
# Copia source e compila
COPY src ./src
RUN mvn package -DskipTests

# Stage 2: Runtime con JRE
FROM eclipse-temurin:17-jre-alpine

WORKDIR /app

# Copia solo JAR compilato dallo stage builder
COPY --from=builder /build/target/myapp.jar app.jar

# Utente non-root
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Confronto dimensioni:

- Single-stage (Maven+JDK): 650 MB
- Multi-stage (solo JRE): 180 MB
- Risparmio: 72%

3.4.3 Esempio: React Frontend

Listing 3.23: Multi-stage: npm build + nginx serve

```
# Stage 1: Build con Node.js
FROM node:18-alpine AS builder

WORKDIR /build

# Installa dipendenze
COPY package*.json ./
RUN npm ci

# Build production
COPY . .
RUN npm run build

# Stage 2: Serve con Nginx
FROM nginx:alpine

# Copia build output
COPY --from=builder /build/dist /usr/share/nginx/html

# Custom nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

# Nginx in foreground
CMD ["nginx", "-g", "daemon off;"]
```

3.4.4 Esempio: Python con compilazione C

Listing 3.24: Multi-stage: build dependencies + runtime

```
# Stage 1: Build con compiler
FROM python:3.11-slim AS builder

RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    g++ \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /build

COPY requirements.txt .
RUN pip wheel --no-cache-dir --wheel-dir /wheels -r requirements.txt

# Stage 2: Runtime senza compiler
FROM python:3.11-slim

COPY --from=builder /wheels /wheels

RUN pip install --no-cache-dir /wheels/* && rm -rf /wheels

WORKDIR /app
COPY . .

CMD ["python", "app.py"]
```

3.5 Ottimizzazione e Best Practices

3.5.1 Layer Caching

Docker cachea i layer se le istruzioni non cambiano.

Listing 3.25: SBAGLIATO: Invalida cache spesso

```
FROM python:3.11-slim

# Ogni modifica a qualsiasi file invalida tutto dopo
COPY . /app
RUN pip install -r /app/requirements.txt

CMD ["python", "/app/app.py"]
```

Listing 3.26: CORRETTO: Ottimizza cache

```
FROM python:3.11-slim

WORKDIR /app

# Copia solo requirements (cambia raramente)
COPY requirements.txt .
RUN pip install -r requirements.txt

# Copia codice app (cambia spesso)
COPY . .
```

```

CMD ["python", "app.py"]
```

3.5.2 Minimizzare Dimensioni Immagine

Strategie per Ridurre Dimensioni

1. **Usa immagini base minimali**
   - Alpine invece di Ubuntu: -60MB+
   - Slim/slim-bullseye per Python/Node: -30MB
   - Distroless per linguaggi compilati
2. **Multi-stage builds**: Solo runtime artifacts
3. **Combina RUN**: Meno layer
4. **Pulisci in stesso layer**:

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

5. **Usa .dockerignore**:

```
# .dockerignore
.git
node_modules
*.log
.env
```

6. **No package manager cache**:

```
# Python
RUN pip install --no-cache-dir -r requirements.txt

# Node
RUN npm ci && npm cache clean --force

# apt
RUN apt-get install -y curl && rm -rf /var/lib/apt/lists/*
```

3.5.3 Sicurezza

Attenzione

Security Best Practices:

1. **Non usare root**

```
USER appuser
```

2. **Scansiona immagini**

```
$ docker scan myapp:latest
$ trivy image myapp:latest
```

3. **Usa immagini ufficiali verificate**
4. **Aggiorna base images regolarmente**
5. **Non embeddare segreti**

```
# SBAGLIATO
ENV API_KEY=super_secret_123

# CORRETTO: usa secrets o env vars a runtime
$ docker run -e API_KEY=$(cat secret.txt) myapp
```

6. **Usa COPY invece di ADD**
7. **Specifica versioni esatte**

```
FROM python:3.11.5-slim  # Non :latest
```

3.5.4 File .dockerignore

Listing 3.27: .dockerignore example

```
# Version control
.git
.gitignore
.gitattributes

# Dependencies
node_modules
bower_components
vendor

# Build output
dist
build
target
*.pyc
__pycache__

# Logs
*.log
logs

# IDE
.vscode
.idea
*.swp

# Environment
.env
.env.local
*.pem

# Documentation
README.md
docs

```

```
# Tests
tests
*.test.js
```

Best Practices Riassunto

Dockerfile Best Practices

1. Usa immagini base ufficiali e minimali
2. Un processo per container
3. Ordina istruzioni per cache (meno variabili prima)
4. Combina RUN per ridurre layer
5. Multi-stage builds per immagini compatte
6. Non eseguire come root (USER)
7. Usa .dockerignore per escludere file
8. COPY preferito ad ADD
9. Specifica versioni esatte (no :latest in prod)
10. Pulisci cache in stesso layer
11. Health checks per monitoring
12. Metadata con LABEL

Errori Comuni

1. **Layer troppo grandi**: Non combinare comandi
2. **Cache invalidation**: COPY . . all'inizio
3. **Root user**: Rischio sicurezza
4. **Segreti nel Dockerfile**: Esposti in history
5. **:latest in produzione**: Non riproducibile
6. **Software inutile**: Aumenta superficie d'attacco
7. **File temporanei**: Non rimossi nello stesso RUN

Esercizi

1. Scrivi un Dockerfile per un'app Python Flask con:
   - Immagine base python:3.11-slim
   - User non-root
   - Requirements.txt cacheable
   - Health check su /health

2. Converti questo Dockerfile a multi-stage:

```
FROM node:18
COPY . /app
WORKDIR /app
RUN npm install && npm run build
CMD ["npm", "start"]
```

3. Ottimizza questa catena RUN (3 layer -> 1):

```
RUN apt-get update
RUN apt-get install -y curl vim
RUN rm -rf /var/lib/apt/lists/*
```

4. Crea un .dockerignore per un progetto Node.js

5. Build un'immagine Go che usa scratch e misura la dimensione finale

Quiz di Verifica

1. Qual è la differenza tra CMD e ENTRYPOINT?

2. Perché multi-stage builds riducono le dimensioni?

3. **Vero/Falso**: ARG persiste nel container runtime.

4. Quale istruzione crea un nuovo layer?

   - a) FROM
   - b) RUN
   - c) ENV
   - d) EXPOSE

5. Come evitare di invalidare la cache quando cambia il codice ma non le dipendenze?

Riepilogo

- **Dockerfile**: Script per automatizzare build immagini
- **FROM**: Immagine base
- **RUN**: Esegue comandi (build-time)
- **COPY**: Copia file nel container
- **CMD**: Comando default (runtime)
- **ENTRYPOINT**: Eseguibile principale
- **Multi-stage**: Ottimizzazione dimensioni
- **Cache**: Ordina istruzioni per massimizzare riuso
- **Sicurezza**: USER, scan, no secrets

Prossimi Passi

Nel prossimo capitolo esploreremo:

- Docker Compose per orchestrare multi-container
- File docker-compose.yml
- Services, networks, volumes
- Deploy stack completi

Riferimenti

- Dockerfile Reference: https://docs.docker.com/engine/reference/builder/
- Best Practices: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- Multi-stage Builds: https://docs.docker.com/build/building/multi-stage/
- Security: https://snyk.io/blog/10-docker-image-security-best-practices/

Capitolo 4

Docker Compose: Orchestrazione Multi-Container

Introduzione

Docker Compose è uno strumento per definire ed eseguire applicazioni Docker multi-container. Con un singolo file YAML puoi configurare tutti i servizi, reti e volumi della tua applicazione e avviarli con un comando.

Obiettivi di apprendimento

- Scrivere file docker-compose.yml
- Definire services, networks, volumes
- Orchestrare stack multi-container
- Gestire dipendenze tra servizi
- Usare variabili d'ambiente e secrets
- Deploy applicazioni complete

4.1 Cos'è Docker Compose

4.1.1 Definizione

Docker Compose è un tool per definire e gestire applicazioni multi-container usando un file YAML dichiarativo.

Vantaggi:

- Un file per tutta l'infrastruttura
- Versionabile con Git
- Riproducibile su qualsiasi ambiente
- Comandi semplici: up, down, logs
- Ideale per sviluppo locale e testing

4.1.2 Installazione

Listing 4.1: Docker Compose V2 (integrato in Docker)

```
# Già incluso in Docker Desktop (macOS/Windows)

# Linux: verifica versione
$ docker compose version
Docker Compose version v2.20.2

# Se manca, installa plugin
$ sudo apt-get install docker-compose-plugin
```

Nota

Compose V1 vs V2:

- V1: `docker-compose` (Python, standalone)
- V2: `docker compose` (Go plugin, integrato)
- V2 è più veloce e il futuro ufficiale

4.2 File docker-compose.yml

4.2.1 Struttura Base

Listing 4.2: docker-compose.yml minimale

```
version: '3.8'

services:
  web:
    image: nginx:alpine
    ports:
      - "8080:80"

  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: secret
```

Sezioni principali:

- `version`: Versione file format (3.8 è comune)
- `services`: Container da eseguire
- `networks`: Reti custom (opzionale)
- `volumes`: Volumi persistenti (opzionale)

4.2.2 Comandi Base

```
# Avvia tutti i servizi (detached)
$ docker compose up -d

```

```
# Build e avvia
$ docker compose up --build

# Ferma e rimuovi container
$ docker compose down

# Ferma e rimuovi anche volumi
$ docker compose down -v

# Lista servizi in esecuzione
$ docker compose ps

# Log di tutti i servizi
$ docker compose logs -f

# Log di un servizio specifico
$ docker compose logs -f web

# Esegui comando in un servizio
$ docker compose exec web sh

# Scala un servizio
$ docker compose up -d --scale web=3
```

4.3 Definire Services

4.3.1 Build da Dockerfile

Listing 4.3: Service con build custom

```
services:
  app:
    build:
      context: ./app
      dockerfile: Dockerfile
      args:
        - APP_ENV=production
    image: myapp:latest
    container_name: my-app
    restart: unless-stopped
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - API_KEY=${API_KEY}
    volumes:
      - ./app:/usr/src/app
      - /usr/src/app/node_modules
```

4.3.2 Usare Immagini Esistenti

Listing 4.4: Service con immagine da registry

```
services:
  redis:
    image: redis:7-alpine
    container_name: redis-cache
    restart: always
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data
    command: redis-server --appendonly yes

volumes:
  redis-data:
```

4.3.3 Opzioni Comuni dei Services

Opzione	Descrizione
`image`	Immagine da usare
`build`	Path Dockerfile per build custom
`container_name`	Nome container (default: progetto_servizio_1)
`ports`	Mapping porte host:container
`environment`	Variabili d'ambiente
`env_file`	File con env vars
`volumes`	Mount volumi o bind mounts
`networks`	Reti a cui connettere
`depends_on`	Dipendenze da altri servizi
`restart`	Policy restart (no/always/on-failure/unless-stopped)
`command`	Override CMD del Dockerfile
`healthcheck`	Controllo salute container
`labels`	Metadata key-value

Tabella 4.1: Opzioni principali dei services

4.4 Dipendenze tra Servizi

4.4.1 depends_on

Listing 4.5: Definire dipendenze

```
services:
  web:
    image: myapp
    depends_on:
      - db
      - redis

  db:
    image: postgres:15

  redis:
    image: redis:alpine
```

Comportamento:

- Compose avvia `db` e `redis` prima di `web`
- Non aspetta che i servizi siano "ready", solo che siano started
- Per aspettare readiness, serve health check

4.4.2 Health Checks e Readiness

Listing 4.6: Aspettare che DB sia pronto

```
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: secret
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  web:
    image: myapp
    depends_on:
      db:
        condition: service_healthy
```

4.5 Networks

4.5.1 Network di Default

Compose crea automaticamente una rete bridge per i servizi.

```
services:
  web:
    image: nginx
  app:
    image: myapp
  db:
    image: postgres

# Automaticamente:
# - Network "myproject_default" creata
# - Tutti i servizi connessi
# - Service discovery: web può raggiungere db via DNS "db"
```

4.5.2 Networks Custom

Listing 4.7: Definire reti multiple

```
services:
  frontend:
    image: react-app
    networks:
      - frontend-net

  api:
    image: node-api
    networks:
      - frontend-net
      - backend-net

  db:
    image: postgres
    networks:
      - backend-net

networks:
  frontend-net:
    driver: bridge
  backend-net:
    driver: bridge
    internal: true  # No accesso internet
```

Risultato:

- Frontend può chiamare API
- API può chiamare DB
- Frontend NON può chiamare DB direttamente
- DB non ha accesso internet (internal)

4.5.3 Network Esistente

```
services:
  app:
    image: myapp
    networks:
      - existing-network

networks:
  existing-network:
    external: true
```

4.6 Volumes

4.6.1 Named Volumes

Listing 4.8: Volumi gestiti da Docker

```
services:
  db:
    image: postgres:15
    volumes:
      - postgres-data:/var/lib/postgresql/data

  redis:
    image: redis:alpine
    volumes:
      - redis-data:/data

volumes:
  postgres-data:
  redis-data:
```

4.6.2 Bind Mounts

Listing 4.9: Mount directory host

```
services:
  web:
    image: nginx
    volumes:
      # Bind mount (path assoluto o relativo)
      - ./html:/usr/share/nginx/html
      - ./nginx.conf:/etc/nginx/nginx.conf:ro  # Read-only

  app:
    build: ./app
    volumes:
      # Hot reload per sviluppo
      - ./app:/usr/src/app
      # Named volume per node_modules
      - /usr/src/app/node_modules
```

4.6.3 Opzioni Volumi Avanzate

```
services:
  db:
    image: postgres
    volumes:
      - type: volume
        source: db-data
        target: /var/lib/postgresql/data
        volume:
          nocopy: true

      - type: bind
        source: ./backup
        target: /backup
        read_only: true

volumes:
  db-data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /mnt/database
```

4.7 Variabili d'Ambiente

4.7.1 File .env

Listing 4.10: .env file

```
# .env
POSTGRES_VERSION=15
POSTGRES_PASSWORD=mysecretpassword
APP_PORT=3000
NODE_ENV=production
```

Listing 4.11: Usare variabili da .env

```
services:
  db:
    image: postgres:${POSTGRES_VERSION}
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}

  app:
    build: ./app
    ports:
      - "${APP_PORT}:3000"
    environment:
      NODE_ENV: ${NODE_ENV}
```

4.7.2 env_file per Container

```
services:
  app:
    image: myapp
    env_file:
      - .env.common
      - .env.production

  db:
    image: postgres
    env_file: database.env
```

Listing 4.12: database.env

```
POSTGRES_USER=admin
POSTGRES_PASSWORD=secret
POSTGRES_DB=myapp
```

4.8 Esempi Completi

4.8.1 Stack LAMP (Linux, Apache, MySQL, PHP)

Listing 4.13: docker-compose.yml per LAMP

```
version: '3.8'

services:
  web:
    image: php:8.2-apache
    container_name: lamp-web
    restart: unless-stopped
    ports:
      - "8080:80"
    volumes:
      - ./www:/var/www/html
    networks:
      - lamp-net
    depends_on:
      - db

  db:
    image: mysql:8.0
    container_name: lamp-db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: myapp
      MYSQL_USER: user
      MYSQL_PASSWORD: userpass
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - lamp-net
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: lamp-phpmyadmin
    restart: unless-stopped
    ports:
      - "8081:80"
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
    networks:
      - lamp-net
    depends_on:
      db:
        condition: service_healthy

networks:
  lamp-net:
    driver: bridge

volumes:
  mysql-data:
```

4.8.2 Stack MERN (MongoDB, Express, React, Node)

Listing 4.14: docker-compose.yml per MERN

```
version: '3.8'

services:
  # Frontend React
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: mern-frontend
    restart: unless-stopped
    ports:
      - "3000:3000"
```

```yaml
    volumes:
      - ./frontend/src:/app/src
      - /app/node_modules
    environment:
      - REACT_APP_API_URL=http://localhost:5000
    networks:
      - mern-net
    depends_on:
      - backend

  # Backend Node.js/Express
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: mern-backend
    restart: unless-stopped
    ports:
      - "5000:5000"
    volumes:
      - ./backend:/app
      - /app/node_modules
    environment:
      - NODE_ENV=development
      - MONGO_URI=mongodb://mongodb:27017/myapp
      - JWT_SECRET=${JWT_SECRET}
    networks:
      - mern-net
    depends_on:
      mongodb:
        condition: service_healthy

  # Database MongoDB
  mongodb:
    image: mongo:7
    container_name: mern-mongodb
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=adminpass
    volumes:
      - mongo-data:/data/db
    networks:
      - mern-net
    healthcheck:
      test: ["CMD", "mongosh", "--eval", "db.adminCommand(’ping’)"]
      interval: 10s
      timeout: 5s
      retries: 5

networks:
  mern-net:
    driver: bridge

volumes:
  mongo-data:
```

4.8.3 Stack Microservizi con Nginx Reverse Proxy

Listing 4.15: Architettura microservizi

```
version: '3.8'

services:
  # Reverse Proxy
  nginx:
    image: nginx:alpine
    container_name: reverse-proxy
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./nginx/certs:/etc/nginx/certs:ro
    networks:
      - frontend-net
    depends_on:
      - auth-service
      - user-service
      - product-service

  # Authentication Service
  auth-service:
    build: ./services/auth
    container_name: auth-service
    restart: unless-stopped
    expose:
      - "3001"
    environment:
      - DB_HOST=auth-db
      - REDIS_HOST=redis
    networks:
      - frontend-net
      - auth-backend-net
    depends_on:
      - auth-db
      - redis

  # User Service
  user-service:
    build: ./services/user
    container_name: user-service
    restart: unless-stopped
    expose:
      - "3002"
    environment:
      - DB_HOST=user-db
    networks:
      - frontend-net
      - user-backend-net
    depends_on:
      - user-db

  # Product Service
  product-service:
    build: ./services/product
    container_name: product-service
    restart: unless-stopped
    expose:
      - "3003"
    environment:
      - DB_HOST=product-db
    networks:
      - frontend-net
      - product-backend-net
    depends_on:
      - product-db

  # Databases
  auth-db:
    image: postgres:15
    container_name: auth-db
    restart: unless-stopped
    environment:
      POSTGRES_DB: auth
      POSTGRES_PASSWORD: authpass
    volumes:
      - auth-db-data:/var/lib/postgresql/data
    networks:
      - auth-backend-net

  user-db:
    image: postgres:15
    container_name: user-db
    restart: unless-stopped
    environment:
      POSTGRES_DB: users
      POSTGRES_PASSWORD: userpass
    volumes:
      - user-db-data:/var/lib/postgresql/data
    networks:
      - user-backend-net

  product-db:
    image: postgres:15
    container_name: product-db
    restart: unless-stopped
    environment:
      POSTGRES_DB: products
      POSTGRES_PASSWORD: productpass
    volumes:
      - product-db-data:/var/lib/postgresql/data
    networks:
      - product-backend-net

  # Cache Redis
  redis:
    image: redis:7-alpine
    container_name: redis
    restart: unless-stopped
    networks:
      - auth-backend-net

```

```
networks:
  frontend-net:
  auth-backend-net:
    internal: true
  user-backend-net:
    internal: true
  product-backend-net:
    internal: true

volumes:
  auth-db-data:
  user-db-data:
  product-db-data:
```

4.8.4 Monitoring Stack (Prometheus + Grafana)

Listing 4.16: Stack monitoring

```
version: '3.8'

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    restart: unless-stopped
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
    networks:
      - monitoring

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    restart: unless-stopped
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-data:/var/lib/grafana
      - ./grafana/dashboards:/etc/grafana/provisioning/dashboards
    networks:
      - monitoring
    depends_on:
      - prometheus

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    restart: unless-stopped
    expose:
      - "9100"
    networks:
      - monitoring

  cadvisor:
    image: gcr.io/cadvisor/cadvisor:latest
    container_name: cadvisor
    restart: unless-stopped
    expose:
      - "8080"
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    networks:
      - monitoring

networks:
  monitoring:
    driver: bridge

volumes:
  prometheus-data:
  grafana-data:
```

4.9 Comandi Avanzati

4.9.1 Override Files

```
# docker-compose.yml (base)
# docker-compose.override.yml (dev)
# docker-compose.prod.yml (production)

# Automatico: usa override se esiste
$ docker compose up

# Specifica file multipli
$ docker compose -f docker-compose.yml -f docker-compose.prod.yml up
```

Listing 4.17: docker-compose.override.yml

```
# Override per sviluppo
services:
  web:
    volumes:
      - ./app:/app  # Hot reload
    environment:
      - DEBUG=true
```

4.9.2 Scale Services

```
# Scala servizio web a 3 repliche
$ docker compose up -d --scale web=3

```

```
# Verifica
$ docker compose ps
NAME           IMAGE    COMMAND    STATUS    PORTS
project-web-1  nginx    ...        Up        0.0.0.0:8081->80/tcp
project-web-2  nginx    ...        Up        0.0.0.0:8082->80/tcp
project-web-3  nginx    ...        Up        0.0.0.0:8083->80/tcp
```

4.9.3 Logs e Monitoring

```
# Log real-time di tutti i servizi
$ docker compose logs -f

# Log di un servizio specifico
$ docker compose logs -f web

# Ultime 100 righe
$ docker compose logs --tail=100

# Top processes
$ docker compose top

# Stats risorse
$ docker stats $(docker compose ps -q)
```

Best Practices

Best Practices Docker Compose

1. **Versionamento**: Commit docker-compose.yml in Git
2. **File .env**: Non committare secrets (usa .gitignore)
3. **Named volumes**: Preferisci a bind mounts per produzione
4. **Health checks**: Definisci per tutti i servizi critici
5. **Restart policies**: Usa `unless-stopped` in prod
6. **Resource limits**:

   ```
   services:
     web:
       deploy:
         resources:
           limits:
             cpus: '0.5'
             memory: 512M
   ```

7. **Networks isolate**: Separa frontend/backend
8. **Container names**: Usa nomi espliciti
9. **Override files**: Separa dev/prod config
10. **Documentazione**: README con istruzioni setup

Errori Comuni

Attenzione

1. **Bind mounts in produzione**: Preferisci named volumes
2. **Port conflicts**: Verifica porte non occupate

```
Error: Bind for 0.0.0.0:3000 failed: port is already allocated
```

3. **depends_on senza health check**: Non garantisce readiness
4. **Secrets in chiaro**: Usa Docker secrets o vault
5. **Network non specificata**: Servizi non comunicano
6. **Volumi non persistenti**: Dati persi con down -v

Esercizi

1. Crea uno stack WordPress + MySQL con docker-compose
2. Implementa un'app Node.js + PostgreSQL + Redis:
   - Health check su database
   - Bind mount per hot reload
   - Named volume per dati PostgreSQL
3. Setup ambiente microservizi:
   - 3 servizi API (users, orders, products)
   - Nginx reverse proxy
   - Database separato per ogni servizio
   - Redis condiviso per caching
4. Crea file override per sviluppo e produzione
5. Implementa stack monitoring con Prometheus + Grafana

Quiz di Verifica

1. Qual è il comando per avviare tutti i servizi in background?
2. Cosa fa `depends_on`? Garantisce che il servizio sia pronto?
3. Come passare variabili d'ambiente a un servizio?
4. Qual è la differenza tra named volume e bind mount?
5. Come vedere i log di un singolo servizio in real-time?

Riepilogo

- **Docker Compose**: Orchestrazione multi-container con YAML
- **Services**: Definizione container e configurazione
- **Networks**: Isolamento e comunicazione tra servizi
- **Volumes**: Persistenza dati
- **depends_on**: Dipendenze e ordine avvio
- **Health checks**: Verificare readiness servizi
- **.env**: Gestione variabili d'ambiente
- **Override**: Configurazioni multiple dev/prod

Prossimi Passi

Nel prossimo capitolo esploreremo:

- Networking Docker approfondito (bridge, host, overlay)
- Gestione avanzata volumi
- Persistenza dati e backup
- Service discovery e load balancing

Riferimenti

- Compose File Reference: `https://docs.docker.com/compose/compose-file/`
- Compose CLI: `https://docs.docker.com/compose/reference/`
- Compose Samples: `https://github.com/docker/awesome-compose`

Capitolo 5

Networking e Volumes

Introduzione

Il networking e la gestione dei volumi sono fondamentali per container che devono comunicare tra loro e persistere dati. Questo capitolo esplora i diversi driver di rete Docker, il service discovery, e le strategie di gestione volumi per garantire persistenza e backup.

Obiettivi di apprendimento

- Comprendere i driver di rete: bridge, host, overlay, none
- Creare reti custom e isolare container
- Configurare port mapping e service discovery
- Gestire volumi per persistenza dati
- Implementare backup e restore di volumi
- Usare bind mounts e tmpfs appropriatamente

5.1 Docker Networking

5.1.1 Concetti Fondamentali

Docker usa **network drivers** per fornire networking ai container:

- **bridge**: Default, rete privata su un singolo host
- **host**: Usa direttamente il network stack dell'host
- **overlay**: Multi-host networking (Swarm/Kubernetes)
- **macvlan**: Assegna MAC address ai container
- **none**: Nessun networking

5.1.2 Comandi Base

```
# Lista reti
$ docker network ls
NETWORK ID      NAME      DRIVER    SCOPE
abc123def456    bridge    bridge    local
```

```
123456789abc   host      host      local
def456abc123   none      null      local

# Crea rete custom
$ docker network create mynetwork
$ docker network create --driver bridge my-bridge-net

# Ispeziona rete
$ docker network inspect mynetwork

# Connetti container a rete
$ docker network connect mynetwork container1

# Disconnetti
$ docker network disconnect mynetwork container1

# Rimuovi rete
$ docker network rm mynetwork

# Pulisci reti non usate
$ docker network prune
```

5.2 Bridge Network

5.2.1 Default Bridge

Quando avvii un container senza specificare `-network`, usa la rete `bridge` di default.

Listing 5.1: Container su bridge default

```
# Avvia due container
$ docker run -d --name container1 alpine sleep 1000
$ docker run -d --name container2 alpine sleep 1000

# Verifica network
$ docker inspect container1 | jq '.[0].NetworkSettings.Networks'
{
  "bridge": {
    "IPAddress": "172.17.0.2",
    ...
  }
}

# Comunicazione via IP (funziona)
$ docker exec container1 ping -c 2 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.123 ms

# Comunicazione via nome (NON funziona su default bridge!)
$ docker exec container1 ping container2
ping: bad address 'container2'
```

Limitazioni default bridge:

- No automatic service discovery (DNS)
- Comunicazione solo via IP
- Tutti i container vedono tutti gli altri

5.2.2 User-Defined Bridge

Best practice: Usa sempre reti custom per service discovery automatico.

Listing 5.2: Custom bridge network

```
# Crea rete custom
$ docker network create --driver bridge my-app-net

# Avvia container su rete custom
$ docker run -d --name web --network my-app-net nginx
$ docker run -d --name db --network my-app-net postgres

# Service discovery via DNS (funziona!)
$ docker exec web ping -c 2 db
PING db (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.089 ms

# Inspect network
$ docker network inspect my-app-net
[
    {
        "Name": "my-app-net",
        "Driver": "bridge",
        "Containers": {
            "abc123": {
                "Name": "web",
                "IPv4Address": "172.18.0.2/16"
            },
            "def456": {
                "Name": "db",
                "IPv4Address": "172.18.0.3/16"
            }
        }
    }
]
```

5.2.3 Configurazione Bridge Avanzata

Listing 5.3: Opzioni custom bridge

```
# Subnet e gateway custom
$ docker network create \
  --driver bridge \
  --subnet 192.168.100.0/24 \
  --gateway 192.168.100.1 \
  --ip-range 192.168.100.0/25 \
  my-custom-net

# IP statico per container
$ docker run -d \
  --name web \
  --network my-custom-net \
  --ip 192.168.100.10 \
  nginx
```

5.3 Host Network

Il container usa direttamente il network stack dell'host, senza isolamento.

Listing 5.4: Host network mode

```
# Container usa network dell'host
$ docker run -d --name nginx-host --network host nginx

# Nginx ascolta su porta 80 dell'HOST (non del container)
$ curl http://localhost:80
<!DOCTYPE html>
<html>
<title>Welcome to nginx!</title>
...

# No port mapping necessario (-p non serve)
```

Vantaggi:

- Performance ottimale (no NAT overhead)
- Accesso diretto a tutte le interfacce host

Svantaggi:

- Nessun isolamento di rete
- Port conflicts se più container usano stessa porta
- Non funziona su Docker Desktop (macOS/Windows)

Quando usarlo:

- Performance critiche (monitoring, load balancer)
- Container che gestisce tutto il network stack (VPN, firewall)

5.4 Overlay Network

Per multi-host networking (Docker Swarm, Kubernetes).

Listing 5.5: Overlay network (Swarm mode)

```
# Inizializza Swarm
$ docker swarm init

# Crea overlay network
$ docker network create \
  --driver overlay \
  --attachable \
  my-overlay-net

# Deploy servizio su overlay
$ docker service create \
  --name web \
  --network my-overlay-net \
  --replicas 3 \
  nginx

# Container su host diversi possono comunicare
```

Caratteristiche:

- Comunicazione tra container su host fisici diversi
- Encryption opzionale del traffico
- Service discovery integrato
- Load balancing automatico

5.5 None Network

Nessun networking, completo isolamento.

```
# Container isolato
$ docker run -d --name isolated --network none alpine sleep 1000

# Verifica: solo loopback
$ docker exec isolated ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    inet 127.0.0.1/8 scope host lo
```

Quando usarlo:

- Elaborazione dati sensibili senza accesso rete
- Testing isolato
- Container che comunicano solo via volumi condivisi

5.6 Port Mapping

5.6.1 Pubblicare Porte

```
# Porta specifica: host:container
$ docker run -d -p 8080:80 nginx

# Porta random host
$ docker run -d -p 80 nginx
$ docker ps  # Vedi porta assegnata (es. 0.0.0.0:32768->80/tcp)

# Multiple porte
$ docker run -d \
  -p 8080:80 \
  -p 8443:443 \
  nginx

# IP specifico host
$ docker run -d -p 127.0.0.1:8080:80 nginx

# UDP
$ docker run -d -p 53:53/udp dns-server
```

5.6.2 Port Binding vs Expose

Listing 5.6: Differenza EXPOSE vs -p

```
# Dockerfile: EXPOSE documenta porta
EXPOSE 80

# Run: -p pubblica effettivamente
$ docker run -p 8080:80 myapp

# Senza -p, porta non accessibile dall'host
$ docker run myapp  # Porta 80 non raggiungibile esternamente
```

5.7 Service Discovery

5.7.1 DNS Automatico

Su reti custom, Docker fornisce DNS automatico.

Listing 5.7: Service discovery example

```
# Crea rete
$ docker network create app-net

# Servizio backend
$ docker run -d \
  --name api \
  --network app-net \
  myapi

# Servizio frontend può chiamare backend via nome
$ docker run -d \
  --name frontend \
  --network app-net \
  -e API_URL=http://api:3000 \
  myfrontend

# Frontend può risolvere "api" via DNS
$ docker exec frontend nslookup api
Server:  127.0.0.11
Address: 127.0.0.11:53

Name:    api
Address: 172.20.0.2
```

5.7.2 Network Aliases

```
# Multipli alias per stesso container
$ docker run -d \
  --name db \
  --network app-net \
  --network-alias database \
  --network-alias postgres \
  postgres

# Raggiungibile via db, database, o postgres
$ docker exec frontend ping database
$ docker exec frontend ping postgres
```

5.8 Isolamento e Sicurezza

5.8.1 Reti Multiple

Listing 5.8: Segmentare applicazione

```
# Rete frontend
$ docker network create frontend-net

# Rete backend (internal, no internet)
$ docker network create --internal backend-net

# Frontend: accessibile esternamente
$ docker run -d \
  --name web \
  --network frontend-net \
  -p 80:80 \
  nginx

# API: su entrambe le reti
$ docker run -d \
  --name api \
  --network frontend-net \
  nginx-api

$ docker network connect backend-net api

# Database: solo backend (non raggiungibile da internet)
$ docker run -d \
  --name db \
  --network backend-net \
  postgres
```

Architettura:

```
Internet
   |
[web] ----+
          |
      [frontend-net]
          |
        [api]
          |
      [backend-net]
          |
        [db]
```

5.8.2 Firewall e IPTables

Docker modifica automaticamente iptables per port mapping.

```
# Visualizza regole Docker
$ sudo iptables -t nat -L -n

```

```
# Disabilita modifica iptables (daemon.json)
{
  "iptables": false
}
```

5.9 Docker Volumes

5.9.1 Tipi di Persistenza

1. **Volumes**: Gestiti da Docker, best practice
2. **Bind mounts**: Directory host montate nel container
3. **tmpfs mounts**: Memoria RAM, non persistente

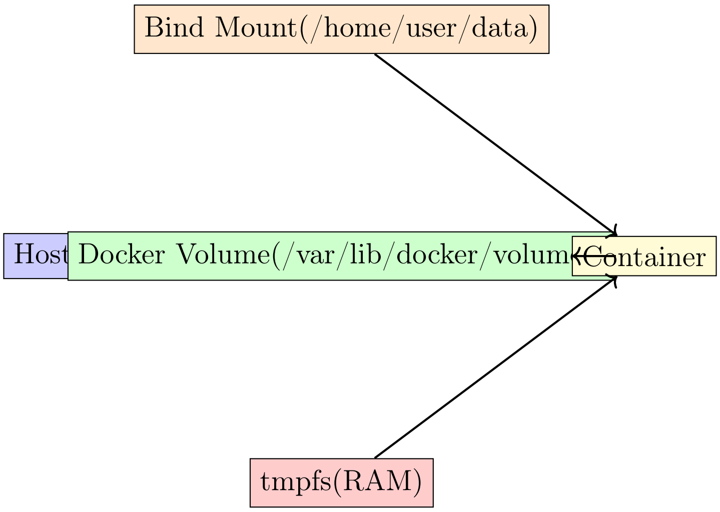
Bind Mount(/home/user/data)
Host
Docker Volume(/var/lib/docker/volum
Container
tmpfs(RAM)

Figura 5.1: Tipi di mount in Docker

5.9.2 Named Volumes

```
# Crea volume
$ docker volume create mydata

# Lista volumi
$ docker volume ls
DRIVER    VOLUME NAME
local     mydata

# Ispeziona
$ docker volume inspect mydata
[
    {
        "Name": "mydata",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/mydata/_data",
        "Labels": {},
        "Scope": "local"
    }
]
```

```

# Usa in container
$ docker run -d \
  --name db \
  -v mydata:/var/lib/postgresql/data \
  postgres

# Rimuovi volume
$ docker volume rm mydata

# Pulisci volumi non usati
$ docker volume prune
```

5.9.3 Bind Mounts

```
# Mount directory host
$ docker run -d \
  --name web \
  -v /home/user/html:/usr/share/nginx/html \
  nginx

# Path relativo (PWD)
$ docker run -d \
  --name app \
  -v $(pwd)/app:/app \
  myapp

# Read-only mount
$ docker run -d \
  --name web \
  -v $(pwd)/html:/usr/share/nginx/html:ro \
  nginx

# Sintassi --mount (più esplicita)
$ docker run -d \
  --mount type=bind,source=$(pwd)/app,target=/app \
  myapp
```

Bind Mounts vs Volumes:

Aspetto	Bind Mount	Volume
Path	Host filesystem	Docker area
Gestione	Manuale	Docker
Performance	Buona	Ottima (Linux)
Portabilità	Bassa	Alta
Backup	Manuale	Docker CLI
Uso	Sviluppo	Produzione

Tabella 5.1: Bind Mount vs Volume

5.9.4 tmpfs Mounts

Dati in RAM, non scritti su disco.

```
# Mount tmpfs
$ docker run -d \
  --name tmptest \
  --tmpfs /app/cache:rw,size=100m \
  myapp

# Sintassi --mount
$ docker run -d \
  --mount type=tmpfs,target=/app/cache,tmpfs-size=100m \
  myapp
```

Quando usare tmpfs:

- Dati sensibili (credenziali temporanee)
- Cache ad alte performance
- File temporanei che non devono persistere

5.10 Gestione Avanzata Volumi

5.10.1 Volume Drivers

```
# Driver locale (default)
$ docker volume create --driver local myvolume

# NFS volume
$ docker volume create \
  --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.1.1,rw \
  --opt device=:/path/to/dir \
  nfs-volume

# Cloud storage (plugin richiesto)
$ docker volume create \
  --driver rexray/s3fs \
  --opt=size=20 \
  s3-volume
```

5.10.2 Backup e Restore

Listing 5.9: Backup volume

```
# Backup volume in tar.gz
$ docker run --rm \
  -v mydata:/data \
  -v $(pwd):/backup \
  alpine \
  tar czf /backup/mydata-backup.tar.gz -C /data .

# Verifica backup
$ ls -lh mydata-backup.tar.gz
-rw-r--r-- 1 user user 1.5M Nov 15 10:00 mydata-backup.tar.gz
```

Listing 5.10: Restore volume

```
# Crea nuovo volume
$ docker volume create mydata-restored
```

```

# Restore da backup
$ docker run --rm \
  -v mydata-restored:/data \
  -v $(pwd):/backup \
  alpine \
  sh -c "cd /data && tar xzf /backup/mydata-backup.tar.gz"
```

5.10.3 Condivisione Volumi tra Container

```
# Container 1 scrive dati
$ docker run -d \
  --name writer \
  -v shared-data:/data \
  alpine \
  sh -c "while true; do date >> /data/log.txt; sleep 5; done"

# Container 2 legge dati
$ docker run -d \
  --name reader \
  -v shared-data:/data:ro \
  alpine \
  sh -c "tail -f /data/log.txt"

# Verifica
$ docker logs reader
Wed Nov 15 10:00:00 UTC 2025
Wed Nov 15 10:00:05 UTC 2025
...
```

5.10.4 Volumes from Container

```
# Data container
$ docker create -v /data --name data-container alpine

# App usa volumi da data-container
$ docker run -d \
  --name app1 \
  --volumes-from data-container \
  myapp

$ docker run -d \
  --name app2 \
  --volumes-from data-container \
  myapp
```

5.11 Esempi Pratici

5.11.1 Database con Persistenza

Listing 5.11: PostgreSQL production setup

```
# Crea rete e volume
$ docker network create db-net
$ docker volume create postgres-data

# Deploy PostgreSQL
$ docker run -d \
  --name postgres \
  --network db-net \
  --restart unless-stopped \
  -e POSTGRES_PASSWORD=secret \
  -v postgres-data:/var/lib/postgresql/data \
  -v $(pwd)/init.sql:/docker-entrypoint-initdb.d/init.sql:ro \
  postgres:15

# Backup automatico (cron job)
$ docker run --rm \
  --network db-net \
  -v postgres-data:/data \
  -v $(pwd)/backups:/backups \
  postgres:15 \
  pg_dump -h postgres -U postgres -F c -f /backups/db-$(date +%Y%m%d).
     dump
```

5.11.2 Sviluppo con Hot Reload

Listing 5.12: Node.js development

```
# Bind mount per hot reload
$ docker run -d \
  --name node-dev \
  -p 3000:3000 \
  -v $(pwd)/app:/usr/src/app \
  -v /usr/src/app/node_modules \
  -e NODE_ENV=development \
  node:18 \
  npm run dev

# Modifiche a app/ si riflettono immediatamente
```

5.11.3 Multi-Tier App Networking

Listing 5.13: 3-tier architecture

```
# Reti
$ docker network create frontend-net
$ docker network create backend-net

# Database (solo backend)
$ docker run -d \
  --name db \
  --network backend-net \
  -v db-data:/var/lib/postgresql/data \
  postgres

# API (frontend + backend)
$ docker run -d \
  --name api \
  --network frontend-net \
  -e DB_HOST=db \
  node-api

$ docker network connect backend-net api

# Web (solo frontend)
$ docker run -d \
  --name web \
  --network frontend-net \
  -p 80:80 \
  nginx
```

Best Practices

Best Practices

Networking:

- Usa reti custom per service discovery automatico
- Segmenta applicazione con reti multiple
- Usa `-internal` per reti senza accesso internet
- Evita `-network host` se non necessario
- Documenta port mapping nei README

Volumes:

- Named volumes in produzione, bind mounts in sviluppo
- Backup regolari di volumi critici
- Usa `:ro` per mount read-only quando possibile
- Pulisci volumi inutilizzati periodicamente
- Considera driver cloud per high availability

Errori Comuni

Attenzione

1. **Default bridge senza DNS**: Usa reti custom
2. **Porta in conflitto**:

   ```
   Error: Bind for 0.0.0.0:80 failed: port is already allocated
   ```

   Soluzione: Cambia porta host o ferma servizio esistente
3. **Volume cancellato per errore**:

   ```
   $ docker compose down -v  # ATTENZIONE: cancella volumi!
   ```

   Soluzione: Ometti -v, usa backup regolari

4. **Bind mount con path errato**: Verifica path assoluti
5. **Permission denied su bind mount**: Controlla ownership/chmod
6. **Container non comunicano**: Verifica stessa rete

Esercizi

1. Crea un'app WordPress:
   - MySQL su rete backend con volume persistente
   - WordPress su rete frontend+backend
   - Nginx reverse proxy su rete frontend
2. Implementa service discovery:
   - 3 container su rete custom
   - Testa ping via hostname
   - Aggiungi network alias
3. Backup e restore:
   - Crea volume con dati PostgreSQL
   - Esegui backup in tar.gz
   - Restore su nuovo volume
   - Verifica integrità dati
4. Hot reload development:
   - Setup React app con bind mount
   - Modifica codice e verifica ricaricamento
   - Confronta con named volume (no hot reload)

Quiz di Verifica

1. Qual è la differenza tra bridge default e custom?
2. Quando useresti `-network host`?
3. Come fare backup di un volume Docker?
4. **Vero/Falso**: tmpfs mounts persistono dopo riavvio container.
5. Quale tipo di mount è consigliato per produzione? Perché?

Riepilogo

- **Bridge**: Rete privata con DNS (custom) o senza (default)
- **Host**: Network stack condiviso, max performance
- **Overlay**: Multi-host per Swarm/K8s

- **Service Discovery**: DNS automatico su reti custom
- **Volumes**: Persistenza gestita da Docker
- **Bind Mounts**: Mount directory host, per sviluppo
- **tmpfs**: Dati in RAM, temporanei
- **Backup**: Usa container helper per tar.gz

Prossimi Passi

Nel prossimo capitolo esploreremo:

- Docker Hub e registry pubblici
- Registry privati (Harbor, AWS ECR, GCR)
- Push/pull immagini
- Tag e versioning
- CI/CD integration

Riferimenti

- Docker Networking: https://docs.docker.com/network/
- Docker Volumes: https://docs.docker.com/storage/volumes/
- Network Drivers: https://docs.docker.com/network/drivers/
- Volume Plugins: https://docs.docker.com/engine/extend/plugins_volume/

Capitolo 6

Docker Registry e Hub

Introduzione

I registry Docker sono repository centralizzati per memorizzare e distribuire immagini Docker. Questo capitolo copre Docker Hub, registry privati, strategie di versioning, e integrazione con pipeline CI/CD.

Obiettivi di apprendimento

- Usare Docker Hub per pull e push di immagini
- Implementare strategie di tagging e versioning
- Setup registry privati (Docker Registry, Harbor)
- Configurare registry cloud (AWS ECR, GCP GCR, Azure ACR)
- Integrare con CI/CD per build e deploy automatici
- Applicare security scanning e best practices

6.1 Docker Hub

6.1.1 Cos'è Docker Hub

Docker Hub è il registry pubblico ufficiale di Docker:

- 100.000+ immagini ufficiali e community
- Gratuito per repository pubblici
- Piani a pagamento per repository privati
- Automated builds da GitHub/Bitbucket
- Webhook e integrazioni

URL: `https://hub.docker.com`

6.1.2 Account e Login

```
# Crea account su hub.docker.com, poi login
$ docker login
Username: myusername
Password:
Login Succeeded

# Login con token (più sicuro)
$ docker login -u myusername -p $(cat token.txt)

# Logout
$ docker logout
```

6.1.3 Pull Immagini

```
# Formato: [REGISTRY/]REPOSITORY[:TAG]

# Pull da Docker Hub (default registry)
$ docker pull nginx
$ docker pull nginx:1.25-alpine
$ docker pull ubuntu:22.04

# Pull da user/org repository
$ docker pull myusername/myapp:latest
$ docker pull bitnami/postgresql:15

# Pull da registry alternativo
$ docker pull ghcr.io/myorg/myapp:v1.0
$ docker pull quay.io/prometheus/prometheus
```

6.1.4 Push Immagini

Listing 6.1: Pubblicare immagine su Docker Hub

```
# 1. Build immagine con tag corretto
$ docker build -t myusername/myapp:v1.0 .

# 2. (Opzionale) Tag aggiuntivo per latest
$ docker tag myusername/myapp:v1.0 myusername/myapp:latest

# 3. Login
$ docker login

# 4. Push
$ docker push myusername/myapp:v1.0
$ docker push myusername/myapp:latest

# Verifica su hub.docker.com/r/myusername/myapp
```

6.1.5 Repository Pubblici vs Privati

```
# Crea repository privato su hub.docker.com

# Push a repository privato
```

Tipo	Visibilità	Costo
Pubblico	Tutti possono pull	Gratis
Privato	Solo autorizzati	1 gratis, poi a pagamento

Tabella 6.1: Repository Docker Hub

```
$ docker push myusername/private-app:v1.0

# Pull richiede autenticazione
$ docker login
$ docker pull myusername/private-app:v1.0
```

6.2 Tagging e Versioning

6.2.1 Strategie di Tag

Best Practices Tagging

1. **Semantic Versioning**: `v1.2.3` (major.minor.patch)
2. **Tag multipli**:

```
myapp:v1.2.3      # Versione specifica
myapp:v1.2        # Minor version
myapp:v1          # Major version
myapp:latest      # Ultima stabile
```

3. **Tag descrittivi**:

```
myapp:v1.2.3-alpine
myapp:v1.2.3-debian
myapp:nightly
myapp:dev
myapp:prod
```

4. **Git commit SHA**:

```
myapp:sha-a1b2c3d
myapp:v1.2.3-a1b2c3d
```

6.2.2 Esempio Completo Tagging

Listing 6.2: Multi-tag workflow

```
# Versione corrente
VERSION=1.2.3

# Build immagine
$ docker build -t myusername/myapp:v${VERSION} .

# Tag multiple version levels
$ docker tag myusername/myapp:v${VERSION} myusername/myapp:v1.2
$ docker tag myusername/myapp:v${VERSION} myusername/myapp:v1
$ docker tag myusername/myapp:v${VERSION} myusername/myapp:latest
```

```

# Push tutti i tag
$ docker push myusername/myapp:v${VERSION}
$ docker push myusername/myapp:v1.2
$ docker push myusername/myapp:v1
$ docker push myusername/myapp:latest

# Oppure push all tags
$ docker push --all-tags myusername/myapp
```

6.2.3 Tag Immutabili

Attenzione

Evita di sovrascrivere tag in produzione!

```
# SBAGLIATO: Sovrascrivi tag esistente
$ docker tag myapp:latest myapp:v1.0
$ docker push myapp:v1.0   # Sovrascrive v1.0 precedente

# CORRETTO: Usa nuovo tag
$ docker tag myapp:latest myapp:v1.1
$ docker push myapp:v1.1
```

Solo `latest`, `dev`, `nightly` dovrebbero essere sovrascritti.

6.3 Registry Privati

6.3.1 Docker Registry (Open Source)

Registry ufficiale Docker, self-hosted.

Listing 6.3: Setup Docker Registry

```
# Deploy registry con Docker
$ docker run -d \
  -p 5000:5000 \
  --name registry \
  --restart always \
  -v registry-data:/var/lib/registry \
  registry:2

# Push a registry locale
$ docker tag myapp localhost:5000/myapp:v1.0
$ docker push localhost:5000/myapp:v1.0

# Pull da registry locale
$ docker pull localhost:5000/myapp:v1.0
```

6.3.2 Registry con HTTPS e Autenticazione

Listing 6.4: Secure registry setup

```
# Genera certificati SSL (self-signed)
$ mkdir -p certs auth
$ openssl req -newkey rsa:4096 -nodes -sha256 \
  -keyout certs/domain.key -x509 -days 365 \
  -out certs/domain.crt

# Crea htpasswd per autenticazione
$ docker run --rm --entrypoint htpasswd \
  httpd:2 -Bbn myuser mypassword > auth/htpasswd

# Deploy registry con TLS e auth
$ docker run -d \
  -p 5000:5000 \
  --name secure-registry \
  --restart always \
  -v $(pwd)/certs:/certs \
  -v $(pwd)/auth:/auth \
  -v registry-data:/var/lib/registry \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -e REGISTRY_AUTH=htpasswd \
  -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
  -e REGISTRY_AUTH_HTPASSWD_REALM="Registry Realm" \
  registry:2

# Login
$ docker login myregistry.com:5000
Username: myuser
Password:
```

6.3.3 Docker Compose per Registry

Listing 6.5: docker-compose.yml per registry

```
version: '3.8'

services:
  registry:
    image: registry:2
    container_name: docker-registry
    restart: always
    ports:
      - "5000:5000"
    environment:
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry
    volumes:
      - registry-data:/data
      - ./auth:/auth
    networks:
      - registry-net

  # UI per navigare registry
  registry-ui:
    image: joxit/docker-registry-ui:latest
    container_name: registry-ui
    restart: always
    ports:
      - "8080:80"
    environment:
      - REGISTRY_TITLE=My Docker Registry
      - REGISTRY_URL=http://registry:5000
      - DELETE_IMAGES=true
      - SHOW_CONTENT_DIGEST=true
    networks:
      - registry-net
    depends_on:
      - registry

networks:
  registry-net:

volumes:
  registry-data:
```

6.4 Harbor: Enterprise Registry

6.4.1 Cos'è Harbor

Harbor è un registry enterprise open-source by VMware/CNCF:

- Web UI completa
- Role-based access control (RBAC)
- Vulnerability scanning integrato
- Image signing e notary
- Replication tra registry
- Webhook e audit logging
- Helm charts support

6.4.2 Installazione Harbor

Listing 6.6: Deploy Harbor con Docker Compose

```
# Download installer
$ wget https://github.com/goharbor/harbor/releases/download/v2.9.0/
   harbor-offline-installer-v2.9.0.tgz
$ tar xzf harbor-offline-installer-v2.9.0.tgz
$ cd harbor

# Configura
$ cp harbor.yml.tmpl harbor.yml
$ vim harbor.yml
# Modifica:
# - hostname: registry.example.com
# - harbor_admin_password: MySecretPass
# - database password
# - certificate paths (se HTTPS)

# Installa
$ sudo ./install.sh --with-trivy --with-chartmuseum

```

```
# Accedi a https://registry.example.com
# User: admin
# Pass: MySecretPass
```

6.4.3 Usare Harbor

```
# Login
$ docker login registry.example.com
Username: admin
Password:

# Tag immagine per Harbor
$ docker tag myapp registry.example.com/myproject/myapp:v1.0

# Push
$ docker push registry.example.com/myproject/myapp:v1.0

# Pull
$ docker pull registry.example.com/myproject/myapp:v1.0
```

6.5 Cloud Registry

6.5.1 AWS Elastic Container Registry (ECR)

Listing 6.7: AWS ECR workflow

```
# Installa AWS CLI
$ aws configure

# Crea repository
$ aws ecr create-repository --repository-name myapp

# Login a ECR
$ aws ecr get-login-password --region us-east-1 | \
  docker login --username AWS --password-stdin \
  123456789012.dkr.ecr.us-east-1.amazonaws.com

# Tag immagine
$ docker tag myapp:v1.0 \
  123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:v1.0

# Push
$ docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:v1.0

# Pull
$ docker pull 123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:v1.0
```

6.5.2 Google Container Registry (GCR)

Listing 6.8: GCR workflow

```
# Installa gcloud CLI
$ gcloud auth configure-docker

# Tag immagine
```

```
$ docker tag myapp:v1.0 gcr.io/my-project-id/myapp:v1.0

# Push
$ docker push gcr.io/my-project-id/myapp:v1.0

# Pull
$ docker pull gcr.io/my-project-id/myapp:v1.0
```

6.5.3 Azure Container Registry (ACR)

Listing 6.9: Azure ACR workflow

```
# Crea registry
$ az acr create --resource-group myResourceGroup \
  --name myregistry --sku Basic

# Login
$ az acr login --name myregistry

# Tag immagine
$ docker tag myapp:v1.0 myregistry.azurecr.io/myapp:v1.0

# Push
$ docker push myregistry.azurecr.io/myapp:v1.0

# Pull
$ docker pull myregistry.azurecr.io/myapp:v1.0
```

6.5.4 Confronto Cloud Registry

Feature	AWS ECR	GCP GCR	Azure ACR
Pricing	Storage + transfer	Storage + egress	Tiered (Basic/Standard/Premium)
Scanning	ECR scan	GCR scan	Defender for Cloud
Geo-replication	Si (Premium)	Multi-region	Si (Premium)
Integrazione	ECS, EKS, Fargate	GKE, Cloud Run	AKS, Container Instances

Tabella 6.2: Cloud Registry Comparison

6.6 CI/CD Integration

6.6.1 GitHub Actions

Listing 6.10: .github/workflows/docker.yml

```
name: Docker Build and Push

on:
  push:
    branches: [ main ]
    tags: [ 'v*' ]

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v4
        with:
          images: myusername/myapp
          tags: |
            type=ref,event=branch
            type=semver,pattern={{version}}
            type=semver,pattern={{major}}.{{minor}}
            type=sha

      - name: Build and push
        uses: docker/build-push-action@v4
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

6.6.2 GitLab CI/CD

Listing 6.11: .gitlab-ci.yml

```
variables:
  IMAGE_NAME: $CI_REGISTRY_IMAGE
  IMAGE_TAG: $CI_COMMIT_REF_SLUG

stages:
  - build
  - push

build:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
  only:
    - main
    - tags

push-latest:
  stage: push
  image: docker:latest
  services:
    - docker:dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
       $CI_REGISTRY
  script:
    - docker pull $IMAGE_NAME:$IMAGE_TAG
    - docker tag $IMAGE_NAME:$IMAGE_TAG $IMAGE_NAME:latest
    - docker push $IMAGE_NAME:latest
  only:
    - main
```

6.6.3 Jenkins Pipeline

Listing 6.12: Jenkinsfile

```
pipeline {
    agent any

    environment {
        REGISTRY = 'myregistry.com:5000'
        IMAGE_NAME = 'myapp'
        IMAGE_TAG = "${env.BUILD_NUMBER}"
    }

    stages {
        stage('Build') {
            steps {
                script {
                    docker.build("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG
                       }")
                }
            }
        }

        stage('Push') {
            steps {
                script {
                    docker.withRegistry("https://${REGISTRY}", 'registry
                       -credentials') {
                        docker.image("${REGISTRY}/${IMAGE_NAME}:${
                           IMAGE_TAG}").push()
                        docker.image("${REGISTRY}/${IMAGE_NAME}:${
                           IMAGE_TAG}").push('latest')
                    }
                }
            }
        }

        stage('Deploy') {
            steps {
                sh '''
                    docker pull ${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}
                    docker stop myapp || true
                    docker rm myapp || true
                    docker run -d --name myapp -p 80:80 ${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}
                '''
            }
        }
    }
}
```

6.7 Security e Best Practices

6.7.1 Image Scanning

Listing 6.13: Scan vulnerabilities

```
# Docker scan (Snyk)
$ docker scan myapp:latest

# Trivy (open source)
$ trivy image myapp:latest

# Grype
$ grype myapp:latest

# Clair
$ clairctl analyze myapp:latest
```

6.7.2 Content Trust

Listing 6.14: Docker Content Trust (DCT)

```
# Abilita content trust
$ export DOCKER_CONTENT_TRUST=1

# Push firma automaticamente
$ docker push myusername/myapp:v1.0
# Richiede passphrase per chiave di signing

# Pull verifica firma
$ docker pull myusername/myapp:v1.0
# Fallisce se firma non valida
```

6.7.3 Image Signing con Cosign

```
# Installa cosign
$ brew install cosign   # macOS
$ apt install cosign    # Linux

# Genera keypair
$ cosign generate-key-pair

```

```
# Firma immagine
$ cosign sign --key cosign.key myregistry.com/myapp:v1.0

# Verifica firma
$ cosign verify --key cosign.pub myregistry.com/myapp:v1.0
```

6.7.4 Best Practices

Security Best Practices

1. **Scan regolarmente**: CI/CD pipeline con scanning
2. **Base images ufficiali**: Usa immagini verificate
3. **Minimal images**: Alpine, distroless
4. **Multi-stage builds**: No build tools in produzione
5. **No secrets in images**:

   ```
   # SBAGLIATO
   ENV API_KEY=secret123

   # CORRETTO
   # Pass at runtime
   $ docker run -e API_KEY=$(vault read secret) myapp
   ```

6. **Versioni esplicite**: No :latest in prod
7. **Content trust**: Firma immagini critiche
8. **Registry privati**: Per codice proprietario
9. **RBAC**: Limita accesso push/pull
10. **Audit logging**: Traccia chi push cosa quando

6.8 Gestione Registry Avanzata

6.8.1 Garbage Collection

Listing 6.15: Cleanup registry storage

```
# Docker Registry garbage collection
$ docker exec registry bin/registry garbage-collect \
  /etc/docker/registry/config.yml

# Delete old images (API)
$ curl -X DELETE http://registry:5000/v2/myapp/manifests/sha256:abc123
    ...
```

6.8.2 Replication

Listing 6.16: Harbor replication example

```
# Replica tra due Harbor registry
```

```
# Via Harbor UI:
# Administration -> Replications -> New Replication Rule
# - Name: prod-to-backup
# - Source: Local
# - Destination: backup-harbor.com
# - Trigger: Event based (push)
```

6.8.3 Webhook Notifications

Listing 6.17: Registry webhook config

```
# /etc/docker/registry/config.yml
notifications:
  endpoints:
    - name: slack
      url: https://hooks.slack.com/services/YOUR/WEBHOOK/URL
      headers:
        Content-Type: [application/json]
      events:
        - push
        - pull
        - delete
```

6.9 Caso di Studio: Production Registry

Listing 6.18: Production-grade registry stack

```
version: '3.8'

services:
  # Harbor core services
  harbor:
    image: goharbor/harbor:v2.9.0
    restart: always
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - harbor-data:/data
      - ./certs:/certs
    environment:
      - HARBOR_ADMIN_PASSWORD=${ADMIN_PASSWORD}
    networks:
      - harbor-net

  # Trivy scanner
  trivy:
    image: goharbor/trivy-adapter-photon:v2.9.0
    restart: always
    environment:
      - SCANNER_TRIVY_CACHE_DIR=/home/scanner/.cache/trivy
    volumes:
      - trivy-cache:/home/scanner/.cache
    networks:
      - harbor-net

  # PostgreSQL database
  postgres:
    image: goharbor/harbor-db:v2.9.0
    restart: always
    environment:
      - POSTGRES_PASSWORD=${DB_PASSWORD}
    volumes:
      - postgres-data:/var/lib/postgresql/data
    networks:
      - harbor-net

  # Redis cache
  redis:
    image: goharbor/redis-photon:v2.9.0
    restart: always
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - harbor-net

  # Nginx reverse proxy
  nginx:
    image: nginx:alpine
    restart: always
    ports:
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./certs:/etc/nginx/certs:ro
    networks:
      - harbor-net
    depends_on:
      - harbor

networks:
  harbor-net:
    driver: bridge

volumes:
  harbor-data:
  trivy-cache:
  postgres-data:
  redis-data:
```

Errori Comuni

Attenzione

1. **Push senza tag**: Default :latest sovrascrive

2. **Insecure registry**:

```
# Error: http: server gave HTTP response to HTTPS client

# Fix: /etc/docker/daemon.json
{
  "insecure-registries": ["myregistry.com:5000"]
}
$ sudo systemctl restart docker
```

3. **Credentials non salvate**: Usa credential helper
4. **Rate limiting Docker Hub**: 100 pull/6h (free tier)
5. **Storage pieno registry**: Setup garbage collection
6. **Tag latest in produzione**: Usa versioni esplicite

Esercizi

1. Setup Docker Hub account e pubblica un'immagine
2. Deploy registry privato:
   - Setup con HTTPS e autenticazione
   - Push/pull immagini
   - Verifica via UI
3. Implementa versioning strategy:
   - Semantic versioning (v1.2.3)
   - Multi-tag (latest, v1, v1.2, v1.2.3)
   - Script automation
4. CI/CD pipeline:
   - GitHub Actions build automatico
   - Push su Docker Hub
   - Deploy su server staging
5. Security scanning:
   - Scansiona immagine con Trivy
   - Risolvi vulnerabilità HIGH/CRITICAL
   - Integra scanning in CI/CD

Quiz di Verifica

1. Qual è il formato completo di un'immagine Docker?
2. Cosa significa tag "latest"? È sicuro in produzione?
3. Differenza tra Docker Registry e Harbor?
4. Come configurare registry insecure (solo HTTP)?
5. Perché è importante scannerizzare le immagini?

Riepilogo

- **Docker Hub**: Registry pubblico ufficiale
- **Tagging**: Semantic versioning, tag multipli
- **Registry privati**: Docker Registry, Harbor
- **Cloud registry**: AWS ECR, GCP GCR, Azure ACR
- **CI/CD**: Automazione build/push/deploy
- **Security**: Scanning, signing, RBAC
- **Best practices**: Versioni esplicite, no secrets, minimal images

Conclusione del Corso

Complimenti! Hai completato il corso Docker e DevOps. Ora sei in grado di:

- Containerizzare qualsiasi applicazione
- Creare Dockerfile ottimizzati
- Orchestrare stack con Docker Compose
- Configurare networking e volumi
- Distribuire su registry pubblici e privati
- Implementare CI/CD pipeline
- Applicare security best practices

Prossimi passi consigliati:

- Kubernetes per orchestrazione enterprise
- Docker Swarm per clustering
- Monitoring con Prometheus/Grafana
- Service mesh con Istio/Linkerd
- Certificazione Docker Certified Associate (DCA)

Riferimenti

- Docker Hub: `https://hub.docker.com`
- Docker Registry: `https://docs.docker.com/registry/`
- Harbor: `https://goharbor.io`
- AWS ECR: `https://aws.amazon.com/ecr/`
- GCP GCR: `https://cloud.google.com/container-registry`
- Azure ACR: `https://azure.microsoft.com/en-us/services/container-registry/`
- Trivy: `https://github.com/aquasecurity/trivy`
- Cosign: `https://github.com/sigstore/cosign`

Capitolo 7

Deployment e Orchestrazione

7.1 Introduzione

Il deployment di applicazioni containerizzate richiede strategie sofisticate per garantire alta disponibilità, scalabilità e zero-downtime. Questo capitolo esplora pattern di deployment, introduzione all'orchestrazione e fondamenti di Kubernetes.

Mappa del capitolo

Sezioni: Strategie di deployment, Docker Swarm, Kubernetes basics, Service mesh, Load balancing, Rolling updates, Blue-green deployment, Canary releases, Health checks avanzati, Secrets management.

7.2 Obiettivi di Apprendimento

- Comprendere le strategie di deployment per applicazioni containerizzate
- Implementare orchestrazione con Docker Swarm e Kubernetes
- Gestire rolling updates e rollback senza downtime
- Configurare health checks e readiness probes
- Applicare pattern di deployment avanzati (blue-green, canary)

7.3 Strategie di Deployment

7.3.1 Deployment Patterns

Listing 7.1: Recreate Strategy - Downtime Accettabile

```
# Stop tutti i container vecchi
docker-compose down

# Deploy nuova versione
docker-compose up -d

# Pro: Semplice, resource-efficient
# Contro: Downtime durante il deploy
```

Listing 7.2: Rolling Update - Zero Downtime

```
# Update incrementale container per container
docker service update \
  --image myapp:v2 \
  --update-parallelism 1 \
  --update-delay 10s \
  --update-failure-action rollback \
  myapp-service

# Pro: Zero downtime, graduale
# Contro: Più complesso, richiede orchestratore
```

7.3.2 Blue-Green Deployment

Listing 7.3: Blue-Green con Docker Compose

```
# docker-compose-blue-green.yml
version: ’3.8’

services:
  # BLUE environment (current production)
  app-blue:
    image: myapp:v1
    networks:
      - app-network
    environment:
      - ENV=production
      - VERSION=blue
    deploy:
      replicas: 3
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.app.rule=Host(‘app.example.com‘)"

  # GREEN environment (new version staging)
  app-green:
    image: myapp:v2
    networks:
      - app-network
    environment:
      - ENV=staging
      - VERSION=green
    deploy:
      replicas: 3
    labels:
      - "traefik.enable=false"  # Non ancora in produzione

  # Load Balancer (Traefik)
  traefik:
    image: traefik:v2.10
    command:
      - "--api.insecure=true"
      - "--providers.docker=true"
      - "--entrypoints.web.address=:80"
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - app-network

networks:
  app-network:
    driver: overlay
```

Listing 7.4: Switch Traffic da Blue a Green

```
#!/bin/bash
# blue-green-switch.sh

echo "Testing GREEN environment health..."
curl -f http://app-green:8080/health || exit 1

echo "Switching traffic to GREEN..."
docker service update \
  --label-add "traefik.enable=true" \
  app-green

docker service update \
  --label-add "traefik.enable=false" \
  app-blue

echo "Traffic switched to GREEN (v2)"
echo "Monitor for issues. To rollback:"
echo "  ./blue-green-switch.sh --rollback"
```

Blue-Green Vantaggi

- **Zero downtime**: Switch istantaneo tra ambienti
- **Fast rollback**: Ritorno immediato alla versione precedente
- **Testing**: Ambiente GREEN testabile prima dello switch
- **Contro**: Richiede risorse doppie durante il deployment

7.3.3 Canary Deployment

Listing 7.5: Canary Release - Traffic Splitting

```
# kubernetes-canary.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-stable
spec:
  replicas: 9  # 90% del traffico
  selector:
    matchLabels:
      app: myapp
      version: stable
  template:
    metadata:
      labels:
        app: myapp
        version: stable
    spec:
      containers:
      - name: myapp
        image: myapp:v1
        ports:
        - containerPort: 8080

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-canary
spec:
  replicas: 1  # 10% del traffico
  selector:
    matchLabels:
      app: myapp
      version: canary
  template:
    metadata:
      labels:
        app: myapp
        version: canary
    spec:
      containers:
      - name: myapp
        image: myapp:v2  # Nuova versione
        ports:
        - containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp  # Match entrambe le versioni
  ports:
  - port: 80
    targetPort: 8080
  type: LoadBalancer
```

Listing 7.6: Canary Progressivo

```
#!/bin/bash
# canary-rollout.sh

# Fase 1: 10% canary
kubectl scale deployment myapp-canary --replicas=1
kubectl scale deployment myapp-stable --replicas=9
sleep 300  # Monitor 5 minuti

# Controllo metriche errori
ERROR_RATE=$(kubectl exec -it prometheus -- \
  curl -s 'http://localhost:9090/api/v1/query?query=error_rate' | \
  jq '.data.result[0].value[1]')
```

```

if (( $(echo "$ERROR_RATE < 0.01" | bc -l) )); then
  # Fase 2: 50% canary
  kubectl scale deployment myapp-canary --replicas=5
  kubectl scale deployment myapp-stable --replicas=5
  sleep 300

  # Fase 3: 100% canary (rollout completo)
  kubectl scale deployment myapp-canary --replicas=10
  kubectl scale deployment myapp-stable --replicas=0
else
  echo "ERROR_RATE too high, rolling back..."
  kubectl scale deployment myapp-canary --replicas=0
fi
```

7.4 Docker Swarm

7.4.1 Inizializzazione Cluster

Listing 7.7: Setup Docker Swarm Cluster

```
# Su manager node
docker swarm init --advertise-addr 192.168.1.10

# Output fornisce token per worker nodes:
# docker swarm join --token SWMTKN-1-xxx... 192.168.1.10:2377

# Su worker nodes
docker swarm join \
  --token SWMTKN-1-5abc... \
  192.168.1.10:2377

# Verifica cluster
docker node ls
# ID        HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
# abc123    manager1  Ready   Active        Leader
# def456    worker1   Ready   Active
# ghi789    worker2   Ready   Active
```

7.4.2 Deploy Stack con Docker Swarm

Listing 7.8: Stack Multi-Service Production

```
# stack-production.yml
version: '3.8'

services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
        failure_action: rollback
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
      placement:
        constraints:
          - node.role == worker
    networks:
      - frontend
    configs:
      - source: nginx_config
        target: /etc/nginx/nginx.conf
    secrets:
      - ssl_certificate
      - ssl_key

  app:
    image: myapp:latest
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: ’0.5’
          memory: 512M
        reservations:
          cpus: ’0.25’
          memory: 256M
      update_config:
        parallelism: 2
        delay: 10s
        monitor: 30s
        failure_action: rollback
        order: start-first  # Start new before stopping old
    networks:
      - frontend
      - backend
    environment:
      - DATABASE_URL_FILE=/run/secrets/db_connection
    secrets:
      - db_connection

  db:
    image: postgres:15-alpine
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.labels.database == true
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    environment:
      - POSTGRES_PASSWORD_FILE=/run/secrets/db_password
    secrets:
      - db_password

  redis:
    image: redis:7-alpine
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.labels.cache == true
    networks:
      - backend

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
    internal: true  # No external access

volumes:
  db-data:
    driver: local

configs:
  nginx_config:
    external: true

secrets:
  ssl_certificate:
    external: true
  ssl_key:
    external: true
  db_connection:
    external: true
  db_password:
    external: true
```

Listing 7.9: Deploy e Gestione Stack

```
# Create secrets
echo "postgresql://user:pass@db:5432/mydb" | \
  docker secret create db_connection -

echo "supersecretpassword" | \
  docker secret create db_password -

# Deploy stack
docker stack deploy -c stack-production.yml myapp

# Monitor services
docker stack services myapp
docker service ls
docker service ps myapp_app

# Scale service
docker service scale myapp_app=10

# Update service
docker service update \
  --image myapp:v2 \
  --update-parallelism 2 \
  myapp_app

# Rollback
docker service rollback myapp_app

# Remove stack
docker stack rm myapp
```

7.5 Kubernetes Fundamentals

7.5.1 Architettura Kubernetes

Componenti Kubernetes Cluster

Control Plane:

- **kube-apiserver**: API REST per gestione cluster
- **etcd**: Database distribuito per stato cluster
- **kube-scheduler**: Assegnazione Pods ai Nodes
- **kube-controller-manager**: Controller per Deployments, Services, etc.

Worker Nodes:

- **kubelet**: Agente che esegue Pods sul node
- **kube-proxy**: Network proxy per Services
- **Container runtime**: Docker, containerd, CRI-O

7.5.2 Deployment Completo Kubernetes

Listing 7.10: Production Deployment con Kubernetes

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
  namespace: production
  labels:
    app: web-app
    version: v1
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1         # Max pods oltre replicas durante update
      maxUnavailable: 0   # Zero downtime
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
        version: v1
    spec:
      containers:
      - name: app
        image: myregistry.io/web-app:v1.2.3
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          name: http

        # Health checks
        livenessProbe:
          httpGet:
            path: /health/live
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3

        readinessProbe:
          httpGet:
            path: /health/ready
            port: 8080
          initialDelaySeconds: 10
          periodSeconds: 5
          timeoutSeconds: 3
          successThreshold: 1
          failureThreshold: 3

        # Startup probe for slow-starting apps
        startupProbe:
          httpGet:
            path: /health/startup
            port: 8080
          initialDelaySeconds: 0
          periodSeconds: 10
          timeoutSeconds: 3
          failureThreshold: 30  # 30*10s = 5 minuti max startup

        # Resource management
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 500m
            memory: 512Mi

        # Environment variables
        env:
        - name: ENV
          value: "production"
        - name: LOG_LEVEL
          value: "info"
        - name: DB_HOST
```

```
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: database.host
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: password

        # Volume mounts
        volumeMounts:
        - name: config
          mountPath: /etc/app/config
          readOnly: true
        - name: cache
          mountPath: /var/cache/app

      # Security context
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 1000

      # Image pull secrets
      imagePullSecrets:
      - name: registry-credentials

      # Volumes
      volumes:
      - name: config
        configMap:
          name: app-config
      - name: cache
        emptyDir: {}

---
# Service
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
  namespace: production
spec:
  selector:
    app: web-app
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  type: ClusterIP
  sessionAffinity: ClientIP

---
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
```

```yaml
metadata:
  name: web-app-ingress
  namespace: production
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rate-limit: "100"
spec:
  tls:
  - hosts:
    - app.example.com
    secretName: web-app-tls
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: web-app-service
            port:
              number: 80

---
# HorizontalPodAutoscaler
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: web-app-hpa
  namespace: production
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: web-app
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80

---
# ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: production
data:
  database.host: "postgres.database.svc.cluster.local"
  database.port: "5432"
  redis.host: "redis.cache.svc.cluster.local"
  app.config.json: |
    {
      "features": {
        "beta": false,
        "analytics": true
      }
    }
```

7.5.3 Gestione Secrets Kubernetes

Listing 7.11: Secrets Management

```
# Create secret da file
kubectl create secret generic db-credentials \
  --from-literal=username=admin \
  --from-literal=password=supersecret \
  --namespace=production

# Create secret da file
kubectl create secret generic tls-cert \
  --from-file=tls.crt=./server.crt \
  --from-file=tls.key=./server.key \
  --namespace=production

# Create Docker registry secret
kubectl create secret docker-registry registry-credentials \
  --docker-server=myregistry.io \
  --docker-username=user \
  --docker-password=pass \
  --docker-email=user@example.com \
  --namespace=production

# Encrypt secrets at rest (encryption config)
# /etc/kubernetes/encryption-config.yaml
cat <<EOF > encryption-config.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: $(head -c 32 /dev/urandom | base64)
      - identity: {}
EOF
```

7.6 Load Balancing e Service Discovery

7.6.1 Kubernetes Services

Listing 7.12: Service Types

```
# ClusterIP (default) - Internal only
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 8080

---
# NodePort - Exposed on each Node
apiVersion: v1
kind: Service
metadata:
  name: web-nodeport
spec:
  type: NodePort
  selector:
    app: web
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30080  # 30000-32767

---
# LoadBalancer - Cloud provider integration
apiVersion: v1
kind: Service
metadata:
  name: web-lb
spec:
  type: LoadBalancer
  selector:
    app: web
  ports:
  - port: 80
    targetPort: 8080

---
# Headless Service - Direct pod access
apiVersion: v1
kind: Service
metadata:
  name: database-headless
spec:
  clusterIP: None  # Headless
  selector:
    app: database
  ports:
  - port: 5432
```

7.7 Advanced Health Checks

7.7.1 Multi-Level Health Checks

Listing 7.13: Health Check Endpoints in Go

```go
// healthcheck.go
package main

import (
    "database/sql"
    "encoding/json"
    "net/http"
    "time"
)

type HealthChecker struct {
    db    *sql.DB
    redis *RedisClient
}

// Liveness: Is the app running?
func (h *HealthChecker) LivenessHandler(w http.ResponseWriter, r *http.
   Request) {
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("OK"))
}

// Readiness: Can the app serve traffic?
func (h *HealthChecker) ReadinessHandler(w http.ResponseWriter, r *http.
   Request) {
    status := map[string]interface{}{
        "status": "UP",
        "checks": make(map[string]string),
    }

    // Check database
    ctx, cancel := context.WithTimeout(r.Context(), 2*time.Second)
    defer cancel()

    if err := h.db.PingContext(ctx); err != nil {
        status["status"] = "DOWN"
        status["checks"].(map[string]string)["database"] = "DOWN"
        w.WriteHeader(http.StatusServiceUnavailable)
    } else {
        status["checks"].(map[string]string)["database"] = "UP"
    }

    // Check Redis
    if err := h.redis.Ping(ctx); err != nil {
        status["status"] = "DOWN"
        status["checks"].(map[string]string)["redis"] = "DOWN"
        w.WriteHeader(http.StatusServiceUnavailable)
    } else {
        status["checks"].(map[string]string)["redis"] = "UP"
    }

    json.NewEncoder(w).Encode(status)
}
```

```go

// Startup: Is initialization complete?
func (h *HealthChecker) StartupHandler(w http.ResponseWriter, r *http.Request) {
    if !h.isInitialized() {
        w.WriteHeader(http.StatusServiceUnavailable)
        w.Write([]byte("Initializing..."))
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("Ready"))
}
```

7.8 Deployment Automation

7.8.1 GitOps con ArgoCD

Listing 7.14: ArgoCD Application

```yaml
# argocd-application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: web-app
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/myorg/k8s-manifests
    targetRevision: main
    path: apps/web-app/production

  destination:
    server: https://kubernetes.default.svc
    namespace: production

  syncPolicy:
    automated:
      prune: true      # Delete resources not in Git
      selfHeal: true   # Auto-sync on drift
      allowEmpty: false
    syncOptions:
    - CreateNamespace=true
    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m
```

7.9 Best Practice Deployment

Production Deployment Checklist

1. **Health Checks**: Implementare liveness, readiness, startup probes
2. **Resource Limits**: Definire CPU/memory requests e limits
3. **Rolling Updates**: Configurare maxSurge e maxUnavailable
4. **Secrets**: Mai hardcode credentials, usare Secrets/Vault
5. **Monitoring**: Prometheus metrics, Grafana dashboards
6. **Logging**: Centralized logging (ELK, Loki)
7. **Security**: NetworkPolicies, PodSecurityPolicies
8. **Backup**: Velero per backup Kubernetes
9. **Disaster Recovery**: Multi-zone/region deployment
10. **GitOps**: Versioned infrastructure as code

7.10 Errori Comuni

- **Errore**: Deployment senza health checks
  - **Conseguenza**: Traffic inviato a pods non pronti
  - **Soluzione**: Implementare readiness probe
- **Errore**: Resource limits non configurati
  - **Conseguenza**: OOMKilled, performance degradation
  - **Soluzione**: Profiling e configurazione requests/limits
- **Errore**: Secrets in ConfigMaps o environment variables
  - **Conseguenza**: Credential exposure
  - **Soluzione**: Usare Kubernetes Secrets + encryption at rest

7.11 Riepilogo

Abbiamo esplorato strategie di deployment production-ready: blue-green per switch istantanei, canary per rollout graduali, rolling updates per zero downtime. Docker Swarm offre orchestrazione semplice per cluster piccoli, mentre Kubernetes fornisce piattaforma enterprise-grade con autoscaling, service discovery, e GitOps integration.

7.12 Riferimenti

- Kubernetes Documentation: https://kubernetes.io/docs/
- Docker Swarm: https://docs.docker.com/engine/swarm/
- ArgoCD GitOps: https://argo-cd.readthedocs.io/

- Prometheus Monitoring: https://prometheus.io/docs/

Capitolo 8

CI/CD con Docker

8.1 Introduzione

L'integrazione continua e il deployment continuo (CI/CD) con Docker trasformano il processo di sviluppo, testing e rilascio del software. Questo capitolo esplora pipeline complete con GitHub Actions, GitLab CI, e best practices per containerized workflows.

Mappa del capitolo

Sezioni: CI/CD fundamentals, GitHub Actions workflows, GitLab CI pipelines, Docker build optimization, Multi-stage testing, Security scanning, Container registry management, Deployment automation, Rollback strategies.

8.2 Obiettivi di Apprendimento

- Implementare pipeline CI/CD complete per applicazioni Docker
- Ottimizzare Docker builds con layer caching e multi-stage
- Integrare security scanning (Trivy, Snyk) nelle pipeline
- Configurare automated deployments con rollback
- Gestire container registries e image versioning

8.3 CI/CD Pipeline Architecture

Fasi Pipeline Tipica

1. **Build**: Compilazione applicazione e Docker image
2. **Test**: Unit tests, integration tests, e2e tests
3. **Security Scan**: Vulnerability scanning di dependencies e image
4. **Push**: Pubblicazione image su container registry
5. **Deploy**: Deployment automatico su staging/production
6. **Verify**: Health checks e smoke tests post-deployment
7. **Notify**: Notifiche Slack/Teams/Email

8.4 GitHub Actions Complete Workflow

8.4.1 Multi-Stage CI/CD Pipeline

Listing 8.1: GitHub Actions - Complete Production Pipeline

```yaml
# .github/workflows/docker-ci-cd.yml
name: Docker CI/CD Pipeline

on:
  push:
    branches: [main, develop]
    tags: [’v*’]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}
  DOCKER_BUILDKIT: 1

jobs:
  # JOB 1: Build and Test Application
  build-and-test:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write

    steps:
    - name: Checkout code
      uses: actions/checkout@v4
      with:
        fetch-depth: 0  # Full history for better caching

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3
      with:
        driver-opts: |
          image=moby/buildkit:latest
          network=host

    - name: Cache Docker layers
      uses: actions/cache@v3
      with:
        path: /tmp/.buildx-cache
        key: ${{ runner.os }}-buildx-${{ github.sha }}
        restore-keys: |
          ${{ runner.os }}-buildx-

    - name: Build test image
      uses: docker/build-push-action@v5
      with:
        context: .
        target: test  # Multi-stage build target
        push: false
        load: true
        tags: myapp:test
        cache-from: type=local,src=/tmp/.buildx-cache
        cache-to: type=local,dest=/tmp/.buildx-cache-new,mode=max

    - name: Run unit tests
      run: |
        docker run --rm myapp:test npm run test:unit

    - name: Run integration tests
      run: |
        docker-compose -f docker-compose.test.yml up \
          --abort-on-container-exit \
          --exit-code-from app

    - name: Upload test results
      if: always()
      uses: actions/upload-artifact@v3
      with:
        name: test-results
        path: |
          coverage/
          test-results/

    # Rotate cache to prevent unlimited growth
    - name: Move cache
      run: |
        rm -rf /tmp/.buildx-cache
        mv /tmp/.buildx-cache-new /tmp/.buildx-cache

  # JOB 2: Security Scanning
  security-scan:
    runs-on: ubuntu-latest
    needs: build-and-test
    permissions:
      contents: read
      security-events: write

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Build image for scanning
      run: |
        docker build -t myapp:scan .

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        image-ref: myapp:scan
        format: ’sarif’
        output: ’trivy-results.sarif’
        severity: ’CRITICAL,HIGH’
        exit-code: ’1’  # Fail on vulnerabilities

    - name: Upload Trivy results to GitHub Security
      uses: github/codeql-action/upload-sarif@v2
      if: always()
      with:
        sarif_file: ’trivy-results.sarif’

```

```yaml
    - name: Run Snyk security scan
      uses: snyk/actions/docker@master
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
      with:
        image: myapp:scan
        args: --severity-threshold=high

    - name: Scan Dockerfile with Hadolint
      uses: hadolint/hadolint-action@v3.1.0
      with:
        dockerfile: Dockerfile
        failure-threshold: warning

  # JOB 3: Build and Push Production Image
  build-and-push:
    runs-on: ubuntu-latest
    needs: [build-and-test, security-scan]
    if: github.event_name != 'pull_request'
    permissions:
      contents: read
      packages: write

    outputs:
      image-tag: ${{ steps.meta.outputs.tags }}
      image-digest: ${{ steps.build.outputs.digest }}

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up QEMU
      uses: docker/setup-qemu-action@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to GitHub Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
        tags: |
          type=ref,event=branch
          type=semver,pattern={{version}}
          type=semver,pattern={{major}}.{{minor}}
          type=sha,prefix={{branch}}-
          type=raw,value=latest,enable={{is_default_branch}}

    - name: Build and push multi-arch image
      id: build
      uses: docker/build-push-action@v5
      with:
        context: .
        platforms: linux/amd64,linux/arm64
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:buildcache
        cache-to: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:buildcache,mode=max
        build-args: |
          BUILD_DATE=${{ github.event.repository.updated_at }}
          VCS_REF=${{ github.sha }}
          VERSION=${{ steps.meta.outputs.version }}

    - name: Sign image with Cosign
      env:
        COSIGN_EXPERIMENTAL: 1
      run: |
        cosign sign --yes \
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}@${{ steps.build.outputs.digest }}

  # JOB 4: Deploy to Staging
  deploy-staging:
    runs-on: ubuntu-latest
    needs: build-and-push
    environment:
      name: staging
      url: https://staging.example.com
    if: github.ref == ’refs/heads/develop’

    steps:
    - name: Checkout deployment manifests
      uses: actions/checkout@v4
      with:
        repository: myorg/k8s-manifests
        token: ${{ secrets.DEPLOY_TOKEN }}

    - name: Setup kubectl
      uses: azure/setup-kubectl@v3

    - name: Configure kubeconfig
      run: |
        echo "${{ secrets.KUBECONFIG_STAGING }}" | base64 -d > kubeconfig
        export KUBECONFIG=kubeconfig

    - name: Update image tag
      run: |
        cd apps/myapp/staging
        kustomize edit set image \
          myapp=${{ needs.build-and-push.outputs.image-tag }}

    - name: Deploy to staging
      run: |
        kubectl apply -k apps/myapp/staging
        kubectl rollout status deployment/myapp -n staging --timeout=5m

    - name: Run smoke tests
      run: |
        sleep 30
        curl -f https://staging.example.com/health || exit 1

    - name: Notify Slack
      if: always()
      uses: slackapi/slack-github-action@v1
      with:
        payload: |
          {
            "text": "Staging deployment: ${{ job.status }}",
            "blocks": [
              {
                "type": "section",
                "text": {
                  "type": "mrkdwn",
                  "text": "*Staging Deployment*\nStatus: ${{ job.status }}\nImage: ${{ needs.build-and-push.outputs.image-tag }}"
                }
              }
            ]
          }
      env:
        SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK }}

  # JOB 5: Deploy to Production
  deploy-production:
    runs-on: ubuntu-latest
    needs: build-and-push
    environment:
      name: production
      url: https://example.com
    if: startsWith(github.ref, 'refs/tags/v')

    steps:
    - name: Checkout deployment manifests
      uses: actions/checkout@v4
      with:
        repository: myorg/k8s-manifests
        token: ${{ secrets.DEPLOY_TOKEN }}

    - name: Setup kubectl
      uses: azure/setup-kubectl@v3

    - name: Configure kubeconfig
      run: |
        echo "${{ secrets.KUBECONFIG_PROD }}" | base64 -d > kubeconfig
        export KUBECONFIG=kubeconfig

    - name: Create deployment backup
      run: |
        kubectl get deployment myapp -n production -o yaml > backup-deployment.yaml
        kubectl get configmap -n production -o yaml > backup-configmap.yaml

    - name: Update image tag
      run: |
        cd apps/myapp/production
        kustomize edit set image \
          myapp=${{ needs.build-and-push.outputs.image-tag }}

    - name: Deploy to production (Blue-Green)
      run: |
        # Deploy to green environment
        kubectl apply -k apps/myapp/production/green
        kubectl rollout status deployment/myapp-green -n production --timeout=10m

        # Run production smoke tests
        ./scripts/smoke-test.sh https://green.example.com

        # Switch traffic to green
        kubectl patch service myapp -n production \
          -p '{"spec":{"selector":{"version":"green"}}}'

        # Wait and verify
        sleep 60

        # Scale down blue
        kubectl scale deployment/myapp-blue -n production --replicas=0

    - name: Verify deployment
      run: |
        kubectl get pods -n production
        kubectl get events -n production --sort-by='.lastTimestamp'

    - name: Rollback on failure
      if: failure()
      run: |
        kubectl apply -f backup-deployment.yaml
        kubectl patch service myapp -n production \
          -p '{"spec":{"selector":{"version":"blue"}}}'

    - name: Create GitHub Release
      if: success()
      uses: actions/create-release@v1
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      with:
        tag_name: ${{ github.ref }}
        release_name: Release ${{ github.ref }}
        body: |
          Production deployment successful
          Image: ${{ needs.build-and-push.outputs.image-tag }}
          Digest: ${{ needs.build-and-push.outputs.image-digest }}

  # JOB 6: Performance Testing
  performance-test:
    runs-on: ubuntu-latest
    needs: deploy-staging
    if: github.ref == 'refs/heads/develop'

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Run k6 load test
      uses: grafana/k6-action@v0.3.0
      with:
        filename: tests/load-test.js
        cloud: true
        token: ${{ secrets.K6_CLOUD_TOKEN }}

    - name: Upload performance results
      uses: actions/upload-artifact@v3
      with:
        name: performance-results
        path: results/
```

8.5 GitLab CI Complete Pipeline

8.5.1 GitLab CI/CD Configuration

Listing 8.2: .gitlab-ci.yml - Enterprise Pipeline

```
# .gitlab-ci.yml
variables:
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  REGISTRY: $CI_REGISTRY
  IMAGE: $CI_REGISTRY_IMAGE
  DOCKER_BUILDKIT: 1

stages:
  - build
  - test
  - security
  - package
  - deploy-staging
  - deploy-production

# Template per Docker build con cache
.docker-build-template: &docker-build
  image: docker:24
  services:
    - docker:24-dind
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
       $CI_REGISTRY

# BUILD STAGE
build:app:
  <<: *docker-build
  stage: build
  script:
    - |
      docker build \
        --cache-from $IMAGE:latest \
        --build-arg BUILDKIT_INLINE_CACHE=1 \
        --target builder \
        -t $IMAGE:builder-$CI_COMMIT_SHA \
        .
    - docker push $IMAGE:builder-$CI_COMMIT_SHA
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main"
    - if: $CI_COMMIT_BRANCH == "develop"

# TEST STAGE
test:unit:
  <<: *docker-build
  stage: test
  dependencies:
    - build:app
  script:
    - docker pull $IMAGE:builder-$CI_COMMIT_SHA
    - |
      docker run --rm \
        -v $PWD/coverage:/app/coverage \
        $IMAGE:builder-$CI_COMMIT_SHA \
        npm run test:unit -- --coverage
  coverage: ’/Statements\s+:\s+(\d+\.\d+)%/’
  artifacts:
    reports:
      junit: coverage/junit.xml
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
    paths:
      - coverage/
    expire_in: 1 week

test:integration:
  <<: *docker-build
  stage: test
  services:
    - postgres:15-alpine
    - redis:7-alpine
  variables:
    POSTGRES_DB: testdb
    POSTGRES_USER: testuser
    POSTGRES_PASSWORD: testpass
    DATABASE_URL: postgres://testuser:testpass@postgres:5432/testdb
    REDIS_URL: redis://redis:6379
  script:
    - docker pull $IMAGE:builder-$CI_COMMIT_SHA
    - |
      docker run --rm \
        --network host \
        -e DATABASE_URL=$DATABASE_URL \
        -e REDIS_URL=$REDIS_URL \
        $IMAGE:builder-$CI_COMMIT_SHA \
        npm run test:integration
  artifacts:
    reports:
      junit: test-results/integration.xml

test:e2e:
  image: cypress/browsers:latest
  stage: test
  services:
    - name: $IMAGE:builder-$CI_COMMIT_SHA
      alias: app
  script:
    - npm ci
    - npm run cy:run --env baseUrl=http://app:3000
  artifacts:
    when: always
    paths:
      - cypress/videos/
      - cypress/screenshots/
    expire_in: 1 week

# SECURITY STAGE
security:trivy:
  image: aquasec/trivy:latest
  stage: security
  script:
    - trivy image --exit-code 0 --no-progress --format json -o trivy-report.json $IMAGE:builder-$CI_COMMIT_SHA
    - trivy image --exit-code 1 --severity CRITICAL --no-progress $IMAGE:builder-$CI_COMMIT_SHA
  artifacts:
    reports:
      container_scanning: trivy-report.json
  allow_failure: false

security:sast:
  stage: security
  image: returntocorp/semgrep
  script:
    - semgrep --config=auto --json --output=sast-report.json .
  artifacts:
    reports:
      sast: sast-report.json

security:dependency-scan:
  image: node:20-alpine
  stage: security
  script:
    - npm audit --audit-level=high --json > npm-audit.json
  artifacts:
    reports:
      dependency_scanning: npm-audit.json
  allow_failure: true

security:secrets-scan:
  image: trufflesecurity/trufflehog:latest
  stage: security
  script:
    - trufflehog git file://. --json > secrets-report.json
  artifacts:
    paths:
      - secrets-report.json
  allow_failure: false

# PACKAGE STAGE
package:production:
  <<: *docker-build
  stage: package
  script:
    # Build final production image
    - |
      docker build \
        --cache-from $IMAGE:latest \
        --build-arg BUILDKIT_INLINE_CACHE=1 \
        --label "org.opencontainers.image.created=$(date -Iseconds)" \
        --label "org.opencontainers.image.revision=$CI_COMMIT_SHA" \
        --label "org.opencontainers.image.version=$CI_COMMIT_TAG" \
        -t $IMAGE:$CI_COMMIT_SHA \
        -t $IMAGE:$CI_COMMIT_REF_SLUG \
        .

    # Push all tags
    - docker push $IMAGE:$CI_COMMIT_SHA
    - docker push $IMAGE:$CI_COMMIT_REF_SLUG

    # Tag latest if main branch
    - |
      if [ "$CI_COMMIT_BRANCH" == "main" ]; then
        docker tag $IMAGE:$CI_COMMIT_SHA $IMAGE:latest
        docker push $IMAGE:latest
      fi

    # Tag with version if tagged commit
    - |
      if [ -n "$CI_COMMIT_TAG" ]; then
        docker tag $IMAGE:$CI_COMMIT_SHA $IMAGE:$CI_COMMIT_TAG
        docker push $IMAGE:$CI_COMMIT_TAG
      fi
  only:
    - main
    - develop
    - tags

# DEPLOY STAGING
deploy:staging:
  stage: deploy-staging
  image: bitnami/kubectl:latest
  environment:
    name: staging
    url: https://staging.example.com
    on_stop: stop:staging
  script:
    - kubectl config use-context staging-cluster
    - |
      kubectl set image deployment/myapp \
        myapp=$IMAGE:$CI_COMMIT_SHA \
        -n staging
    - kubectl rollout status deployment/myapp -n staging --timeout=5m
    - sleep 30
    - curl -f https://staging.example.com/health || exit 1
  only:
    - develop

stop:staging:
  stage: deploy-staging
  image: bitnami/kubectl:latest
  environment:
    name: staging
    action: stop
  script:
    - kubectl scale deployment/myapp --replicas=0 -n staging
  when: manual
  only:
    - develop

# DEPLOY PRODUCTION
deploy:production:
  stage: deploy-production
  image: bitnami/kubectl:latest
  environment:
    name: production
    url: https://example.com
  before_script:
    - kubectl config use-context production-cluster
  script:
    # Backup current deployment
    - kubectl get deployment myapp -n production -o yaml > backup.yaml

    # Canary deployment (10%)
    - |
      kubectl apply -f - <<EOF
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: myapp-canary
        namespace: production
      spec:
        replicas: 1
        selector:
          matchLabels:
            app: myapp
            track: canary
        template:
          metadata:
            labels:
              app: myapp
              track: canary
          spec:
            containers:
            - name: myapp
              image: $IMAGE:$CI_COMMIT_SHA
      EOF

    - sleep 120  # Monitor canary

    # Check error rate
    - |
      ERROR_RATE=$(curl -s ’http://prometheus:9090/api/v1/query?query=error_rate{track="canary"}’ | jq -r ’.data.result[0].value[1]’)
      if (( $(echo "$ERROR_RATE > 0.05" | bc -l) )); then
        echo "Canary error rate too high: $ERROR_RATE"
        kubectl delete deployment myapp-canary -n production
        exit 1
      fi

    # Full rollout
    - |
      kubectl set image deployment/myapp \
        myapp=$IMAGE:$CI_COMMIT_SHA \
        -n production
    - kubectl rollout status deployment/myapp -n production --timeout=10m

    # Cleanup canary
    - kubectl delete deployment myapp-canary -n production

  after_script:
    - |
      if [ $CI_JOB_STATUS == ’failed’ ]; then
        echo "Deployment failed, rolling back..."
        kubectl apply -f backup.yaml
      fi

  only:
    - tags
  when: manual  # Require manual approval for production

# ROLLBACK
rollback:production:
  stage: deploy-production
  image: bitnami/kubectl:latest
  environment:
    name: production
  script:
    - kubectl config use-context production-cluster
    - kubectl rollout undo deployment/myapp -n production
    - kubectl rollout status deployment/myapp -n production
  when: manual
  only:
    - tags
```

8.6 Docker Build Optimization

8.6.1 Multi-Stage Dockerfile Optimized

Listing 8.3: Optimized Multi-Stage Build

```
# Dockerfile - Production optimized
# syntax=docker/dockerfile:1.4

# Stage 1: Base dependencies
FROM node:20-alpine AS base
WORKDIR /app
RUN apk add --no-cache \
    dumb-init \
    ca-certificates
ENV NODE_ENV=production

# Stage 2: Dependencies
FROM base AS dependencies
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci --only=production && \
    npm cache clean --force

# Stage 3: Build
FROM base AS builder
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci
COPY . .
RUN npm run build && \
    npm prune --production

# Stage 4: Test
FROM builder AS test
ENV NODE_ENV=test
RUN npm ci
COPY --from=builder /app/dist ./dist
CMD ["npm", "run", "test"]

# Stage 5: Production
FROM base AS production

# Security: non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

# Copy only production files
COPY --from=dependencies --chown=nodejs:nodejs /app/node_modules ./node_modules
COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --chown=nodejs:nodejs package.json ./

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
    CMD node healthcheck.js

USER nodejs
EXPOSE 3000

# Use dumb-init for proper signal handling
ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "dist/server.js"]

# Labels
LABEL org.opencontainers.image.source="https://github.com/myorg/myapp"
LABEL org.opencontainers.image.description="Production-optimized Node.js application"
LABEL org.opencontainers.image.licenses="MIT"
```

8.7 Test Automation

8.7.1 Docker Compose for Testing

Listing 8.4: docker-compose.test.yml

```
version: '3.8'

services:
  app:
    build:
      context: .
      target: test
    environment:
      - NODE_ENV=test
      - DATABASE_URL=postgres://test:test@postgres:5432/testdb
      - REDIS_URL=redis://redis:6379
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_started
    command: npm run test:all

  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: testdb
      POSTGRES_USER: test
      POSTGRES_PASSWORD: test
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U test"]
      interval: 10s
      timeout: 5s
      retries: 5
    tmpfs:
      - /var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 3
```

8.8 Container Registry Management

8.8.1 Multi-Registry Push

Listing 8.5: Push to Multiple Registries

```
#!/bin/bash
# multi-registry-push.sh

set -e

```

```bash
IMAGE_NAME="myapp"
VERSION="${1:-latest}"

REGISTRIES=(
  "docker.io/myorg"
  "ghcr.io/myorg"
  "gcr.io/myproject"
  "myregistry.example.com"
)

# Build once
docker build -t ${IMAGE_NAME}:${VERSION} .

# Push to all registries
for registry in "${REGISTRIES[@]}"; do
  echo "Pushing to $registry..."

  docker tag ${IMAGE_NAME}:${VERSION} ${registry}/${IMAGE_NAME}:${VERSION}
  docker tag ${IMAGE_NAME}:${VERSION} ${registry}/${IMAGE_NAME}:latest

  docker push ${registry}/${IMAGE_NAME}:${VERSION}
  docker push ${registry}/${IMAGE_NAME}:latest
done

# Generate SBOM (Software Bill of Materials)
syft ${IMAGE_NAME}:${VERSION} -o spdx-json > sbom.spdx.json

# Sign images with Cosign
for registry in "${REGISTRIES[@]}"; do
  cosign sign --key cosign.key ${registry}/${IMAGE_NAME}:${VERSION}
done

echo "Image pushed to all registries and signed successfully"
```

8.9 Advanced CI/CD Patterns

8.9.1 Matrix Testing Strategy

Listing 8.6: GitHub Actions Matrix Testing

```yaml
# .github/workflows/matrix-test.yml
name: Matrix Testing

on: [push, pull_request]

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        node: [18, 20, 21]
        database: [postgres, mysql, mongodb]
        exclude:
          # Exclude specific combinations
          - os: windows-latest
            database: mongodb

    steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js ${{ matrix.node }}
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node }}

    - name: Start database container
      run: |
        docker run -d \
          --name test-db \
          -e POSTGRES_PASSWORD=test \
          ${{ matrix.database }}:latest

    - name: Run tests
      env:
        DB_TYPE: ${{ matrix.database }}
      run: npm run test:integration
```

8.10 Secrets Management in CI/CD

8.10.1 Vault Integration

Listing 8.7: GitLab CI with HashiCorp Vault

```
# .gitlab-ci.yml with Vault
variables:
  VAULT_ADDR: https://vault.example.com

deploy:production:
  stage: deploy
  id_tokens:
    VAULT_ID_TOKEN:
      aud: https://vault.example.com
  secrets:
    DATABASE_PASSWORD:
      vault: production/database/password@secret
      file: false
    API_KEY:
      vault: production/api/key@secret
      file: false
  script:
    - echo "Deploying with secrets from Vault..."
    - export DB_PASSWORD=$DATABASE_PASSWORD
    - kubectl create secret generic app-secrets \
        --from-literal=db-password=$DATABASE_PASSWORD \
        --from-literal=api-key=$API_KEY \
        -n production --dry-run=client -o yaml | kubectl apply -f -
```

8.11 Best Practices CI/CD

Production CI/CD Checklist

1. **Build Once, Deploy Many**: Stessa image per tutti gli ambienti
2. **Immutable Tags**: Mai riusare tag (no 'latest' in prod)
3. **Security Scanning**: Integrare Trivy/Snyk in pipeline
4. **Layer Caching**: Usare BuildKit cache per speed
5. **Multi-Stage**: Separare build, test, production stages
6. **Secrets**: Mai hardcode, usare secrets management
7. **Rollback**: Automated rollback on health check failure
8. **Notifications**: Slack/Teams alerts per deployments
9. **Artifact Signing**: Cosign per image signing
10. **SBOM**: Generare Software Bill of Materials

8.12 Errori Comuni

- **Errore**: Usare tag 'latest' in production
  - **Conseguenza**: Deployments non riproducibili
  - **Soluzione**: Semantic versioning o SHA commits
- **Errore**: Build senza layer caching
  - **Conseguenza**: Pipeline lente (10+ minuti)
  - **Soluzione**: BuildKit con registry cache
- **Errore**: Secrets in environment variables
  - **Conseguenza**: Exposure in logs/history
  - **Soluzione**: File-based secrets o Vault

8.13 Riepilogo

CI/CD con Docker richiede pipeline robuste con build optimization, security scanning, automated testing, e deployment strategies. GitHub Actions e GitLab CI offrono ecosistemi completi per containerized workflows, mentre tools come Trivy, Cosign e Vault garantiscono security best practices.

8.14 Riferimenti

- GitHub Actions: https://docs.github.com/actions
- GitLab CI: https://docs.gitlab.com/ee/ci/
- Trivy Security Scanner: https://trivy.dev/

- Cosign Image Signing: https://github.com/sigstore/cosign

Capitolo 9

Monitoring e Logging

9.1 Introduzione

Il monitoring e logging di container Docker è essenziale per production environments. Questo capitolo esplora strategie di observability, centralized logging, metrics collection, distributed tracing, e alerting systems per garantire reliability e troubleshooting efficace.

Mappa del capitolo

Sezioni: Docker logs management, Centralized logging (ELK, Loki), Prometheus metrics, Grafana dashboards, Distributed tracing, Health checks avanzati, Alerting con Alertmanager, Performance monitoring, Log aggregation patterns.

9.2 Obiettivi di Apprendimento

- Implementare centralized logging con ELK Stack e Grafana Loki
- Configurare Prometheus per metrics collection da containers
- Creare Grafana dashboards per visualizzazione real-time
- Implementare distributed tracing con Jaeger
- Configurare alerting rules e notification channels
- Applicare structured logging best practices

9.3 Docker Logs Fundamentals

9.3.1 Docker Logging Drivers

Listing 9.1: Docker Compose - Logging Configuration

```
# docker-compose.yml
version: '3.8'

services:
  app:
    image: myapp:latest
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
        labels: "production,app"
        env: "ENV,VERSION"

  nginx:
    image: nginx:alpine
    logging:
      driver: "syslog"
      options:
        syslog-address: "tcp://localhost:514"
        tag: "nginx-{{.Name}}"

  database:
    image: postgres:15
    logging:
      driver: "fluentd"
      options:
        fluentd-address: "localhost:24224"
        tag: "docker.{{.Name}}"
        fluentd-async: "true"
```

Listing 9.2: Docker Logs Commands

```
# Visualizza logs in real-time
docker logs -f container_name

# Logs con timestamp
docker logs -t container_name

# Ultimi N logs
docker logs --tail 100 container_name

# Logs in range temporale
docker logs --since 2024-01-01T10:00:00 \
            --until 2024-01-01T11:00:00 \
            container_name

# Follow logs di tutti i container in compose
docker-compose logs -f

# Logs di specifico service
docker-compose logs -f app

# Logs con grep
docker logs container_name 2>&1 | grep ERROR
```

9.4 Centralized Logging con ELK Stack

9.4.1 ELK Stack Setup Completo

Listing 9.3: ELK Stack - Docker Compose

```
# docker-compose-elk.yml
version: '3.8'

services:
  # Elasticsearch
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
      - xpack.security.enabled=false
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
    networks:
      - elk
    healthcheck:
      test: ["CMD-SHELL", "curl -f http://localhost:9200/_cluster/health || exit 1"]
      interval: 30s
      timeout: 10s
      retries: 5

  # Logstash
  logstash:
    image: docker.elastic.co/logstash/logstash:8.11.0
    volumes:
      - ./logstash/pipeline:/usr/share/logstash/pipeline
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
    ports:
      - "5000:5000/tcp"
      - "5000:5000/udp"
      - "9600:9600"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elk
    depends_on:
      elasticsearch:
        condition: service_healthy

  # Kibana
  kibana:
    image: docker.elastic.co/kibana/kibana:8.11.0
    ports:
      - "5601:5601"
    environment:
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200
    networks:
      - elk
    depends_on:
      elasticsearch:
        condition: service_healthy

  # Filebeat per raccogliere logs da containers
  filebeat:
    image: docker.elastic.co/beats/filebeat:8.11.0
    user: root
    volumes:
      - ./filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml:ro
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
    command: filebeat -e -strict.perms=false
    networks:
      - elk
    depends_on:
      elasticsearch:
        condition: service_healthy

  # Application con structured logging
  app:
    image: myapp:latest
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
        labels: "app,production"
    labels:
      - "logging=enabled"
    networks:
      - elk

networks:
  elk:
    driver: bridge

volumes:
  elasticsearch-data:
    driver: local
```

9.4.2 Logstash Pipeline Configuration

Listing 9.4: logstash/pipeline/logstash.conf

```
# Logstash pipeline for Docker logs
input {
  beats {
    port => 5044
  }

  tcp {
    port => 5000
    codec => json
  }

  # HTTP input per custom logs
  http {
    port => 8080
    codec => json
  }
}

filter {
  # Parse Docker JSON logs
  if [docker][container][name] {
    mutate {
      add_field => {
        "container_name" => "%{[docker][container][name]}"
        "container_id" => "%{[docker][container][id]}"
      }
    }
  }

  # Parse application logs (JSON format)
  if [message] =~ /^\{.*\}$/ {
    json {
      source => "message"
      target => "app"
    }
  }

  # Parse nginx access logs
  if [container_name] =~ /nginx/ {
    grok {
      match => {
        "message" => '%{IPORHOST:client_ip} - %{USER:user} \[%{HTTPDATE:timestamp}\] "%{WORD:method} %{URIPATHPARAM:request} HTTP/%{NUMBER:http_version}" %{INT:status_code} %{INT:bytes} "%{DATA:referrer}" "%{DATA:user_agent}"'
      }
    }
    date {
      match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
    }
  }

  # Extract error severity
  if [message] =~ /ERROR|FATAL/ {
    mutate {
      add_field => { "severity" => "error" }
    }
  } else if [message] =~ /WARN/ {
    mutate {
      add_field => { "severity" => "warning" }
    }
  } else {
    mutate {
      add_field => { "severity" => "info" }
    }
  }

  # Add geo-location per IP
  if [client_ip] {
    geoip {
      source => "client_ip"
      target => "geoip"
    }
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "docker-logs-%{+YYYY.MM.dd}"
  }

  # Debug output
  if [severity] == "error" {
    stdout {
      codec => rubydebug
    }
  }
}
```

9.4.3 Filebeat Configuration

Listing 9.5: filebeat/filebeat.yml

```
filebeat.inputs:
- type: container
  paths:
    - '/var/lib/docker/containers/*/*.log'
  processors:
    - add_docker_metadata:
        host: "unix:///var/run/docker.sock"
    - decode_json_fields:
        fields: ["message"]
        target: "json"
        overwrite_keys: true

filebeat.autodiscover:
  providers:
    - type: docker
      hints.enabled: true
      templates:
        - condition:
            contains:
              docker.container.labels.logging: "enabled"
          config:
            - type: container
              paths:
                - /var/lib/docker/containers/${data.docker.container.id
                    }/*.log

output.logstash:
  hosts: ["logstash:5044"]
  loadbalance: true

logging.level: info
logging.to_files: true
logging.files:
  path: /var/log/filebeat
  name: filebeat
  keepfiles: 7
  permissions: 0644
```

9.5 Grafana Loki - Lightweight Logging

9.5.1 Loki Stack Setup

Listing 9.6: Grafana Loki Stack

```
# docker-compose-loki.yml
version: '3.8'

services:
  loki:
    image: grafana/loki:2.9.0
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/local-config.yaml
    volumes:
      - ./loki/loki-config.yaml:/etc/loki/local-config.yaml
      - loki-data:/loki
    networks:
      - monitoring

  promtail:
    image: grafana/promtail:2.9.0
    volumes:
      - /var/log:/var/log:ro
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - ./promtail/promtail-config.yaml:/etc/promtail/config.yml
    command: -config.file=/etc/promtail/config.yml
    networks:
      - monitoring
    depends_on:
      - loki

  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
      - GF_USERS_ALLOW_SIGN_UP=false
    volumes:
      - grafana-data:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning
    networks:
      - monitoring
    depends_on:
      - loki

networks:
  monitoring:
    driver: bridge

volumes:
  loki-data:
  grafana-data:
```

9.5.2 Promtail Configuration

Listing 9.7: promtail/promtail-config.yaml

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:
  # Docker containers
  - job_name: docker
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
        refresh_interval: 5s
    relabel_configs:
      - source_labels: [’__meta_docker_container_name’]
        regex: ’/(.*)’
        target_label: ’container’
      - source_labels: [’__meta_docker_container_log_stream’]
        target_label: ’stream’
      - source_labels: [’__meta_docker_container_label_com_docker_compose_service’]
        target_label: ’service’
    pipeline_stages:
      - docker: {}
      - json:
          expressions:
            level: level
            message: message
            timestamp: timestamp
      - labels:
          level:
          stream:
      - timestamp:
          source: timestamp
          format: RFC3339Nano

  # System logs
  - job_name: system
    static_configs:
      - targets:
          - localhost
        labels:
          job: varlogs
          __path__: /var/log/*.log
```

9.6 Prometheus Metrics Collection

9.6.1 Prometheus Stack

Listing 9.8: Prometheus + Exporters

```
# docker-compose-prometheus.yml
version: ’3.8’

services:
  prometheus:
    image: prom/prometheus:v2.48.0
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/usr/share/prometheus/console_libraries'
      - '--web.console.templates=/usr/share/prometheus/consoles'
      - '--web.enable-lifecycle'
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - ./prometheus/alerts.yml:/etc/prometheus/alerts.yml
      - prometheus-data:/prometheus
    networks:
      - monitoring

  # Node Exporter per metrics di sistema
  node-exporter:
    image: prom/node-exporter:v1.7.0
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    ports:
      - "9100:9100"
    networks:
      - monitoring

  # cAdvisor per metrics containers
  cadvisor:
    image: gcr.io/cadvisor/cadvisor:v0.47.0
    privileged: true
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:ro
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
      - /dev/disk/:/dev/disk:ro
    ports:
      - "8080:8080"
    networks:
      - monitoring

  # Alertmanager
  alertmanager:
    image: prom/alertmanager:v0.26.0
    command:
      - '--config.file=/etc/alertmanager/config.yml'
      - '--storage.path=/alertmanager'
    ports:
      - "9093:9093"
    volumes:
      - ./alertmanager/config.yml:/etc/alertmanager/config.yml
      - alertmanager-data:/alertmanager
    networks:
      - monitoring

  # Application con Prometheus metrics
  app:
    image: myapp:latest
    ports:
      - "8000:8000"
    environment:
      - PROMETHEUS_METRICS_PORT=9091
    labels:
      - "prometheus.io/scrape=true"
      - "prometheus.io/port=9091"
      - "prometheus.io/path=/metrics"
    networks:
      - monitoring

networks:
  monitoring:
    driver: bridge

volumes:
  prometheus-data:
  alertmanager-data:
```

9.6.2 Prometheus Configuration

Listing 9.9: prometheus/prometheus.yml

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    cluster: 'docker-cluster'
    environment: 'production'

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets: ['alertmanager:9093']

# Load rules
rule_files:
  - "alerts.yml"

scrape_configs:
  # Prometheus self-monitoring
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  # Node Exporter
  - job_name: 'node-exporter'
    static_configs:
      - targets: [’node-exporter:9100’]

  # cAdvisor
  - job_name: ’cadvisor’
    static_configs:
      - targets: [’cadvisor:8080’]

  # Docker daemon metrics
  - job_name: ’docker’
    static_configs:
      - targets: [’host.docker.internal:9323’]

  # Docker Swarm service discovery
  - job_name: ’docker-swarm’
    dockerswarm_sd_configs:
      - host: unix:///var/run/docker.sock
        role: tasks
    relabel_configs:
      - source_labels: [
         __meta_dockerswarm_service_label_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [
         __meta_dockerswarm_service_label_prometheus_io_port]
        target_label: __address__
        regex: ([^:]+)(?::\d+)?
        replacement: $1:${1}

  # Kubernetes pods (if running in K8s)
  - job_name: ’kubernetes-pods’
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [
         __meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [
         __meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__,
         __meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
```

9.6.3 Application Metrics in Go

Listing 9.10: Prometheus Metrics Instrumentation

```
// metrics.go
package main

import (
    "net/http"
    "time"

    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promauto"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
    // Counter: incrementa sempre
    httpRequestsTotal = promauto.NewCounterVec(
        prometheus.CounterOpts{
            Name: "http_requests_total",
            Help: "Total number of HTTP requests",
        },
        []string{"method", "endpoint", "status"},
    )

    // Histogram: distribuzione valori (latency, sizes)
    httpRequestDuration = promauto.NewHistogramVec(
        prometheus.HistogramOpts{
            Name:    "http_request_duration_seconds",
            Help:    "HTTP request latency distribution",
            Buckets: prometheus.DefBuckets,
        },
        []string{"method", "endpoint"},
    )

    // Gauge: valore che può salire/scendere
    activeConnections = promauto.NewGauge(
        prometheus.GaugeOpts{
            Name: "active_connections",
            Help: "Number of active connections",
        },
    )

    // Summary: come histogram ma con quantili
    requestSize = promauto.NewSummaryVec(
        prometheus.SummaryOpts{
            Name:       "http_request_size_bytes",
            Help:       "HTTP request size in bytes",
            Objectives: map[float64]float64{0.5: 0.05, 0.9: 0.01, 0.99: 0.001},
        },
        []string{"method"},
    )
)

// Middleware per tracking automatico
func prometheusMiddleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        start := time.Now()

        // Track active connections
        activeConnections.Inc()
        defer activeConnections.Dec()

```

```go
        // Track request size
        requestSize.WithLabelValues(r.Method).Observe(float64(r.ContentLength))

        // Wrap ResponseWriter per catturare status code
        wrapped := &responseWriter{ResponseWriter: w, statusCode: http.StatusOK}

        next.ServeHTTP(wrapped, r)

        duration := time.Since(start).Seconds()

        // Record metrics
        httpRequestsTotal.WithLabelValues(
            r.Method,
            r.URL.Path,
            http.StatusText(wrapped.statusCode),
        ).Inc()

        httpRequestDuration.WithLabelValues(
            r.Method,
            r.URL.Path,
        ).Observe(duration)
    })
}

func main() {
    // Application handlers
    mux := http.NewServeMux()
    mux.HandleFunc("/api/users", handleUsers)
    mux.HandleFunc("/health", handleHealth)

    // Prometheus metrics endpoint
    mux.Handle("/metrics", promhttp.Handler())

    // Apply middleware
    handler := prometheusMiddleware(mux)

    http.ListenAndServe(":8000", handler)
}
```

9.7 Alert Rules

9.7.1 Prometheus Alert Rules

Listing 9.11: prometheus/alerts.yml

```yaml
groups:
  - name: container_alerts
    interval: 30s
    rules:
      # High CPU usage
      - alert: HighCPUUsage
        expr: |
          100 - (avg by(instance) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High CPU usage on {{ $labels.instance }}"
          description: "CPU usage is above 80% (current: {{ $value }}%)"

      # High memory usage
      - alert: HighMemoryUsage
        expr: |
          (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100 > 90
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "High memory usage on {{ $labels.instance }}"
          description: "Memory usage is above 90% (current: {{ $value }}%)"

      # Container down
      - alert: ContainerDown
        expr: |
          up{job="docker"} == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Container {{ $labels.instance }} is down"
          description: "Container has been down for more than 1 minute"

      # High error rate
      - alert: HighErrorRate
        expr: |
          rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m]) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High HTTP error rate on {{ $labels.instance }}"
          description: "Error rate is above 5% (current: {{ $value }}%)"

      # Slow requests
      - alert: SlowRequests
        expr: |
          histogram_quantile(0.99, rate(http_request_duration_seconds_bucket[5m])) > 1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Slow requests detected on {{ $labels.instance }}"
          description: "99th percentile latency is above 1s (current: {{ $value }}s)"

      # Disk space
      - alert: DiskSpaceLow
        expr: |
          (node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"}) * 100 < 10
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Disk space low on {{ $labels.instance }}"
          description: "Disk space is below 10% (current: {{ $value }}%)"

  - name: docker_alerts
    interval: 30s
    rules:
      # Too many containers
      - alert: TooManyContainers
        expr: |
          count(container_last_seen) > 50
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "Too many containers running"
          description: "More than 50 containers are running (current: {{ $value }})"

      # Container restart loop
      - alert: ContainerRestartLoop
        expr: |
          rate(container_last_seen{name!~"POD"}[5m]) > 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Container {{ $labels.name }} is restarting"
          description: "Container has restarted multiple times in the last 5 minutes"
```

9.7.2 Alertmanager Configuration

Listing 9.12: alertmanager/config.yml

```
global:
  resolve_timeout: 5m
  slack_api_url: ’https://hooks.slack.com/services/YOUR/WEBHOOK/URL’

# Templates
templates:
  - ’/etc/alertmanager/templates/*.tmpl’

# Routing tree
route:
  group_by: [’alertname’, ’cluster’, ’service’]
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 12h
  receiver: ’default’

  routes:
    # Critical alerts -> PagerDuty + Slack
    - match:
        severity: critical
      receiver: 'pagerduty-critical'
      continue: true

    - match:
        severity: critical
      receiver: 'slack-critical'

    # Warning alerts -> Slack only
    - match:
        severity: warning
      receiver: 'slack-warnings'

    # Database alerts
    - match_re:
        service: ^(postgres|mysql|redis)$
      receiver: 'database-team'

receivers:
  - name: 'default'
    email_configs:
      - to: 'alerts@example.com'
        from: 'alertmanager@example.com'
        smarthost: 'smtp.example.com:587'
        auth_username: 'alertmanager@example.com'
        auth_password: 'password'

  - name: 'slack-critical'
    slack_configs:
      - channel: '#alerts-critical'
        title: 'CRITICAL: {{ .CommonAnnotations.summary }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{ end }}'
        color: 'danger'
        send_resolved: true

  - name: 'slack-warnings'
    slack_configs:
      - channel: '#alerts-warnings'
        title: 'Warning: {{ .CommonAnnotations.summary }}'
        text: '{{ range .Alerts }}{{ .Annotations.description }}{{ end }}'
        color: 'warning'

  - name: 'pagerduty-critical'
    pagerduty_configs:
      - service_key: 'YOUR_PAGERDUTY_KEY'
        description: '{{ .CommonAnnotations.summary }}'

  - name: 'database-team'
    webhook_configs:
      - url: 'http://internal-alerts-api/webhook'
        send_resolved: true

inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match:
      severity: 'warning'
    equal: ['alertname', 'instance']
```

9.8 Distributed Tracing

9.8.1 Jaeger Tracing Setup

Listing 9.13: Jaeger All-in-One

```
# docker-compose-tracing.yml
version: '3.8'

services:
  jaeger:
    image: jaegertracing/all-in-one:1.51
    environment:
      - COLLECTOR_ZIPKIN_HOST_PORT=:9411
      - COLLECTOR_OTLP_ENABLED=true
    ports:
      - "5775:5775/udp"   # accept zipkin.thrift compact
      - "6831:6831/udp"   # accept jaeger.thrift compact
      - "6832:6832/udp"   # accept jaeger.thrift binary
      - "5778:5778"       # serve configs
      - "16686:16686"     # serve frontend
      - "14250:14250"     # accept gRPC
      - "14268:14268"     # accept jaeger.thrift
      - "14269:14269"     # admin port
      - "9411:9411"       # Zipkin compatible
      - "4317:4317"       # OTLP gRPC
      - "4318:4318"       # OTLP HTTP
    networks:
      - tracing

  app:
    image: myapp:latest
    environment:
      - JAEGER_AGENT_HOST=jaeger
      - JAEGER_AGENT_PORT=6831
      - JAEGER_SAMPLER_TYPE=const
      - JAEGER_SAMPLER_PARAM=1
    networks:
      - tracing

networks:
  tracing:
    driver: bridge
```

9.9 Structured Logging Best Practices

9.9.1 Structured Logging Example

Listing 9.14: Structured Logging with Zap

```go
// logger.go
package main

import (
    "go.uber.org/zap"
    "go.uber.org/zap/zapcore"
)

func NewLogger() (*zap.Logger, error) {
    config := zap.NewProductionConfig()

    config.EncoderConfig.TimeKey = "timestamp"
    config.EncoderConfig.EncodeTime = zapcore.ISO8601TimeEncoder

    config.OutputPaths = []string{"stdout"}
    config.ErrorOutputPaths = []string{"stderr"}

    return config.Build()
}

func main() {
    logger, _ := NewLogger()
    defer logger.Sync()

    // Structured fields
    logger.Info("User login",
        zap.String("user_id", "12345"),
        zap.String("ip", "192.168.1.100"),
        zap.Duration("latency", 150*time.Millisecond),
    )

    // Error with stack trace
    logger.Error("Database connection failed",
        zap.Error(err),
        zap.String("database", "postgres"),
        zap.Int("retry_count", 3),
    )
}
```

9.10 Best Practices

Monitoring/Logging Checklist

1. **Structured Logging**: JSON format per parsing automatico
2. **Log Levels**: DEBUG, INFO, WARN, ERROR, FATAL
3. **Correlation IDs**: Trace requests attraverso microservices
4. **Retention Policy**: 30-90 giorni per compliance
5. **Sampling**: Non loggare ogni richiesta in high-traffic
6. **Alerting**: Alert su anomalie, non su soglie fisse
7. **Dashboards**: Grafana boards per business metrics

8. **Security**: No credentials/PII nei logs

9.11 Riepilogo

Monitoring e logging efficaci richiedono centralized logging (ELK/Loki), metrics collection (Prometheus), visualization (Grafana), e distributed tracing (Jaeger). Structured logging, alerting rules, e retention policies garantiscono observability completa per production environments.

9.12 Riferimenti

- Prometheus: https://prometheus.io/docs/
- Grafana Loki: https://grafana.com/docs/loki/
- ELK Stack: https://www.elastic.co/elastic-stack
- Jaeger: https://www.jaegertracing.io/docs/

Capitolo 10

Best Practices e Security

10.1 Introduzione

Security, optimization e best practices sono fondamentali per production-ready Docker deployments. Questo capitolo copre Dockerfile optimization, layer caching, .dockerignore, security hardening, vulnerability scanning, e compliance requirements.

Mappa del capitolo

Sezioni: Dockerfile best practices, Layer caching optimization, .dockerignore patterns, Security hardening, User namespaces, Secrets management, Image scanning, Network security, Resource limits, Production checklist.

10.2 Obiettivi di Apprendimento

- Ottimizzare Dockerfiles per build speed e image size
- Implementare security best practices (non-root users, read-only filesystem)
- Configurare .dockerignore per build efficiency
- Utilizzare layer caching e BuildKit features
- Scansionare images per vulnerabilità
- Applicare least privilege principle e network isolation

10.3 Dockerfile Optimization

10.3.1 Esempio: Before vs After Optimization

Listing 10.1: Dockerfile NON Ottimizzato

```
# BAD BAD: Inefficient, large image, security issues
FROM node:20

WORKDIR /app

# BAD Copia tutto (inclusi node_modules, .git, etc)
COPY . .

# BAD Esegue come root
# BAD No cache layer optimization
RUN npm install

# BAD Exposes source code
# BAD Development dependencies included

EXPOSE 3000
CMD ["node", "server.js"]
```

Listing 10.2: Dockerfile OTTIMIZZATO

```
#     GOOD: Multi-stage, optimized, secure
# syntax=docker/dockerfile:1.4

# Stage 1: Dependencies
FROM node:20-alpine AS deps
WORKDIR /app
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci --only=production

# Stage 2: Builder
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci
COPY . .
RUN npm run build

# Stage 3: Production
FROM node:20-alpine AS production

# Install security updates
RUN apk upgrade --no-cache

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

WORKDIR /app

# Copy only production artifacts
COPY --from=deps --chown=nodejs:nodejs /app/node_modules ./node_modules
COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --chown=nodejs:nodejs package.json ./

# Security: run as non-root
USER nodejs

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s \
    CMD node healthcheck.js || exit 1

EXPOSE 3000

# Use exec form for proper signal handling
CMD ["node", "dist/server.js"]

```

```
# Metadata labels
LABEL org.opencontainers.image.source="https://github.com/org/repo"
LABEL org.opencontainers.image.version="1.0.0"
LABEL org.opencontainers.image.licenses="MIT"
```

10.3.2 Layer Caching Optimization

Listing 10.3: Optimal Layer Order

```
# Ordine corretto per massimizzare cache hits
FROM python:3.11-slim

# 1. System packages (cambiano raramente)
RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# 2. Requirements (cambiano occasionalmente)
COPY requirements.txt .
RUN --mount=type=cache,target=/root/.cache/pip \
    pip install --no-cache-dir -r requirements.txt

# 3. Application code (cambia frequentemente)
COPY . .

# Questo ordine garantisce:
# - System packages: cache hit quasi sempre
# - Requirements: cache hit se requirements.txt non cambia
# - Code: rebuild solo questo layer se cambia codice
```

10.3.3 BuildKit Advanced Features

Listing 10.4: BuildKit Cache Mounts e Secrets

```
# syntax=docker/dockerfile:1.4

FROM golang:1.21-alpine AS builder

WORKDIR /app

# Cache mount per Go modules
COPY go.mod go.sum ./
RUN --mount=type=cache,target=/go/pkg/mod \
    go mod download

# Secret mount (non saved in image)
RUN --mount=type=secret,id=netrc,target=/root/.netrc \
    go build -o app .

# SSH mount per private repos
RUN --mount=type=ssh \
    git clone git@github.com:private/repo.git

# Bind mount (source files non copiati nell'image)
RUN --mount=type=bind,source=.,target=/src \
    cd /src && go build -o /app/binary

FROM alpine:latest
COPY --from=builder /app/binary /usr/local/bin/
CMD ["binary"]
```

Listing 10.5: Build con BuildKit Features

```
# Enable BuildKit
export DOCKER_BUILDKIT=1

# Build con secret
docker build \
  --secret id=netrc,src=$HOME/.netrc \
  --ssh default \
  --tag myapp:latest .

# Build con cache from registry
docker build \
  --cache-from myregistry.io/myapp:latest \
  --tag myapp:latest .

# Export cache to registry
docker build \
  --cache-to type=registry,ref=myregistry.io/myapp:buildcache \
  --tag myapp:latest .
```

10.4 .dockerignore Best Practices

10.4.1 Comprehensive .dockerignore

Listing 10.6: .dockerignore - Complete Template

```
# Version control
.git
.gitignore
.gitattributes
.gitmodules

# CI/CD
.github
.gitlab-ci.yml
.travis.yml
Jenkinsfile

# Documentation
README.md
CHANGELOG.md
LICENSE
docs/
*.md

# Dependencies (rebuild from package files)
node_modules/
vendor/
venv/
__pycache__/
```

```
*.pyc
*.pyo

# Build artifacts
dist/
build/
target/
*.o
*.a
*.so

# IDE
.vscode/
.idea/
*.swp
*.swo
*~
.DS_Store

# Logs
*.log
logs/
npm-debug.log*
yarn-debug.log*

# Test files
tests/
test/
spec/
*.test.js
*.spec.js
coverage/
.nyc_output/

# Environment
.env
.env.local
.env.*.local
*.pem
*.key

# Temp files
tmp/
temp/
*.tmp

# Docker
Dockerfile*
docker-compose*.yml
.dockerignore

# Build cache
.cache/
.npm/
.yarn/

# OS files
Thumbs.db
```

```
desktop.ini

# Large data files (if not needed)
*.csv
*.zip
*.tar.gz
datasets/

# Negative patterns (exceptions)
!dist/index.html  # Include specific file
```

.dockerignore Impact

Benefici:

- **Build Speed**: Riduce context size da GB a MB
- **Security**: Esclude .env, .git con secrets
- **Image Size**: Non include test files, docs
- **Cache**: Migliora layer caching efficiency

Esempio:

- Senza .dockerignore: Context 2.5 GB, build 5 minuti
- Con .dockerignore: Context 50 MB, build 30 secondi

10.5 Security Hardening

10.5.1 Non-Root User

Listing 10.7: Multiple User Strategies

```
# Strategy 1: Alpine adduser
FROM alpine:latest
RUN addgroup -g 1001 -S appgroup && \
    adduser -S appuser -u 1001 -G appgroup
USER appuser

# Strategy 2: Debian/Ubuntu useradd
FROM ubuntu:22.04
RUN groupadd -r appgroup -g 1001 && \
    useradd -r -u 1001 -g appgroup appuser
USER appuser

# Strategy 3: Existing user (nginx example)
FROM nginx:alpine
USER nginx

# Strategy 4: Numeric UID (Kubernetes SecurityContext)
FROM node:20-alpine
USER 1001:1001

# Permissions per non-root user
FROM node:20-alpine
```

```
RUN adduser -D -u 1001 nodejs
WORKDIR /app
COPY --chown=nodejs:nodejs . .
USER nodejs
```

10.5.2 Read-Only Root Filesystem

Listing 10.8: Read-Only Filesystem in Docker Compose

```
version: '3.8'

services:
  app:
    image: myapp:latest
    read_only: true  # Root filesystem read-only
    tmpfs:
      - /tmp:size=100M,mode=1777
      - /var/run:size=10M,mode=755
    volumes:
      # Writable volumes only where necessary
      - app-cache:/app/cache:rw
      - app-logs:/app/logs:rw

volumes:
  app-cache:
  app-logs:
```

Listing 10.9: Read-Only in Kubernetes

```
# kubernetes-security.yaml
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1001
    fsGroup: 1001
    seccompProfile:
      type: RuntimeDefault

  containers:
  - name: app
    image: myapp:latest
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
          - ALL
    volumeMounts:
    - name: cache
      mountPath: /tmp
    - name: logs
      mountPath: /var/log

  volumes:
```

```
  - name: cache
    emptyDir: {}
  - name: logs
    emptyDir: {}
```

10.5.3 Security Scanning

Trivy - Comprehensive Scanning

Listing 10.10: Trivy Security Scanning

```
# Install Trivy
curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/
   contrib/install.sh | sh -s -- -b /usr/local/bin

# Scan image per vulnerabilities
trivy image myapp:latest

# Scan solo CRITICAL e HIGH
trivy image --severity CRITICAL,HIGH myapp:latest

# Output formattato
trivy image --format json --output results.json myapp:latest
trivy image --format sarif --output trivy-results.sarif myapp:latest

# Scan Dockerfile
trivy config Dockerfile

# Scan filesystem
trivy fs /path/to/project

# Scan con exit code (CI/CD integration)
trivy image --exit-code 1 --severity CRITICAL myapp:latest

# Ignore unfixed vulnerabilities
trivy image --ignore-unfixed myapp:latest

# Scan con database update
trivy image --download-db-only
trivy image --skip-db-update myapp:latest
```

Docker Scout

Listing 10.11: Docker Scout Analysis

```
# Enable Docker Scout
docker scout quickview myapp:latest

# Detailed CVE report
docker scout cves myapp:latest

# Compare images
docker scout compare --to myapp:v1.0 myapp:latest

# Recommendations
docker scout recommendations myapp:latest

```

```
# SBOM (Software Bill of Materials)
docker scout sbom myapp:latest
```

Snyk Container Security

Listing 10.12: Snyk Scanning

```
# Install Snyk CLI
npm install -g snyk

# Authenticate
snyk auth

# Test image
snyk container test myapp:latest

# Monitor image in Snyk dashboard
snyk container monitor myapp:latest

# Test con severity threshold
snyk container test myapp:latest --severity-threshold=high

# Generate HTML report
snyk container test myapp:latest --json | snyk-to-html -o results.html
```

10.6 Network Security

10.6.1 Network Isolation

Listing 10.13: Network Segmentation

```
# docker-compose-network-security.yml
version: '3.8'

services:
  # Frontend (public)
  frontend:
    image: nginx:alpine
    networks:
      - public
      - frontend-backend
    ports:
      - "80:80"
      - "443:443"

  # Backend (internal)
  backend:
    image: myapp:latest
    networks:
      - frontend-backend
      - backend-database
    # No ports exposed externally

  # Database (isolated)
  database:
    image: postgres:15-alpine
    networks:
      - backend-database  # Solo backend può accedere
    # No external access

networks:
  public:
    driver: bridge
  frontend-backend:
    driver: bridge
    internal: false
  backend-database:
    driver: bridge
    internal: true  # No internet access
```

10.6.2 Kubernetes Network Policies

Listing 10.14: NetworkPolicy - Deny All by Default

```
# deny-all.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress

---
# Allow specific traffic
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-backend-to-db
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Egress
  egress:
  # Allow DNS
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
    ports:
    - protocol: UDP
      port: 53

  # Allow database access
  - to:
    - podSelector:
        matchLabels:
          app: postgres
    ports:
    - protocol: TCP
      port: 5432

---
# Allow ingress to frontend
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-to-frontend
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: frontend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

10.7 Resource Limits

10.7.1 Docker Resource Constraints

Listing 10.15: Resource Limits in Docker Compose

```
version: '3.8'

services:
  app:
    image: myapp:latest
    deploy:
      resources:
        limits:
          cpus: '1.5'          # Max 1.5 CPU cores
          memory: 1024M        # Max 1GB RAM
          pids: 100            # Max 100 processes
        reservations:
          cpus: '0.5'          # Guaranteed 0.5 CPU
          memory: 512M         # Guaranteed 512MB
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3

  # OOMKilled prevention
  database:
    image: postgres:15
    deploy:
      resources:
        limits:
          memory: 2G
        reservations:
          memory: 1G
    # Memory swappiness (0-100, lower = less swap)
    sysctls:
      - vm.swappiness=10
```

Listing 10.16: Docker Run Resource Limits

```
# CPU limits
docker run -d \
  --cpus="1.5" \
  --cpu-shares=1024 \
  myapp:latest

# Memory limits
docker run -d \
  --memory="1g" \
  --memory-reservation="512m" \
  --memory-swap="2g" \
  --oom-kill-disable=false \
  myapp:latest

# Disk I/O limits
docker run -d \
  --device-read-bps /dev/sda:10mb \
  --device-write-bps /dev/sda:10mb \
  myapp:latest

# PIDs limit
docker run -d \
  --pids-limit=100 \
  myapp:latest
```

10.8 Secrets Management

10.8.1 Docker Secrets (Swarm)

Listing 10.17: Docker Secrets Best Practices

```
# Create secret from file
docker secret create db_password /path/to/password.txt

# Create secret from stdin
echo "supersecretpassword" | docker secret create db_password -

# Create secret with labels
docker secret create db_password - <<EOF
$(openssl rand -base64 32)
EOF

# Use in stack
cat <<EOF | docker stack deploy -c - myapp
version: '3.8'
```

```
services:
  app:
    image: myapp:latest
    secrets:
      - db_password
      - api_key
    environment:
      DB_PASSWORD_FILE: /run/secrets/db_password

secrets:
  db_password:
    external: true
  api_key:
    external: true
EOF

# Rotate secret
docker secret create db_password_v2 - < new_password.txt
docker service update \
  --secret-rm db_password \
  --secret-add source=db_password_v2,target=db_password \
  myapp
```

10.8.2 Kubernetes Secrets

Listing 10.18: Kubernetes Secrets with Encryption

```
# Create generic secret
kubectl create secret generic db-credentials \
  --from-literal=username=admin \
  --from-literal=password=$(openssl rand -base64 32)

# Create from file
kubectl create secret generic tls-cert \
  --from-file=tls.crt=./server.crt \
  --from-file=tls.key=./server.key

# Encryption at rest configuration
# /etc/kubernetes/enc/enc.yaml
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: $(head -c 32 /dev/urandom | base64)
      - identity: {}

# Apply encryption config in API server
# --encryption-provider-config=/etc/kubernetes/enc/enc.yaml
```

10.8.3 External Secrets Operator

Listing 10.19: HashiCorp Vault Integration

```
# Install External Secrets Operator
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets external-secrets/external-secrets

# SecretStore (Vault backend)
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: vault-backend
  namespace: production
spec:
  provider:
    vault:
      server: "https://vault.example.com"
      path: "secret"
      version: "v2"
      auth:
        kubernetes:
          mountPath: "kubernetes"
          role: "production"

---
# ExternalSecret
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: database-credentials
  namespace: production
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: vault-backend
    kind: SecretStore
  target:
    name: db-credentials
    creationPolicy: Owner
  data:
  - secretKey: username
    remoteRef:
      key: database/prod
      property: username
  - secretKey: password
    remoteRef:
      key: database/prod
      property: password
```

10.9 Image Signing and Verification

10.9.1 Cosign - Image Signing

Listing 10.20: Cosign Image Signing

```
# Install Cosign
go install github.com/sigstore/cosign/v2/cmd/cosign@latest

# Generate key pair
```

```
cosign generate-key-pair

# Sign image
cosign sign --key cosign.key myregistry.io/myapp:v1.0.0

# Verify signature
cosign verify --key cosign.pub myregistry.io/myapp:v1.0.0

# Keyless signing (Sigstore)
COSIGN_EXPERIMENTAL=1 cosign sign myregistry.io/myapp:v1.0.0

# Attach SBOM
syft myapp:latest -o spdx-json > sbom.spdx.json
cosign attach sbom --sbom sbom.spdx.json myregistry.io/myapp:v1.0.0

# Policy enforcement (Kubernetes)
apiVersion: v1
kind: Pod
metadata:
  name: signed-pod
  annotations:
    cosign.sigstore.dev/signature: "verified"
spec:
  containers:
  - name: app
    image: myregistry.io/myapp:v1.0.0
```

10.10 Production Deployment Checklist

Security & Best Practices Checklist

Dockerfile:

Multi-stage build per minimizzare image size

Non-root user configurato

No secrets hardcoded

Health check implementato

.dockerignore completo

Base image aggiornata (no vulnerabilities)

Security:

Image scanning (Trivy/Snyk) in CI/CD

Read-only root filesystem

Capabilities dropped (Linux capabilities)

Secrets in external vault (no env vars)

Network policies configurate

Image signing con Cosign

Resources:

CPU/Memory limits definiti

Resource requests configurati

PID limits per prevenire fork bombs

Disk I/O limits se necessario

Observability:

Structured logging implementato

Prometheus metrics exposed

Health/Readiness probes configurati

Distributed tracing setup

Compliance:

SBOM generato e attached

License compliance verificata

Audit logs abilitati

Data encryption at rest

10.11 Common Security Anti-Patterns

- **Anti-Pattern**: Running as root user
  - **Risk**: Container breakout, privilege escalation
  - **Fix**: USER directive, SecurityContext in K8s
- **Anti-Pattern**: Secrets in ENV variables
  - **Risk**: Visible in docker inspect, logs
  - **Fix**: File-based secrets, Vault integration
- **Anti-Pattern**: Using 'latest' tag in production
  - **Risk**: Non-deterministic deployments
  - **Fix**: Semantic versioning, SHA digests
- **Anti-Pattern**: No resource limits
  - **Risk**: Resource exhaustion, noisy neighbor
  - **Fix**: Explicit CPU/Memory limits
- **Anti-Pattern**: Ignoring CVE vulnerabilities
  - **Risk**: Exploitable vulnerabilities in production
  - **Fix**: Automated scanning, patch management

10.12 Performance Optimization

10.12.1 Image Size Reduction

Listing 10.21: Image Size Comparison

```
# Bad: Ubuntu base (hundreds of MB)
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y python3
# Result: ~500MB

# Better: Slim variant
FROM python:3.11-slim
# Result: ~150MB

# Best: Alpine (minimal)
FROM python:3.11-alpine
# Result: ~50MB

# Distroless (no shell, minimal attack surface)
FROM gcr.io/distroless/python3
# Result: ~60MB, ultra-secure
```

10.13 Errori Comuni

- **Errore**: Layer caching inefficace
  - **Sintomo**: Build sempre da zero, lenti
  - **Soluzione**: Ordine corretto layer, BuildKit cache
- **Errore**: Context troppo grande
  - **Sintomo**: "Sending build context" richiede minuti
  - **Soluzione**: .dockerignore completo
- **Errore**: Permessi file sbagliati con COPY
  - **Sintomo**: Permission denied quando esegue app
  - **Soluzione**: –chown flag in COPY

10.14 Riepilogo

Best practices Docker richiedono Dockerfile optimization (multi-stage, layer caching), security hardening (non-root, read-only FS, network policies), secrets management (Vault, External Secrets), vulnerability scanning (Trivy, Snyk), e resource limits. Production deployments devono seguire checklist completa per security, performance, e compliance.

10.15 Riferimenti

- Docker Security: https://docs.docker.com/engine/security/
- CIS Docker Benchmark: https://www.cisecurity.org/benchmark/docker
- OWASP Container Security: https://owasp.org/www-project-docker-top-10/

- Trivy: `https://trivy.dev/`
- Sigstore Cosign: `https://docs.sigstore.dev/cosign/`

Appendice A

Appendice: Cheat Sheet Comandi Docker

A.1 Container Management

A.1.1 Lifecycle Commands

Listing A.1: Container Basics

```
# Run container
docker run -d --name myapp nginx:alpine
docker run -it --rm alpine sh              # Interactive, auto-remove
docker run -d -p 8080:80 nginx             # Port mapping
docker run -d -v /data:/app/data myapp     # Volume mount

# Start/Stop/Restart
docker start container_name
docker stop container_name
docker restart container_name
docker pause container_name                # Pause processes
docker unpause container_name

# Remove containers
docker rm container_name                   # Remove stopped container
docker rm -f container_name                # Force remove running
docker container prune                     # Remove all stopped
docker rm $(docker ps -aq)                 # Remove all containers

# Execute commands in running container
docker exec -it container_name bash
docker exec container_name ls /app
docker exec -u root container_name sh      # As different user
```

A.1.2 Inspection & Monitoring

Listing A.2: Container Info

```
# List containers
docker ps                                  # Running containers
docker ps -a                               # All containers
docker ps -q                               # Only IDs
docker ps --filter "status=exited"
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
```

```

# Inspect container
docker inspect container_name
docker inspect --format='{{.State.Status}}' container_name
docker inspect --format='{{.NetworkSettings.IPAddress}}' container_name

# Logs
docker logs container_name
docker logs -f container_name                # Follow
docker logs --tail 100 container_name
docker logs --since 2024-01-01 container_name
docker logs -t container_name                # With timestamps

# Stats & Resource Usage
docker stats                                 # Real-time stats
docker stats --no-stream                     # One-time snapshot
docker top container_name                    # Processes in container

# Events
docker events                                # Real-time events
docker events --since 1h --filter type=container
```

A.1.3 Advanced Container Operations

Listing A.3: Advanced Commands

```
# Copy files to/from container
docker cp local_file.txt container:/path/
docker cp container:/app/log.txt ./

# Commit container to image
docker commit container_name new_image:tag

# Export/Import container filesystem
docker export container_name > container.tar
docker import container.tar new_image:tag

# Attach to running container
docker attach container_name                 # Attach to STDIN/STDOUT

# Rename container
docker rename old_name new_name

# Update container resources
docker update --cpus 2 --memory 1g container_name

# Wait for container to stop
docker wait container_name
```

A.2 Image Management

A.2.1 Image Operations

Listing A.4: Image Commands

```
# Pull images
```

```
docker pull nginx:alpine
docker pull --platform linux/amd64 ubuntu:22.04

# List images
docker images
docker images -a                          # Include intermediates
docker images --filter "dangling=true"    # Untagged images
docker images --format "table {{.Repository}}:{{.Tag}}\t{{.Size}}"

# Tag images
docker tag myapp:latest myregistry.io/myapp:v1.0.0
docker tag myapp:latest myapp:stable

# Push to registry
docker push myregistry.io/myapp:v1.0.0

# Remove images
docker rmi image_name:tag
docker rmi -f image_id                    # Force remove
docker image prune                        # Remove dangling
docker image prune -a                     # Remove unused
docker rmi $(docker images -q)            # Remove all

# Save/Load images
docker save myapp:latest > myapp.tar
docker save myapp:latest | gzip > myapp.tar.gz
docker load < myapp.tar

# Image history
docker history myapp:latest
docker history --no-trunc myapp:latest    # Full commands

# Inspect image
docker inspect myapp:latest
docker inspect --format='{{.Config.Env}}' myapp:latest
```

A.2.2 Build Commands

Listing A.5: Docker Build

```
# Basic build
docker build -t myapp:latest .
docker build -t myapp:v1.0.0 -f Dockerfile.prod .

# Build arguments
docker build --build-arg VERSION=1.0.0 -t myapp .
docker build --build-arg HTTP_PROXY=http://proxy:8080 -t myapp .

# BuildKit features
export DOCKER_BUILDKIT=1
docker build --cache-from myapp:latest -t myapp:new .
docker build --secret id=npmrc,src=$HOME/.npmrc -t myapp .
docker build --ssh default -t myapp .

# Multi-platform build
docker buildx create --use
docker buildx build --platform linux/amd64,linux/arm64 \
  -t myapp:latest --push .

# Target specific stage
docker build --target production -t myapp:prod .

# No cache
docker build --no-cache -t myapp .
docker build --pull -t myapp .             # Pull base image

# Squash layers (experimental)
docker build --squash -t myapp .
```

A.3 Volume Management

A.3.1 Volume Commands

Listing A.6: Volumes

```
# Create volume
docker volume create myvolume
docker volume create --driver local \
  --opt type=nfs \
  --opt o=addr=192.168.1.1,rw \
  --opt device=:/path/to/dir \
  nfs-volume

# List volumes
docker volume ls
docker volume ls --filter "dangling=true"

# Inspect volume
docker volume inspect myvolume

# Remove volumes
docker volume rm myvolume
docker volume prune                         # Remove unused
docker volume prune -f                      # No confirmation

# Use volume in container
docker run -d -v myvolume:/app/data myapp
docker run -d -v /host/path:/container/path:ro myapp
docker run -d --mount source=myvolume,target=/data myapp
```

A.4 Network Management

A.4.1 Network Commands

Listing A.7: Docker Networks

```
# Create networks
docker network create mynetwork
docker network create --driver bridge mybridge
docker network create --driver overlay --attachable myoverlay
docker network create --subnet 172.20.0.0/16 custom-net

```

```
# List networks
docker network ls
docker network ls --filter driver=bridge

# Inspect network
docker network inspect mynetwork
docker network inspect --format='{{json .Containers}}' mynetwork

# Connect/Disconnect containers
docker network connect mynetwork container_name
docker network disconnect mynetwork container_name

# Remove networks
docker network rm mynetwork
docker network prune                        # Remove unused

# Create container on specific network
docker run -d --network mynetwork --name app myapp
docker run -d --network mynetwork --ip 172.20.0.10 myapp
```

A.5 Docker Compose

A.5.1 Compose Commands

Listing A.8: Docker Compose

```
# Start services
docker-compose up                           # Foreground
docker-compose up -d                        # Detached
docker-compose up --build                   # Rebuild images
docker-compose up --force-recreate          # Recreate containers
docker-compose up --scale app=3             # Scale service

# Stop services
docker-compose stop
docker-compose down                         # Stop and remove
docker-compose down -v                      # Remove volumes too
docker-compose down --rmi all               # Remove images

# View services
docker-compose ps
docker-compose ps -a
docker-compose top

# Logs
docker-compose logs
docker-compose logs -f app
docker-compose logs --tail=100

# Execute commands
docker-compose exec app bash
docker-compose exec -T app npm test        # No TTY
docker-compose run --rm app npm install    # One-off command

# Build
docker-compose build
docker-compose build --no-cache app
```

```
docker-compose build --pull

# Configuration
docker-compose config                      # Validate and view
docker-compose config --services           # List services
docker-compose config --volumes

# Pull images
docker-compose pull
docker-compose pull app

# Restart services
docker-compose restart
docker-compose restart app

# Pause/Unpause
docker-compose pause
docker-compose unpause
```

A.6 Docker Swarm

A.6.1 Swarm Management

Listing A.9: Swarm Commands

```
# Initialize swarm
docker swarm init
docker swarm init --advertise-addr 192.168.1.10

# Join swarm
docker swarm join --token TOKEN 192.168.1.10:2377
docker swarm join-token worker             # Get worker token
docker swarm join-token manager            # Get manager token

# Leave swarm
docker swarm leave
docker swarm leave --force                 # Force manager leave

# Node management
docker node ls
docker node inspect node_name
docker node update --availability drain node_name
docker node update --label-add type=worker node_name
docker node rm node_name

# Service management
docker service create --name web --replicas 3 -p 80:80 nginx
docker service ls
docker service ps web
docker service inspect web
docker service logs web

# Scale service
docker service scale web=5

# Update service
docker service update --image nginx:alpine web
docker service update --replicas 10 web
docker service update --rollback web

# Remove service
docker service rm web

# Stack management
docker stack deploy -c docker-compose.yml mystack
docker stack ls
docker stack services mystack
docker stack ps mystack
docker stack rm mystack
```

A.7 Registry & Authentication

A.7.1 Registry Commands

Listing A.10: Registry Operations

```
# Login to registry
docker login
docker login myregistry.io
docker login -u username -p password myregistry.io
docker login ghcr.io -u USERNAME --password-stdin < token.txt

# Logout
docker logout
docker logout myregistry.io

# Search images
docker search nginx
docker search --filter stars=100 nginx

# Push/Pull with different registries
docker pull myregistry.io/myapp:v1.0.0
docker push ghcr.io/myorg/myapp:latest

# Tag for different registries
docker tag myapp:latest docker.io/myorg/myapp:latest
docker tag myapp:latest ghcr.io/myorg/myapp:latest
docker tag myapp:latest gcr.io/myproject/myapp:latest
```

A.8 System Management

A.8.1 System Commands

Listing A.11: System Operations

```
# System info
docker info
docker version
docker system df                        # Disk usage

# Clean up
docker system prune                     # Remove unused data
docker system prune -a                     # Remove all unused
docker system prune --volumes              # Include volumes
docker system prune -a --volumes -f        # Force, all+volumes

# Events
docker system events
docker system events --since 1h
docker system events --filter type=container

# Check plugins
docker plugin ls
docker plugin install plugin_name
docker plugin disable plugin_name
docker plugin rm plugin_name
```

A.9 Security & Scanning

A.9.1 Security Commands

Listing A.12: Security & Scanning

```
# Scan image (Docker Scout)
docker scout quickview myapp:latest
docker scout cves myapp:latest
docker scout recommendations myapp:latest

# Trivy scanning
trivy image myapp:latest
trivy image --severity HIGH,CRITICAL myapp:latest
trivy image --exit-code 1 myapp:latest
trivy fs .                                 # Scan filesystem

# Image signing (Cosign)
cosign sign --key cosign.key myregistry.io/myapp:v1.0.0
cosign verify --key cosign.pub myregistry.io/myapp:v1.0.0

# Secrets management (Swarm)
echo "password" | docker secret create db_pass -
docker secret ls
docker secret inspect db_pass
docker secret rm db_pass

# Config management
docker config create nginx_conf nginx.conf
docker config ls
docker config inspect nginx_conf
docker config rm nginx_conf
```

A.10 Advanced Debugging

A.10.1 Debugging Commands

Listing A.13: Debugging

```
# Container inspection
```

```
docker inspect --format='{{json .State}}' container | jq
docker inspect --format='{{.NetworkSettings.Networks}}' container

# Check container processes
docker top container
docker stats container --no-stream

# Port mappings
docker port container

# Filesystem changes
docker diff container

# Resource usage
docker stats --all --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage
    }}"

# Network troubleshooting
docker network inspect bridge
docker exec container ping other_container
docker exec container netstat -tuln
docker exec container ip addr show

# Health check status
docker inspect --format='{{.State.Health.Status}}' container
docker inspect --format='{{json .State.Health}}' container | jq

# Check why container exited
docker inspect --format='{{.State.ExitCode}}' container
docker logs --tail 50 container
```

A.11 Context & Remote Docker

A.11.1 Context Management

Listing A.14: Docker Context

```
# List contexts
docker context ls

# Create context
docker context create remote-docker \
  --docker "host=ssh://user@remote-host"

# Use context
docker context use remote-docker
docker context use default

# Inspect context
docker context inspect remote-docker

# Remove context
docker context rm remote-docker

# Remote Docker via SSH
docker -H ssh://user@remote-host ps
export DOCKER_HOST=ssh://user@remote-host
```

A.12 BuildKit Advanced

A.12.1 BuildKit Commands

Listing A.15: BuildKit

```
# Enable BuildKit
export DOCKER_BUILDKIT=1

# Buildx commands
docker buildx create --name mybuilder --use
docker buildx ls
docker buildx inspect mybuilder
docker buildx use mybuilder

# Multi-platform build
docker buildx build --platform linux/amd64,linux/arm64 \
  -t myapp:latest --push .

# Build with cache
docker buildx build \
  --cache-from type=registry,ref=myapp:buildcache \
  --cache-to type=registry,ref=myapp:buildcache,mode=max \
  -t myapp:latest .

# Build with secrets
docker buildx build \
  --secret id=aws,src=$HOME/.aws/credentials \
  -t myapp .

# Build with SSH
docker buildx build --ssh default -t myapp .

# Inspect build
docker buildx imagetools inspect myapp:latest

# Remove builder
docker buildx rm mybuilder
```

A.13 Performance & Optimization

A.13.1 Performance Commands

Listing A.16: Performance

```
# Benchmark build
time docker build -t myapp .

# Check layer sizes
docker history myapp:latest --human=true --no-trunc=false

# Analyze image
dive myapp:latest                          # Interactive layer analysis

# Check disk usage
docker system df -v

```

```
# Container resource limits
docker run -d \
  --cpus="1.5" \
  --memory="1g" \
  --memory-reservation="512m" \
  --pids-limit=100 \
  myapp

# Check container resource usage
docker stats --no-stream --format \
  "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO
    }}"
```

A.14 Useful One-Liners

A.14.1 Cheat Sheet One-Liners

Listing A.17: Useful One-Liners

```
# Remove all stopped containers
docker rm $(docker ps -aq -f status=exited)

# Remove all dangling images
docker rmi $(docker images -q -f dangling=true)

# Stop all running containers
docker stop $(docker ps -q)

# Get container IP address
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end
    }}' container

# Get container logs for last hour
docker logs --since 1h container

# Follow logs from all compose services
docker-compose logs -f --tail=100

# Execute command in all running containers
docker ps -q | xargs -I {} docker exec {} command

# Backup volume
docker run --rm -v myvolume:/data -v $(pwd):/backup \
  alpine tar czf /backup/backup.tar.gz /data

# Restore volume
docker run --rm -v myvolume:/data -v $(pwd):/backup \
  alpine tar xzf /backup/backup.tar.gz -C /

# Get all container IPs
docker ps -q | xargs docker inspect \
  --format='{{.Name}} {{range.NetworkSettings.Networks}}{{.IPAddress}}{{
    end}}'

# Clean everything (DANGEROUS!)
docker system prune -a --volumes -f

```

```
# Monitor container resources real-time
watch -n 1 'docker stats --no-stream'

# Get environment variables from container
docker inspect --format='{{.Config.Env}}' container

# Check which containers use an image
docker ps -a --filter ancestor=myapp:latest

# Export container as tarball
docker export container > container-backup.tar

# Get container creation time
docker inspect --format='{{.Created}}' container
```

A.15 Environment Variables

A.15.1 Useful Docker Environment Variables

Listing A.18: Environment Variables

```
# Enable BuildKit
export DOCKER_BUILDKIT=1
export COMPOSE_DOCKER_CLI_BUILD=1

# Docker host
export DOCKER_HOST=tcp://192.168.1.10:2376
export DOCKER_HOST=ssh://user@remote
export DOCKER_HOST=unix:///var/run/docker.sock

# Docker TLS
export DOCKER_TLS_VERIFY=1
export DOCKER_CERT_PATH=/path/to/certs

# Registry config
export DOCKER_CONFIG=$HOME/.docker

# Buildx builder
export BUILDX_BUILDER=mybuilder

# Compose project
export COMPOSE_PROJECT_NAME=myproject
export COMPOSE_FILE=docker-compose.yml:docker-compose.override.yml

# Log driver
export DOCKER_LOGGING_DRIVER=json-file

# Default platform
export DOCKER_DEFAULT_PLATFORM=linux/amd64
```

A.16 Quick Reference Tables

A.16.1 Common Flags

Flag	**Description**
-d	Detached mode (background)
-it	Interactive + TTY
–rm	Auto-remove on exit
-p	Port mapping (host:container)
-v	Volume mount
-e	Environment variable
–name	Container name
–network	Network to connect
-u	User (UID:GID)
–restart	Restart policy
-w	Working directory
–entrypoint	Override entrypoint
–env-file	Load env from file
–link	Link to another container
-h	Hostname

A.16.2 Restart Policies

Policy	**Behavior**
no	Never restart (default)
on-failure	Restart on non-zero exit
always	Always restart
unless-stopped	Restart unless manually stopped

A.16.3 Network Drivers

Driver	**Use Case**
bridge	Single host networking
host	Use host network stack
overlay	Swarm multi-host networking
macvlan	Assign MAC address to container
none	Disable networking

A.17 Riferimenti Rapidi

- Docker CLI Reference: `https://docs.docker.com/engine/reference/commandline/cli/`
- Docker Compose CLI: `https://docs.docker.com/compose/reference/`
- Dockerfile Reference: `https://docs.docker.com/engine/reference/builder/`
- Docker Hub: `https://hub.docker.com/`

Appendice B

Appendice: Progetti Completi

B.1 Introduzione

Questa appendice contiene progetti completi end-to-end per consolidare le competenze Docker acquisite. Ogni progetto include Dockerfile ottimizzato, docker-compose.yml, CI/CD pipeline, monitoring setup, e deployment strategy.

Progetti Inclusi

1. **Full-Stack Web Application**: React + Node.js + PostgreSQL + Redis
2. **Microservices Architecture**: API Gateway + 3 Services + Message Queue
3. **WordPress Production Setup**: Nginx + PHP-FPM + MySQL + Redis
4. **Data Pipeline**: Apache Airflow + Postgres + Redis
5. **Monitoring Stack**: Prometheus + Grafana + Loki + Alertmanager
6. **CI/CD Platform**: Jenkins + Docker-in-Docker + Registry

B.2 Progetto 1: Full-Stack MERN Application

B.2.1 Architettura

```
+-------------
|   Nginx     | :80 (Reverse Proxy + Static)
+------------+
       |
   +--------------------
   |        |             |
+----  +--------  +------
|React|  |Node.js  |  | Redis |
| SPA |  |  API    |  | Cache |
+-----+  +--------+  +-------+
              |
         +---------
         |PostgreSQL|
         | Database |
         +----------+
```

B.2.2 Directory Structure

Listing B.1: Project Structure

```
fullstack-app/
+-- frontend/
|   +-- Dockerfile
|   +-- package.json
|   +-- src/
|   +-- public/
+-- backend/
|   +-- Dockerfile
|   +-- package.json
|   +-- src/
|   +-- tests/
+-- nginx/
|   +-- Dockerfile
|   +-- nginx.conf
+-- docker-compose.yml
+-- docker-compose.prod.yml
+-- .env.example
+-- .dockerignore
+-- .github/
    +-- workflows/
        +-- ci-cd.yml
```

B.2.3 Frontend Dockerfile

Listing B.2: frontend/Dockerfile

```
# syntax=docker/dockerfile:1.4

# Stage 1: Build
FROM node:20-alpine AS builder

WORKDIR /app

# Install dependencies
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci

# Build application
COPY . .
RUN npm run build

# Stage 2: Production
FROM nginx:alpine

# Copy built assets
COPY --from=builder /app/build /usr/share/nginx/html

# Custom nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Health check
HEALTHCHECK --interval=30s --timeout=3s \
    CMD wget --quiet --tries=1 --spider http://localhost/health || exit
        1

EXPOSE 80
```

B.2.4 Backend Dockerfile

Listing B.3: backend/Dockerfile

```
# syntax=docker/dockerfile:1.4

FROM node:20-alpine AS base
RUN apk add --no-cache dumb-init
WORKDIR /app

# Dependencies
FROM base AS dependencies
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci --only=production

# Build
FROM base AS builder
COPY package*.json ./
RUN --mount=type=cache,target=/root/.npm \
    npm ci
COPY . .
RUN npm run build

# Test
FROM builder AS test
ENV NODE_ENV=test
RUN npm run test

# Production
FROM base AS production

# Security: non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001

# Copy artifacts
COPY --from=dependencies --chown=nodejs:nodejs /app/node_modules ./
   node_modules
COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --chown=nodejs:nodejs package.json ./

USER nodejs

HEALTHCHECK --interval=30s --timeout=3s --start-period=40s \
    CMD node healthcheck.js || exit 1

EXPOSE 3000

ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "dist/server.js"]
```

B.2.5 Docker Compose - Development

Listing B.4: docker-compose.yml

```
version: '3.8'

services:
  # PostgreSQL Database
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: ${DB_NAME:-appdb}
      POSTGRES_USER: ${DB_USER:-appuser}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-changeme}
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./backend/init-db.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${DB_USER:-appuser}"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - backend

  # Redis Cache
  redis:
    image: redis:7-alpine
    command: redis-server --appendonly yes
    volumes:
      - redis-data:/data
    ports:
      - "6379:6379"
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 5
    networks:
      - backend

  # Backend API
  backend:
    build:
      context: ./backend
      target: development
    environment:
      NODE_ENV: development
      DATABASE_URL: postgresql://${DB_USER:-appuser}:${DB_PASSWORD:-changeme}@postgres:5432/${DB_NAME:-appdb}
      REDIS_URL: redis://redis:6379
      JWT_SECRET: ${JWT_SECRET:-dev-secret}
    volumes:
      - ./backend/src:/app/src
      - ./backend/package.json:/app/package.json
      - backend-modules:/app/node_modules
    ports:
      - "3000:3000"
      - "9229:9229"  # Debugger
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy
    networks:
      - backend
      - frontend
    command: npm run dev

  # Frontend React App
  frontend:
    build:
      context: ./frontend
      target: development
    environment:
      REACT_APP_API_URL: http://localhost:3000
      CHOKIDAR_USEPOLLING: "true"
    volumes:
      - ./frontend/src:/app/src
      - ./frontend/public:/app/public
      - ./frontend/package.json:/app/package.json
      - frontend-modules:/app/node_modules
    ports:
      - "8080:3000"
    networks:
      - frontend
    command: npm start

  # Nginx Reverse Proxy
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx/nginx.dev.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "80:80"
    depends_on:
      - backend
      - frontend
    networks:
      - frontend

  # Adminer (Database GUI)
  adminer:
    image: adminer:latest
    ports:
      - "8081:8080"
    networks:
      - backend
    environment:
      ADMINER_DEFAULT_SERVER: postgres

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge

volumes:
  postgres-data:
  redis-data:
  backend-modules:
  frontend-modules:
```

B.2.6 Docker Compose - Production

Listing B.5: docker-compose.prod.yml

```
version: '3.8'

services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password
    volumes:
      - postgres-data:/var/lib/postgresql/data
    networks:
      - backend
    secrets:
      - db_password
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: '1'
          memory: 2G
        reservations:
          cpus: '0.5'
          memory: 1G

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis-data:/data
    networks:
      - backend
    deploy:
      replicas: 1
      resources:
        limits:
          cpus: '0.5'
          memory: 512M

  backend:
    image: myregistry.io/backend:${VERSION:-latest}
    environment:
      NODE_ENV: production
      DATABASE_URL_FILE: /run/secrets/database_url
      REDIS_URL_FILE: /run/secrets/redis_url
      JWT_SECRET_FILE: /run/secrets/jwt_secret
    networks:
      - backend
      - frontend
    secrets:
      - database_url
      - redis_url
      - jwt_secret
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
        order: start-first
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: ’1’
          memory: 1G
        reservations:
          cpus: ’0.25’
          memory: 256M
    healthcheck:
      test: ["CMD", "node", "healthcheck.js"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 40s

  frontend:
    image: myregistry.io/frontend:${VERSION:-latest}
    networks:
      - frontend
    deploy:
      replicas: 2
      resources:
        limits:
          cpus: ’0.5’
          memory: 256M

  nginx:
    image: myregistry.io/nginx:${VERSION:-latest}
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/ssl:/etc/nginx/ssl:ro
    networks:
      - frontend
    depends_on:
      - backend
      - frontend
    deploy:
      replicas: 2
      resources:
        limits:
          cpus: ’0.5’
          memory: 256M

secrets:
  db_password:
    external: true
  database_url:
    external: true
  redis_url:
    external: true
  jwt_secret:
    external: true

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
    internal: true

volumes:
  postgres-data:
  redis-data:
```

B.2.7 Nginx Configuration

Listing B.6: nginx/nginx.conf

```
upstream backend {
    least_conn;
    server backend:3000 max_fails=3 fail_timeout=30s;
}

upstream frontend {
    server frontend:80;
}

# Rate limiting
limit_req_zone $binary_remote_addr zone=api_limit:10m rate=10r/s;
limit_conn_zone $binary_remote_addr zone=addr:10m;

server {
    listen 80;
    server_name example.com;

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Strict-Transport-Security "max-age=31536000" always;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css application/json application/javascript;
    gzip_min_length 1000;

    # API endpoints
    location /api {
        limit_req zone=api_limit burst=20 nodelay;
        limit_conn addr 10;

        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection ’upgrade’;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;

        # Timeouts
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # Frontend SPA
    location / {
        proxy_pass http://frontend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        # SPA routing
        try_files $uri $uri/ /index.html;
    }

    # Static assets caching
    location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg|woff|woff2|ttf)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # Health check endpoint
    location /health {
        access_log off;
        return 200 "OK\n";
        add_header Content-Type text/plain;
    }
}
```

B.2.8 GitHub Actions CI/CD

Listing B.7: .github/workflows/ci-cd.yml

```
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
```

```

env:
  REGISTRY: ghcr.io
  IMAGE_PREFIX: ${{ github.repository }}

jobs:
  test-backend:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: test
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ’20’
          cache: ’npm’
          cache-dependency-path: backend/package-lock.json

      - name: Install dependencies
        working-directory: backend
        run: npm ci

      - name: Run linter
        working-directory: backend
        run: npm run lint

      - name: Run tests
        working-directory: backend
        run: npm test
        env:
          DATABASE_URL: postgresql://postgres:test@localhost:5432/testdb

      - name: Build
        working-directory: backend
        run: npm run build

  test-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ’20’
          cache: ’npm’
          cache-dependency-path: frontend/package-lock.json

      - name: Install dependencies
        working-directory: frontend
        run: npm ci

      - name: Run linter
        working-directory: frontend
        run: npm run lint

      - name: Run tests
        working-directory: frontend
        run: npm test -- --coverage

      - name: Build
        working-directory: frontend
        run: npm run build

  build-and-push:
    needs: [test-backend, test-frontend]
    runs-on: ubuntu-latest
    if: github.event_name != ’pull_request’
    permissions:
      contents: read
      packages: write

    strategy:
      matrix:
        service: [backend, frontend, nginx]

    steps:
      - uses: actions/checkout@v4

      - name: Login to GitHub Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_PREFIX }}/${{ matrix.service }}
          tags: |
            type=ref,event=branch
            type=sha
            type=raw,value=latest,enable={{is_default_branch}}

      - name: Build and push
        uses: docker/build-push-action@v5
        with:
          context: ./${{ matrix.service }}
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          cache-from: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_PREFIX }}/${{ matrix.service }}:buildcache
          cache-to: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_PREFIX }}/${{ matrix.service }}:buildcache,mode=max

  deploy:
    needs: build-and-push
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production

    steps:
      - uses: actions/checkout@v4

      - name: Deploy to production
        run: |
          echo "Deploying to production..."
          # Add deployment commands here
```

B.3 Progetto 2: Microservices Architecture

B.3.1 Architettura Microservices

```
                +-------------
                |   Traefik   | API Gateway
                |   :80/:443  |
                +------------+
                       |
     +----------------------------------
     |                 |                 |
+--------      +--------      +--------
| User    |      | Product |      | Order   |
| Service |      | Service |      | Service |
+--------+      +--------+      +--------+
     |                |                 |
+--------      +--------      +--------
|MongoDB  |      |Postgres |      |Postgres |
+---------+      +---------+      +---------+
     |                |                 |
     +---------------------------------+
                      |
                +-----------
                |  RabbitMQ  | Message Broker
                +------------+
```

B.3.2 Microservices Docker Compose

Listing B.8: microservices/docker-compose.yml

```
version: '3.8'

services:
  # API Gateway - Traefik
  traefik:
    image: traefik:v2.10
    command:
      - "--api.insecure=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--metrics.prometheus=true"
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    networks:
      - microservices

  # User Service
  user-service:
    build:
      context: ./services/user
    environment:
      MONGO_URL: mongodb://mongodb:27017/users
      RABBITMQ_URL: amqp://rabbitmq:5672
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.user.rule=PathPrefix(‘/api/users‘)"
      - "traefik.http.services.user.loadbalancer.server.port=3000"
    depends_on:
      - mongodb
      - rabbitmq
    networks:
      - microservices
    deploy:
      replicas: 3

  # Product Service
  product-service:
    build:
      context: ./services/product
    environment:
      DATABASE_URL: postgresql://postgres:password@product-db:5432/
          products
      RABBITMQ_URL: amqp://rabbitmq:5672
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.product.rule=PathPrefix(‘/api/products‘)"
      - "traefik.http.services.product.loadbalancer.server.port=3000"
    depends_on:
      - product-db
      - rabbitmq
    networks:
      - microservices
    deploy:
      replicas: 3

  # Order Service
  order-service:
    build:
      context: ./services/order
    environment:
      DATABASE_URL: postgresql://postgres:password@order-db:5432/orders
      RABBITMQ_URL: amqp://rabbitmq:5672
      USER_SERVICE_URL: http://user-service:3000
      PRODUCT_SERVICE_URL: http://product-service:3000
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.order.rule=PathPrefix(‘/api/orders‘)"
      - "traefik.http.services.order.loadbalancer.server.port=3000"
    depends_on:
      - order-db
      - rabbitmq
    networks:
      - microservices
    deploy:
      replicas: 3

  # MongoDB for User Service
  mongodb:
    image: mongo:7
    volumes:
      - mongodb-data:/data/db
    networks:
      - microservices

  # PostgreSQL for Product Service
  product-db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: products
      POSTGRES_PASSWORD: password
    volumes:
      - product-db-data:/var/lib/postgresql/data
    networks:
      - microservices

  # PostgreSQL for Order Service
  order-db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: orders
      POSTGRES_PASSWORD: password
    volumes:
      - order-db-data:/var/lib/postgresql/data
    networks:
      - microservices

  # RabbitMQ Message Broker
  rabbitmq:
    image: rabbitmq:3-management-alpine
    ports:
      - "5672:5672"
      - "15672:15672"
    volumes:
      - rabbitmq-data:/var/lib/rabbitmq
    networks:
      - microservices

  # Prometheus
  prometheus:
    image: prom/prometheus:v2.48.0
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-data:/prometheus
    ports:
      - "9090:9090"
    networks:
      - microservices

  # Grafana
  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"
    environment:
      GF_SECURITY_ADMIN_PASSWORD: admin
    volumes:
      - grafana-data:/var/lib/grafana
    networks:
      - microservices

networks:
  microservices:
    driver: overlay

volumes:
  mongodb-data:
  product-db-data:
  order-db-data:
  rabbitmq-data:
  prometheus-data:
  grafana-data:
```

B.4 Progetto 3: WordPress Production

B.4.1 WordPress Stack

Listing B.9: wordpress/docker-compose.yml

```
version: ’3.8’

services:
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - wordpress-data:/var/www/html:ro
      - ./ssl:/etc/nginx/ssl:ro
    ports:
      - "80:80"
      - "443:443"
    depends_on:
      - wordpress
    networks:
      - frontend
    deploy:
      replicas: 2
      resources:
        limits:
          cpus: ’0.5’
          memory: 256M

  wordpress:
    image: wordpress:php8.2-fpm-alpine
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_USER: ${DB_USER}
      WORDPRESS_DB_PASSWORD_FILE: /run/secrets/db_password
      WORDPRESS_DB_NAME: ${DB_NAME}
      WORDPRESS_REDIS_HOST: redis
      WORDPRESS_REDIS_PORT: 6379
    volumes:
      - wordpress-data:/var/www/html
      - ./php.ini:/usr/local/etc/php/conf.d/custom.ini
    networks:
      - frontend
      - backend
    secrets:
      - db_password
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: ’1’
          memory: 512M

  mysql:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD_FILE: /run/secrets/db_password
      MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
    volumes:
      - mysql-data:/var/lib/mysql
      - ./mysql-config:/etc/mysql/conf.d
    networks:
      - backend
    secrets:
      - db_password
      - db_root_password
    deploy:
      replicas: 1
      resources:
        limits:
          cpus: ’2’
          memory: 2G

  redis:
    image: redis:7-alpine
    command: redis-server --maxmemory 256mb --maxmemory-policy allkeys-lru
    volumes:
      - redis-data:/data
    networks:
      - backend
    deploy:
      replicas: 1

  # WP-CLI for management
  wpcli:
    image: wordpress:cli
    user: "33:33"
    volumes:
      - wordpress-data:/var/www/html
    networks:
      - backend
    command: wp --info
    profiles:
      - tools

secrets:
  db_password:
    external: true
  db_root_password:
    external: true

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
    internal: true

volumes:
  wordpress-data:
  mysql-data:
  redis-data:
```

B.5 Progetto 4: Data Pipeline con Airflow

B.5.1 Apache Airflow Stack

Listing B.10: airflow/docker-compose.yml

```
version: '3.8'

x-airflow-common: &airflow-common
  image: apache/airflow:2.7.0
  environment:
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:
        airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:
        airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth'
  volumes:
    - ./dags:/opt/airflow/dags
    - ./logs:/opt/airflow/logs
    - ./plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 5s
      retries: 5
    restart: always

  redis:
    image: redis:latest
    expose:
      - 6379
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
      timeout: 30s
      retries: 50
    restart: always

  airflow-webserver:
    <<: *airflow-common
    command: webserver
    ports:
      - 8080:8080
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always

  airflow-scheduler:
    <<: *airflow-common
    command: scheduler
    healthcheck:
      test: ["CMD-SHELL", ’airflow jobs check --job-type SchedulerJob --hostname "$${HOSTNAME}"’]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always

  airflow-worker:
    <<: *airflow-common
    command: celery worker
    healthcheck:
      test:
        - "CMD-SHELL"
        - ’celery --app airflow.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}"’
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    deploy:
      replicas: 3

  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", ’airflow jobs check --job-type TriggererJob --hostname "$${HOSTNAME}"’]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always

  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    command:
      - -c
      - |
        mkdir -p /sources/logs /sources/dags /sources/plugins
        chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
        exec /entrypoint airflow version

  flower:
    <<: *airflow-common
    command: celery flower
    ports:
      - 5555:5555
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always

volumes:
  postgres-db-volume:
```

B.6 Esercizi Pratici

B.6.1 Esercizio 1: Multi-Stage Build Optimization

Obiettivo: Ottimizzare un Dockerfile esistente riducendo image size del 70%.

Tasks:

1. Convertire single-stage a multi-stage build
2. Implementare BuildKit cache mounts
3. Configurare .dockerignore completo
4. Misurare reduction in image size e build time

B.6.2 Esercizio 2: Zero-Downtime Deployment

Obiettivo: Implementare blue-green deployment con Docker Swarm.

Tasks:

1. Setup Docker Swarm cluster (1 manager, 2 workers)
2. Deploy applicazione in ambiente "blue"
3. Deploy nuova versione in ambiente "green"
4. Implementare traffic switch script
5. Test rollback procedure

B.6.3 Esercizio 3: Complete Observability

Obiettivo: Setup monitoring completo per microservices.

Tasks:

1. Deploy Prometheus + Grafana + Loki stack
2. Instrumentare 3 microservices con metrics
3. Configurare centralized logging
4. Creare Grafana dashboards
5. Setup alert rules e notification channels

B.6.4 Esercizio 4: Security Hardening

Obiettivo: Applicare security best practices.

Tasks:

1. Scan existing images con Trivy/Snyk
2. Fix tutte le vulnerabilities CRITICAL/HIGH
3. Implementare non-root users
4. Configurare read-only filesystem
5. Setup secrets management con Vault
6. Implement image signing con Cosign

B.7 Progetti Challenge

B.7.1 Challenge 1: Production-Ready E-Commerce

Build complete e-commerce platform con:

- Frontend: Next.js
- Backend: NestJS API
- Databases: PostgreSQL + MongoDB + Redis
- Payment: Stripe integration
- Email: SMTP service
- Storage: MinIO (S3-compatible)
- Search: Elasticsearch
- CI/CD: GitHub Actions
- Monitoring: Prometheus/Grafana
- Requirements: 99.9% uptime, <200ms API latency

B.7.2 Challenge 2: Scalable Chat Application

Real-time chat con WebSocket:

- Backend: Socket.io cluster
- Message broker: Redis Pub/Sub
- Database: PostgreSQL
- Load balancer: HAProxy
- Horizontal scaling: 3-10 instances
- Features: Typing indicators, read receipts, file sharing
- Metrics: Messages/sec, active connections, latency

B.8 Soluzioni e Best Practices

B.8.1 Deployment Strategy Decision Matrix

Strategy	**Downtime**	**Resources**	**Complexity**
Recreate	High	Low	Low
Rolling	None	Medium	Medium
Blue-Green	None	High (2x)	Medium
Canary	None	Medium	High
A/B Testing	None	Medium	High

B.8.2 Resource Sizing Guide

Service Type	CPU	Memory
Node.js API	0.5-1 core	256-512MB
React SPA (built)	0.25 core	128MB
PostgreSQL	1-2 cores	1-2GB
Redis	0.5 core	256-512MB
Nginx	0.5 core	128-256MB

B.9 Riferimenti

- Docker Samples: https://github.com/docker/awesome-compose
- Production Patterns: https://github.com/docker/docker-bench-security
- Kubernetes Patterns: https://github.com/kubernetes/examples
- Microservices Examples: https://microservices.io/patterns/