

Containers

Docker → ambiente dove mettere le applicazioni di riferimento (di virtualizzazione) → applicazioni che puoi dare ai clienti persone che servono i file

Definizione di Container in Informatica

Un **container** è una struttura o un'unità che incapsula dati, risorse o applicazioni per facilitarne la gestione, il trasporto e l'esecuzione in ambienti controllati. Il concetto di container può essere applicato in diversi ambiti dell'informatica, tra cui la virtualizzazione, il web development e la gestione dei dati.

1. Container Software (Docker, Kubernetes)

I container software sono ambienti isolati che includono codice, dipendenze e configurazioni necessarie per eseguire un'applicazione.

✓ Vantaggi:

- **Portabilità:** Funzionano su qualsiasi piattaforma che supporta i container.
- **Isolamento:** Evitano conflitti tra applicazioni diverse.
- **Leggerezza:** Consumo di risorse inferiore rispetto alle macchine virtuali.
- **Scalabilità:** Facili da replicare e distribuire su larga scala.

✗ Svantaggi:

- **Sicurezza:** Condividono il kernel del sistema operativo, aumentando il rischio di attacchi.
 - **Gestione complessa:** Richiedono strumenti come Kubernetes per orchestrare più container.
 - **Persistenza dati:** I dati possono andare persi se non si usano volumi esterni.
-

2. Container nel Web Development (HTML & CSS)

Un container in web development è un elemento HTML (come `<div>` o `<section>`) che organizza il layout della pagina.

✓ Vantaggi:

- **Organizzazione del contenuto:** Aiuta a strutturare il layout.
- **Facile personalizzazione:** Consente l'applicazione di stili CSS mirati.
- **Compatibilità con Flexbox e Grid:** Facilita la creazione di layout responsivi.

✗ Svantaggi:

- **Eccessivo annidamento:** Troppi container possono rendere il codice difficile da leggere.
 - **Performance:** Un uso errato può rallentare il rendering della pagina.
-

3. Container nelle Strutture Dati (Array, Liste, HashMap)

Un container in programmazione è una struttura dati che memorizza e organizza oggetti (es. array, liste, stack).

✓ Vantaggi:

- **Efficienza:** Permette la gestione e il recupero rapido dei dati.
- **Flessibilità:** Disponibile in diverse forme (dinamico, statico, indicizzato, non indicizzato).
- **Struttura modulare:** Facilita la programmazione e il riutilizzo del codice.

✗ Svantaggi:

- **Uso errato della memoria:** Alcuni container consumano più memoria di quanto necessario.
 - **Tempo di accesso variabile:** Strutture diverse hanno tempi di accesso differenti (es. liste vs array).
-

💡 Conclusione

I container sono strumenti fondamentali in vari ambiti dell'informatica. La loro efficacia dipende dal contesto d'uso e dalla corretta implementazione. 🚀

Docker → crea e gestisce i container → container technologies

Docker → puoi usare a riga di comando oppure dall'applicazione web

Docker → come funzionamento sotto c'è Linux

Docker → tecnologia che serve per sviluppare un'applicazione → solo quello che serve → con tutte le dipendenze necessarie

Docker → unità organizzativa per scambiare molte funzionalità in poco spazio si sposta i dati rimangono uguali → spostato dati rimane tutto uguale

✓ Vantaggi:

- Più facile spostare i dati da un dispositivo ad un altro e anche molto più veloce
- diversi elementi di produzione si adatta molto facilmente

- Condividere ambiente di sviluppo con tutte le persone con le caratteristiche che ti servono
→ più container collegati tra di loro

Framework → insieme di dati che servono per collegarsi alle sue opzioni corrette con le varie funzioni

Definizione di Framework

Un **framework** è un insieme di strumenti, librerie e regole che fornisce una struttura predefinita per lo sviluppo di software. Aiuta i programmatori a scrivere codice in modo più efficiente, evitando di dover costruire tutto da zero.

✓ Vantaggi di un Framework

- **Maggiore produttività:** Fornisce componenti pronti all'uso.
- **Struttura organizzata:** Impone regole e schemi per scrivere codice più ordinato.
- **Sicurezza:** Include meccanismi di protezione contro vulnerabilità comuni.
- **Manutenibilità:** Favorisce il riutilizzo del codice e facilita gli aggiornamenti.
- **Comunità e supporto:** Spesso ha una vasta documentazione e una community attiva.

✗ Svantaggi di un Framework

- **Curva di apprendimento:** Alcuni framework sono complessi e richiedono tempo per essere imparati.
- **Rigidità:** Impone una struttura che può limitare la libertà dello sviluppatore.
- **Overhead:** Può introdurre codice e funzionalità non necessarie, rallentando le prestazioni.

Esempi di Framework

- **Web Development:** React, Angular, Vue.js
- **Backend:** Django (Python), Spring (Java), Express.js (Node.js)
- **Mobile:** Flutter, React Native
- **Machine Learning:** TensorFlow, PyTorch

💡 **Conclusione:** Un framework semplifica lo sviluppo, ma richiede studio per sfruttarne al meglio le potenzialità! 🚀

Virtual Machine (VM) vs Docker (Containerization)

Virtual Machine (VM)

Una **macchina virtuale** (VM) è un ambiente emulato che simula un sistema operativo completo. Utilizza un hypervisor (es. VMware, VirtualBox, Hyper-V, KVM) per eseguire più sistemi operativi su una singola macchina fisica.

Vantaggi delle VM

- ✓ **Isolamento completo** – Ogni VM esegue un intero sistema operativo, garantendo un forte isolamento tra le applicazioni.
- ✓ **Compatibilità** – Supporta qualsiasi sistema operativo, indipendentemente dall'host.
- ✓ **Sicurezza** – L'isolamento tra VM rende difficile la compromissione di una VM da parte di un'altra.
- ✓ **Affidabilità** – Adatto per eseguire applicazioni legacy o software che richiedono una configurazione complessa.

Svantaggi delle VM

- ✗ **Consumo di risorse** – Ogni VM esegue un intero sistema operativo, occupando molta RAM, CPU e spazio su disco.
 - ✗ **Lentezza nell'avvio** – Avviare una VM richiede tempo perché bisogna caricare un intero OS.
 - ✗ **Meno portabilità** – Spostare e ridistribuire VM è più pesante rispetto ai container.
-

Docker (Containerization)

Docker è una piattaforma di **containerizzazione** che permette di eseguire applicazioni in ambienti isolati (container) senza avviare un intero sistema operativo.

Vantaggi di Docker

- ✓ **Leggero** – I container condividono il kernel dell'OS host, riducendo il consumo di risorse.
- ✓ **Avvio rapido** – I container si avviano in pochi secondi.
- ✓ **Portabilità** – I container sono indipendenti dalla piattaforma e funzionano su qualsiasi macchina con Docker.
- ✓ **Scalabilità** – Permette di eseguire e gestire molteplici istanze di un'applicazione con facilità.
- ✓ **Facile gestione** – Si integra bene con sistemi di orchestrazione come Kubernetes.

Svantaggi di Docker

- ✗ **Isolamento limitato** – I container condividono il kernel dell'host, quindi possono avere problemi di sicurezza rispetto alle VM.
- ✗ **Compatibilità limitata** – Funziona meglio su Linux; su Windows richiede workaround o WSL2.
- ✗ **Meno adatto per applicazioni complesse** – Non è ideale per applicazioni che necessitano di un intero sistema operativo o dipendenze molto specifiche.

Quando scegliere VM o Docker?

Scenario	VM	Docker
Eseguire software con dipendenze OS-specifiche	✓	✗
Massima sicurezza e isolamento	✓	✗
Avvio rapido e leggerezza	✗	✓
Deploy di microservizi	✗	✓
Applicazioni legacy	✓	✗
Scalabilità orizzontale (Kubernetes, cloud)	✗	✓

Docker → aiuta a gestire e creare strutture di container

Linux → pronto a gestire container

Configurazione Docker Desktop

Esegui i Seguenti comandi nel cmd come amministratore

Riattivazione hypervisor → `bcdedit /set hypervisorlaunchtype auto`

Esegui i Seguenti comandi nel powershell come amministratore

`enable-windowsoptionalfeatures -online -featurename microsoft-hyper-v -all` → aggiunge caratteristiche Hyper Visor che servono per il container rispondi y

`enable-windowsoptionalfeature -online -featurename container -all` → aggiunge i requisiti per docker rispondi y

Esegui i Seguenti comandi nel cmd come amministratore

`wsl --update` → aggiorni wsl

Crei account di docker da docker desktop installando l'applicazione docker desktop dal file .exe

Riavi PC → e avvi docker desktop

Username mio → nicolamarano

per vedere se funziona vai su cmd → scrivi docker e vedi le varie configurazioni possibili

```
top          Display the running processes of a container
unpause      Unpause all processes within one or more containers
update       Update configuration of one or more containers
wait         Block until one or more containers stop, then print their exit codes

Global Options:
  --config string      Location of client config files (default
                        "C:\\Users\\Marano Nicola\\.docker")
  -c, --context string  Name of the context to use to connect to the
                        daemon (overrides DOCKER_HOST env var and
                        default context set with "docker context use")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket to connect to
  -l, --log-level string Set the logging level ("debug", "info",
                        "warn", "error", "fatal") (default "info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string    Trust certs signed only by this CA (default
                        "C:\\Users\\Marano Nicola\\.docker\\ca.pem")
  --tlscert string      Path to TLS certificate file (default
                        "C:\\Users\\Marano Nicola\\.docker\\cert.pem")
  --tlskey string       Path to TLS key file (default
                        "C:\\Users\\Marano Nicola\\.docker\\key.pem")
  --tlsverify           Use TLS and verify the remote
  -v, --version         Print version information and quit

Run 'docker COMMAND --help' for more information on a command.

For more help on how to use Docker, head to https://docs.docker.com/go/guides/

C:\\Users\\Marano Nicola>|
```

metti installazione nelle impostazioni → start docker when you sign in your computer

Creazione Primo Container di una pagina web in rete

scarichi estensioni in visual studio code → scaricando estensioni docker extension pack e prettier e docker

Container → basati su immagine

Apri terminale in visual studio code → con ctrl + maiusc + ò

docker build . → crea immagine sul container nel docker → dove appoggia container interpretando le istruzioni del docker file

docker images → vedi immagini presenti nel disco

docker run -p porta:porta nome immagine → avvi docker l'immagine sta girando

fai nel browser cerci 127.0.0.1 o localhost:porta → e vedi il sito funzionare

docker ps → vedo container che stanno funzionando in questo momento → quando tempo è creato e da quanto gira → su che porta funziona → con un nome casuale

docker stop nome container → bloccata il container

docker gira cancella immagine → non gira più il container

1 immagine → anche + container

su docker hub vedi immagine già presenti che tu puoi usare

docker run servizio → crea un container con il servizio che mi interessa

docker run node → crei container creando il container con quel servizio → fa girare container in base a quello che scarichi

docker ps -a → vedi docker attivi e non attivi (tutti)

docker run -it node → esegue node

11 Febbraio 2025

Puoi anche eliminare immagini già create e anche i container

docker -rmi prune → elimini tutte le immagini create

docker run nome servizio → creo servizio basato sul servizio che ti serve e dopo la scarica lui in automatico e crea il container

docker images → vedi immagine

docker stop nome container → blocca il container

docker rm nome o id container → elimina container

docker build . → esegui file dockerfile e crea immagine

docker run id → esegui il container

docker rm nome container o id → elimini container

docker run -p porta destinazione iniziale:porta destinazione finale nome container o id

Immagini in sola lettura → i container fanno funzionare il sito

Docker -help → aiuto ai comandi docker

Docker ps -help → tutti opzioni dopo ps

docker start nome id o nome container → fa ripartire container

Attack mode → blocco terminale → con il container

deattack mode → terminale non bloccato solo se web

Docker run -it nome id o nome dell'immagine di riferimento → esegui in modo iterativo il software di riferimento

docker container prune → elimino tutti i container

Docker image inspect nome immagine → tutti i dati dell'immagine

12 Febbraio 2025

Volumi collegati diversi container tra di loro

Tipi di Volumi in Docker e Loro Descrizione

1. Volumes (Volume Docker)

- **Nome corretto:** volumes
- **Descrizione:**

I volumi sono la modalità più consigliata per gestire la persistenza dei dati in Docker. Sono gestiti direttamente da Docker e salvati nel percorso /var/lib/docker/volumes/.

- Permettono di condividere dati tra più container.
- Sono più sicuri rispetto ai bind mounts perché Docker li gestisce direttamente.
- Possono essere usati su diversi host con strumenti come Docker Swarm.

Esempio di creazione e utilizzo:


```
docker volume create mio_volume
docker run -d --name container1 -v mio_volume:/app/data ubuntu
```

Ora mio_volume è disponibile e collegabile ad altri container.

2. Bind Mounts

- **Nome corretto:** bind mounts
- **Descrizione:**

I bind mounts collegano direttamente una cartella del filesystem host a una cartella del container.

- Consentono di accedere ai file dell'host in tempo reale.
- Offrono maggiore flessibilità, ma sono meno sicuri perché permettono al container di modificare direttamente i file dell'host.
- Utilizzati spesso per lo sviluppo, poiché i cambiamenti sul filesystem host si riflettono immediatamente nel container.

Esempio di utilizzo:

```
docker run -d --name container1 -v /home/utente/data:/app/data ubuntu
```

Ora i file nella directory /home/utente/data dell'host saranno visibili nel container sotto /app/data.

3. Tmpfs Mounts

- **Nome corretto:** tmpfs mounts
- **Descrizione:**

I tmpfs mounts creano un filesystem temporaneo in RAM.

- Ideali per dati sensibili che non devono essere salvati su disco.
- I dati vengono eliminati quando il container viene arrestato.
- Utilizzati spesso per file di cache o dati di sessione.

Esempio di utilizzo:

```
docker run -d --name container1 --tmpfs /app/data:rw,size=100M ubuntu
```

Qui /app/data sarà un'area di memoria temporanea di 100MB.

4. Named Volumes

- **Nome corretto:** named volumes

- **Descrizione:**

I named volumes sono volumi Docker con un nome specifico.

- Sono più facili da gestire rispetto ai volumi anonimi.
- Possono essere condivisi tra più container.
- Utili per archiviazione persistente tra più istanze dello stesso servizio.

Esempio di creazione e utilizzo:

```
docker volume create dati_persistenti
```

```
docker run -d --name container1 -v dati_persistenti:/app/data ubuntu
```

Il volume dati_persistenti sarà disponibile per più container.

5. Anonymous Volumes

- **Nome corretto:** anonymous volumes

- **Descrizione:**

Gli anonymous volumes sono volumi Docker senza nome esplicito.

- Vengono creati automaticamente da Docker quando un container viene avviato con `-v /path/in/container`.
- Sono utili per evitare problemi di permessi su filesystem host.
- Non possono essere facilmente riutilizzati in altri container.

Esempio di utilizzo:

```
docker run -d --name container1 -v /app/data ubuntu
```

Qui Docker creerà un volume anonimo e lo collegherà a `/app/data`.

6. Read-Only Volumes

- **Nome corretto:** read-only volumes

- **Descrizione:**

Sono volumi montati in modalità sola lettura.

- Utilizzati per proteggere dati da modifiche accidentali.
- Possono essere applicati sia ai volumes che ai bind mounts.

Esempio di utilizzo:

```
docker run -d --name container1 -v /home/utente/data:/app/data:ro ubuntu
```

Qui `/app/data` sarà accessibile solo in lettura nel container.

Conclusione

Docker offre diverse modalità di gestione dei volumi a seconda delle esigenze:

- **volumes** → Persistenza sicura dei dati.
- **bind mounts** → Accesso diretto ai file dell'host.
- **tmpfs mounts** → Memoria temporanea in RAM.
- **named volumes** → Volumi con nome specifico, più gestibili.
- **anonymous volumes** → Volumi temporanei, senza nome esplicito.
- **read-only volumes** → Protezione dei dati con accesso in sola lettura.

Docker container nome id1 o nome container1 secondo → ne elimino di più

non si può cancellare l'immagine se ci sono i container collegati → prima bisogna cancellare i container

Docker run -p porta:porta -d --rm nome id o container id → eseguo su detach mode sulla porta preimpostata il file del docker corretto → stoppo si cancella (--rm)

-d → detach mode

docker image inspect nome id → esamina immagine appena creata

Inviare/Togliere file in container in movimento

docker cp percorso file nome id o container id:/nome file copiato nella route del pc con il percorso → copia i dati nel percorso specificato nel container

docker cp nome container o id container:/percorso file cartella di destinazione → metto file nel percorso corretto nel parte locale

→ copiare dati non è buona esecuzione mentre i file sono in esecuzione

--name nome → imposto il nome del container nel comando docker run

docker run -p porta:porta -d --rm --name nome → faccio un container con il nome e con la porta che voglio detach mode e si elimina subito quando lo blocco

anche le immagine possono avere un nome → : → progetto specifico dentro quel lavoro → id univoco per l'immagine

`docker build -t nome percorso:specifico mio .` → creo immagine con il nome

`docker image prune -a` → elimina tutte le immagini

`docker run -p porta:porta -d --rm --name nome container nome immagine` → creo container con il nome e il tag

Pubblicare immagine in Docker Hub

tutto la possono scaricare → o docker hub o registro privato

Docker Hub → sito web → Create a Repository

metti autore → nome app e descrizione → metti pubblica

Create

`docker build -t nome docker/nome container .`

Docker images

`docker login` → entro dentro docker

`docker push nome docker/nome applicazione:ultima tag`

nicolamarano/applicazione

Last pushed less than a minute ago • Repository size: 391 MB

Contiene la nostra app GoalsAPP

Add a category

General

Tags

Builds

Collaborators

Webhooks

Settings

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	less than 1 day	less than a minute

See all

vedi che è andata nella container di docker hub corretto

metto solo quelle che non conosce → risparmio servizio e spazio

docker logout → esci da docker

docker pull nome docker/nome applicazione → prendi applicazione da docher hub

docker image

Voluni → rendere permanenti i dati

Fixed → non si possono cambiare quando sono al suo interno

Dati memorizzati in container o db → che non perdo i dati → fermo o riavvio i dati (permanent)

dati → salvati in volumi (non li perdi)

13 Febbraio 2025

evitare cancellazione dati → servono i volumi

3 tipi:

- Anonymus → cancello container perdo i dati → elimino container, elimino anche il volume e quindi anche i dati
- Named → cancello container mantengo i dati → ricreo container e mi collego i dati a quello
- PaildMount → scrivere direttamente dentro il container

Dati app → i diversi codici sorgenti e temporanei e permanenti

Volume → cartella nel pc → che sta dentro nel pc → mappata nel container

Cartella da container a fuori

Volumi → rimangono anche se container spento e non perdo i dati

PaildMount → gestiti da noi → risolvono problemi specifici

Named → persistenza dei dati

Volumi anonimi →

`docker run -p 3000:80 -d --rm --name feedback-app -v feedback:/app/feedback feedback-node:volume` → fa run container con volumi specificando la cartella

docker volume ls → vedi volumi creati

docker ps

PaidMount → ottimale per leggere e scrivere i dati miei

docker run -d -p porta:porta --name nome -v cartella mappata nel container -v perocrsp file
cartella pc oppure "%cd" -v cartella /app

docker run -d -p 3000:80 --name feedback-app -v feedback:/app/feedback -v "\${PWD}:/app"
feedback-node:volume

Anonymus → non condiviso con altri container → non riutilizzato anche sulla stessa immagine → non riassociare

Named → generale → condiviso e riutilizzato e anche da altri

PaidMount → tutto

Volumi in sola lettura

17 Febbraio 2025

su cmd

mongosh → vai in mongodb

show dbs → vedi i db creati

use database prova → crea db e lo usi

veloce nei dati non strutturati → stampa json

db.nome file → crea tabella

db.utenti.insertOne({elemento:"valorei"}) → inserisci elementi nel db

i dati vengono salvati in una collection di dati

19 Febbraio 2025

MongoDB Compass è l'interfaccia grafica ufficiale di MongoDB, uno dei database NoSQL più popolari. Fornisce un modo facile e visivo per interagire con MongoDB, consentendo agli sviluppatori e agli amministratori di database di gestire, interrogare e visualizzare i dati senza dover utilizzare la riga di comando. È particolarmente utile per chi è nuovo in MongoDB o preferisce un'interfaccia visiva per interagire con il database.

Le principali funzionalità di MongoDB Compass includono:

1. **Interfaccia Intuitiva:** Compass offre un'interfaccia facile da navigare che aiuta gli utenti a esplorare e gestire collezioni, documenti e indici.
2. **Costruttore di Query:** Consente di costruire query MongoDB in modo visivo, utile per chi non è esperto nel linguaggio di query di MongoDB.
3. **Esplorazione Dati:** Compass permette di esplorare i dati in modo visivo, mostrando le collezioni e i documenti in un formato strutturato. È possibile filtrare e ordinare i dati facilmente.
4. **Visualizzazione dello Schema:** Compass analizza automaticamente e visualizza lo schema delle tue collezioni, aiutandoti a comprendere meglio la struttura dei dati e a prendere decisioni informate sulla modellazione e sugli indici.
5. **Gestione degli Indici:** Puoi creare, eliminare e gestire gli indici direttamente dall'interfaccia di Compass, rendendo più facile ottimizzare le prestazioni delle query.
6. **Costruttore di Pipeline di Aggregazione:** Compass supporta la creazione di pipeline di aggregazione, una funzione potente di MongoDB per trasformare e analizzare i dati in modo complesso.
7. **Ottimizzazione delle Prestazioni:** Compass fornisce informazioni sulle prestazioni delle query e degli indici, aiutandoti a ottimizzare l'istanza di MongoDB.

db.utenti.insertMany([{"elementi"},{elemento2}] → inserisci elementi nel db → più elementi

comandi find → cerci utenti

db.utenti.find() → *cerca gli utenti*

db.utenti.findOne({campo:"valore"}) → cerca un elemento con questi dati

db.utenti.find({campo:"valore"}) → cerca tutti elementi con quei dati

db.utenti.find({eta:{\$gt:21}}) → gt maggiori di valore che imponi

db.utenti.find({eta:{\$in:[21,55]}}) → trovami quegli utenti che come età stanno in quell'insieme (IN)

Ecco gli operatori di confronto più comuni in MongoDB:

1. **\$gt (greater than)** → Maggiore di

```
js
CopiaModifica
db.utenti.find({ eta: { $gt: 21 } })
```

Trova tutti gli utenti con età maggiore di 21.

2. **\$gte (greater than or equal)** → Maggiore o uguale a

```
js
CopiaModifica
db.utenti.find({ eta: { $gte: 21 } })
```

Trova tutti gli utenti con età maggiore o uguale a 21.

3. **\$lt (less than)** → Minore di

```
js
CopiaModifica
db.utenti.find({ eta: { $lt: 21 } })
```

Trova tutti gli utenti con età minore di 21.

4. **\$lte (less than or equal)** → Minore o uguale a

```
js
CopiaModifica
db.utenti.find({ eta: { $lte: 21 } })
```

Trova tutti gli utenti con età minore o uguale a 21.

5. **\$eq (equal)** → Uguale a

```
js
CopiaModifica
db.utenti.find({ eta: { $eq: 21 } })
```

Trova tutti gli utenti con età esattamente uguale a 21.

6. **\$ne (not equal)** → Diverso da

```
js
CopiaModifica
db.utenti.find({ eta: { $ne: 21 } })
```

Trova tutti gli utenti con età diversa da 21.

7. **\$in (inside array)** → Contenuto in un insieme

```
js
CopiaModifica
db.utenti.find({ eta: { $in: [21, 55] } })
```

Trova tutti gli utenti con età pari a 21 o 55.

8. **\$nin (not in array)** → Non contenuto in un insieme

```
js
CopiaModifica
db.utenti.find({ eta: { $nin: [21, 55] } })
```

Trova tutti gli utenti con età diversa da 21 e 55.

Questi operatori possono essere combinati con altri operatori logici (\$and, \$or, \$not, \$nor) per ricerche più avanzate!

```
db.utenti.find({
  $and: [
    { città: "Vicenza" },
    { età: { $gt: 21 } }
  ]
})
```

Descrizione:

Questa query cerca tutti gli utenti che soddisfano entrambe le condizioni:

- ✓ **Vivono a Vicenza** (città: "Vicenza")
- ✓ **Hanno un'età maggiore di 21 anni** (età: { \$gt: 21 })

In alternativa, MongoDB permette di scrivere la stessa query in modo più semplice (poiché l'operatore \$and è implicito quando ci sono più condizioni nello stesso oggetto):

```
db.utenti.updateOne(
  { nome: "Daniele" }, // Condizione per trovare l'utente
  { $set: { età: 29, cognome: "Gobbi" } } // Aggiornamento
)
```

cambia età e cognome della prima persona che si chiama Daniele

```
db.utenti.updateMany(
  { città: "Pordenone" }, // Filtra gli utenti che vivono a Pordenone
  { $set: { cap: 33180 } } // Imposta il CAP a 33180
)
```

→ aggiunge campo se non c'è altrimenti modifica a tutti quelli di Pordenone

`${inc}` : condizione → incrementa di 1

`db.utenti.deleteOne({nome:"Giovanna"})` → elimina utente

`docker run -d --name mongodb mongo` → sia container che immagine

20 Febbraio 2025

Stessa rete → più container comunicano senza mettere rete

`--network nome rete` → imposto la rete

`docker run -d --name mongodb --network favorites-net mongo`

`docker network create favorites-net` → crea rete

`docker network ls` → vedi lista reti

`docker run -d --rm -p 3000:3000 --name favorites --network favorites-net favorite-node` → crea rete

25 Febbraio 2025

`docker run --name mongodb -v data:/data/db --rm -d --network goals-net mongo`

docker run → Avvia un nuovo container.

- **--name mongodb** → Assegna al container il nome mongodb, rendendo più facile la gestione (es. `docker stop mongodb`).
- **-v data:/data/db** → Monta un volume chiamato data nella directory /data/db del container. Questo garantisce che i dati del database siano persistenti anche se il container viene rimosso.
- **--rm** → Rimuove automaticamente il container una volta che viene arrestato (senza questa opzione, il container rimarrebbe nella lista dei container terminati).
- **-d** → Esegue il container in background (modalità "detached").

- **--network goals-net** → Collega il container a una rete Docker chiamata goals-net, permettendogli di comunicare con altri container nella stessa rete.
- **mongo** → È l'immagine ufficiale di MongoDB che viene scaricata (se non già presente) ed eseguita.

Questo comando avvia un'istanza di MongoDB in un container Docker con un volume per la persistenza dei dati e la connessione a una rete Docker personalizzata. Il container verrà eliminato automaticamente quando verrà fermato.

```
docker run --name mongodb -v data:/data/db --rm -d --network goals-net -e
MONGO_INITDB_ROOT_USERNAME=its -e MONGO_INITDB_ROOT_PASSWORD=Vmware1! mongo
```

docker run → Avvia un nuovo container.

- **--name mongodb** → Assegna il nome mongodb al container, facilitando la gestione (ad esempio `docker stop mongodb`).
 - **-v data:/data/db** → Monta un volume chiamato data sulla directory /data/db all'interno del container.
 - Questo garantisce che i dati del database siano persistenti, anche se il container viene eliminato.
 - **--rm** → Il container verrà automaticamente rimosso quando si interrompe (senza questa opzione, il container rimarrebbe tra i container spenti).
 - **-d** → Esegue il container in modalità "detached" (in background, senza mostrare i log nel terminale).
 - **--network goals-net** → Collega il container alla rete Docker goals-net, permettendogli di comunicare con altri container nella stessa rete.
 - **-e MONGO_INITDB_ROOT_USERNAME=its** → Imposta la variabile d'ambiente MONGO_INITDB_ROOT_USERNAME, definendo l'utente amministratore del database (its).
 - **-e MONGO_INITDB_ROOT_PASSWORD=Vmware1!** → Imposta la variabile d'ambiente MONGO_INITDB_ROOT_PASSWORD, definendo la password dell'utente root (Vmware1!).
 - **mongo** → Indica l'immagine Docker di MongoDB da scaricare ed eseguire.
-
- **Avvia un'istanza MongoDB** in un container Docker.
 - **Crea un volume** per salvare i dati in /data/db, evitando di perderli se il container viene riavviato.
 - **Si collega alla rete goals-net**, permettendo l'accesso da altri container (ad esempio, un'applicazione che usa MongoDB).
 - **Configura l'utente amministratore** (its) e la password (Vmware1!) per la sicurezza.

- **Viene eseguito in background** e si rimuove automaticamente alla chiusura.

```
docker run --name goals-backend -v ${PWD}:/app -v logs:/app/logs -v /app/node_modules --rm -d  
-p 81:80 --network goals-net goals-node
```

docker run → Esegue un nuovo container.

1. **--name goals-backend** → Assegna il nome goals-backend al container.
2. **-v \${PWD}:/app** → Mappa la directory corrente (\${PWD}) alla directory /app nel container.
 - Questo significa che i file della directory attuale sul tuo host saranno visibili nel container sotto /app.
3. **-v logs:/app/logs** → Crea un volume Docker chiamato logs e lo monta sulla cartella /app/logs del container.
 - Questo è utile per mantenere i file di log separati e persistenti tra più esecuzioni.
4. **-v /app/node_modules** → Monta /app/node_modules come volume anonimo.
 - Questo evita che i node_modules locali interferiscano con quelli del container.
5. **--rm** → Rimuove automaticamente il container quando viene fermato.
 - Questo evita di lasciare container inutilizzati in esecuzione.
6. **-d** → Avvia il container in modalità "detached" (in background).
7. **-p 81:80** → Mappa la porta 80 del container sulla porta 81 dell'host.
 - Se il server Node.js dentro il container gira sulla porta 80, sarà accessibile dall'host sulla porta 81.
8. **--network goals-net** → Connettere il container alla rete Docker chiamata goals-net.
 - Serve per permettere la comunicazione con altri container nella stessa rete.
9. **goals-node** → Indica l'immagine Docker da cui avviare il container.
 - Deve essere un'immagine esistente, creata in precedenza con docker build -t goals-node ..

Esegue un container basato sull'immagine goals-node.

- Monta la cartella del progetto /app, mantenendo i log separati.
- Evita problemi con node_modules montandoli in un volume separato.
- Rende il servizio accessibile sulla porta 81 dell'host.
- Connettere il container a una rete personalizzata goals-net.
- Si rimuove automaticamente quando viene arrestato.

Nodemon → faccio una modifica blocca container ribuilda e fa le modifiche in automatico nel container e nel volume dove è collegato

docker logs goals-backend → vedi i logs che sono nell'immagine in questo momento e cosa sta facendo l'applicazione in questo momento

Docker Compose → appunti dove tieni in memorizza le cose da fare tutte i vari container e i vari volumi, le varie immagini creando anche la rete corretta → ottimizza nel processo di setup → dentro anche tutte le porte dei container rete immagini