Corso Docker

Docker Compose

Francesco Sinisi (francesco.sinisi@cnaf.infn.it)

Sommario

- Introduzione
- Caso d’uso
- Configurazione
- CLI

Introduzione

Cos’è Docker Compose e come si installa

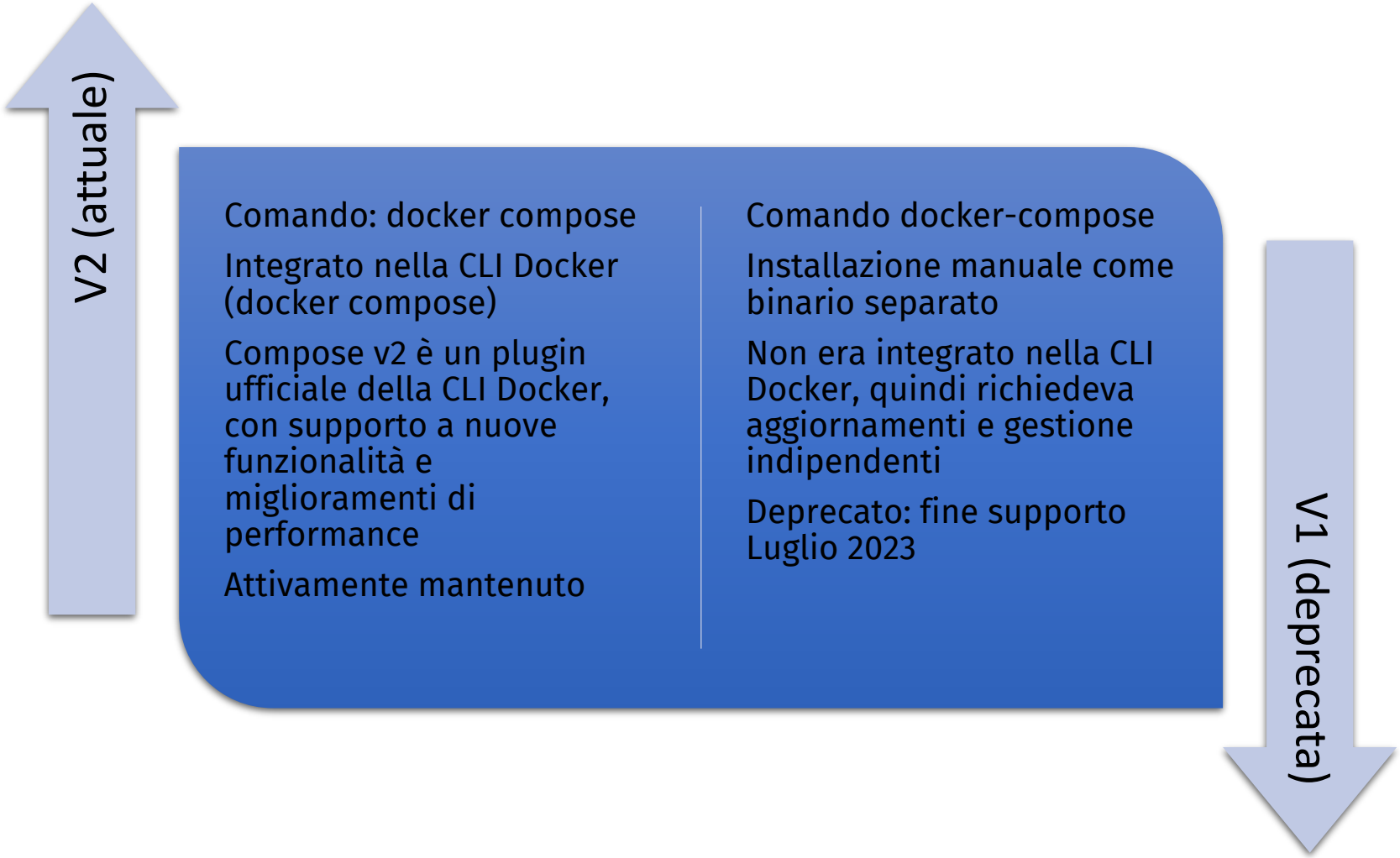
Cos’è Docker Compose

Docker Compose è uno strumento che **semplifica la gestione di applicazioni multi-container**. Invece di avviare manualmente ogni container con comandi separati, Compose consente di definire l'intera architettura in un unico file YAML.

Con Docker Compose puoi **descrivere, configurare e avviare più container con un solo comando**. È particolarmente utile quando un'applicazione richiede più servizi, come un'app web, un database e un sistema di cache.

Il cuore di Docker Compose è il file **docker-compose.yml**, dove si definiscono i servizi, le reti e i volumi necessari per far funzionare l'applicazione in modo coordinato.

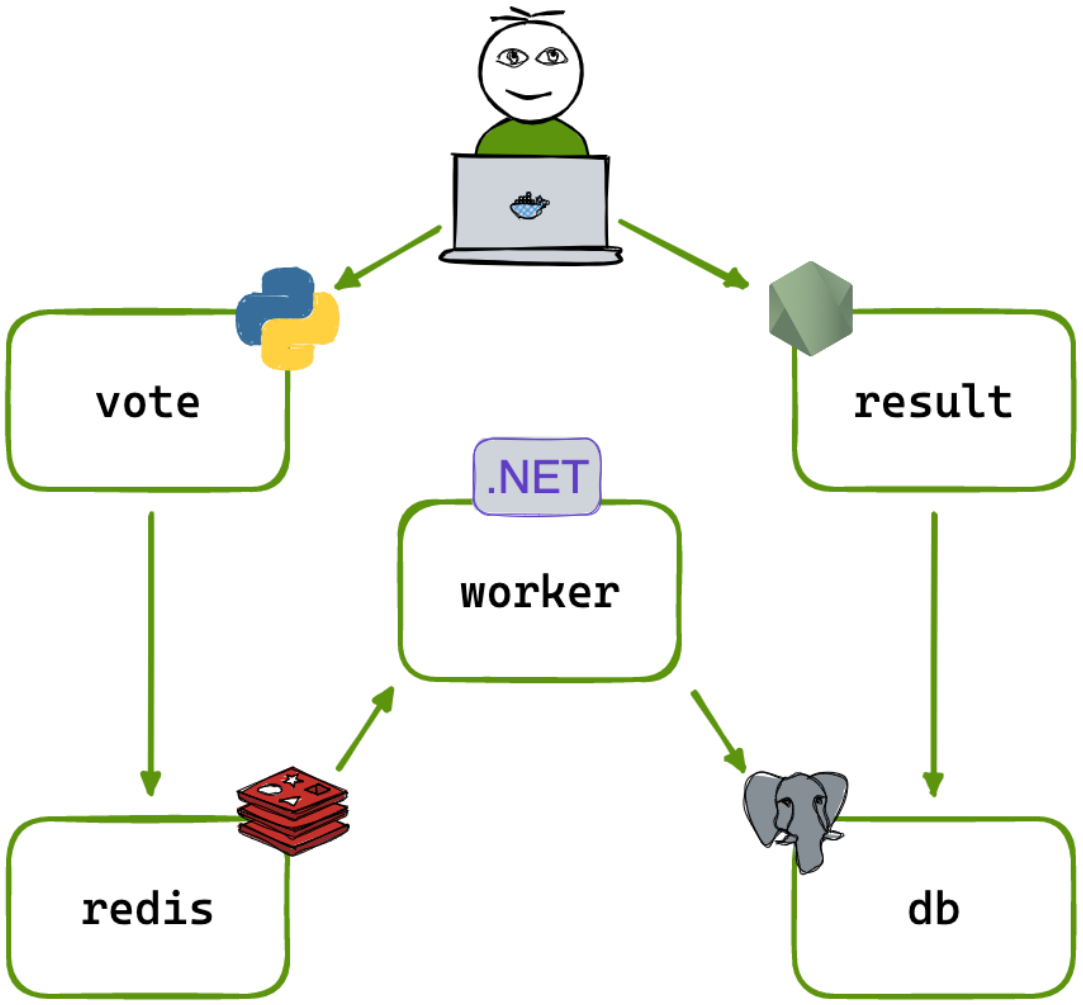
Installazione



Caso d’uso

Voting-app

Voting-app



vote — Interfaccia web per votare tra due opzioni. Espone un form semplice che invia le preferenze a Redis.

redis — Database in-memory per memorizzare temporaneamente i voti. Usato per la rapidità nella raccolta delle preferenze.

worker — Servizio backend che trasferisce i voti da Redis a PostgreSQL. Funziona come ponte asincrono tra il frontend e il database.

db — Database relazionale per conservare i risultati in modo persistente. Riceve i dati elaborati dal worker.

result — Visualizza i risultati in tempo reale. Recupera i dati dal database PostgreSQL e li mostra in una pagina web.

Lanciamo l’app con docker run
I servizi comunicano tra di loro?

vote	docker run -d --name vote -p 8080:80 dockersamples/examplevotingapp_vote
redis	docker run -d --name redis redis:alpine
worker	docker run -d --name worker dockersamples/examplevotingapp_worker
db	docker run -d --name db -e POSTGRES_USER=POSTGRES -e POSTGRES_PASSWORD=postgres postgres:15-alpine
result	docker run -d --name result -p 8081:80 dockersamples/examplevotingapp_result

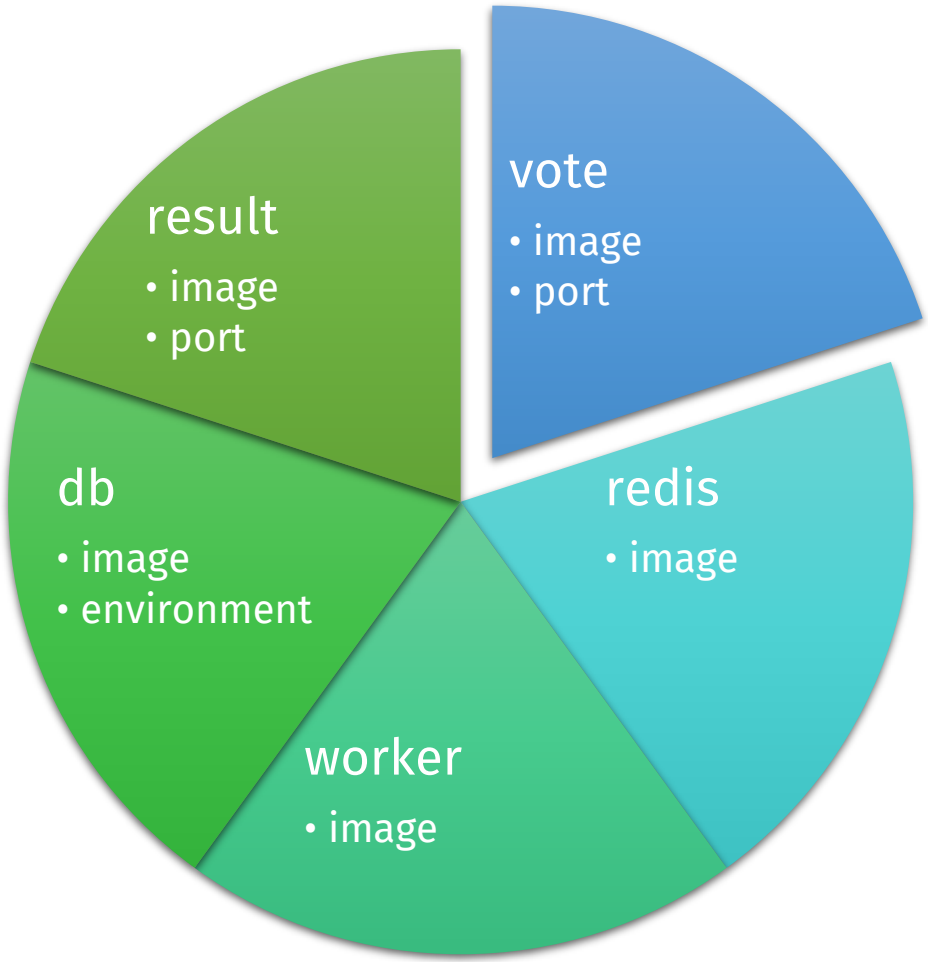
Lanciamo l’app con docker run
Creiamo e aggiungiamo una network

network	docker network create voting-net
vote	docker run -d --name vote -p 8080:80 --network voting-net dockersamples/examplevotingapp_vote
redis	docker run -d --name redis --network voting-net redis:alpine
worker	docker run -d --name worker --network voting-net dockersamples/examplevotingapp_worker
db	docker run -d --name db -e POSTGRES_USER=POSTGRES -e POSTGRES_PASSWORD=postgres --network voting-net postgres:15-alpine
result	docker run -d --name result -p 8081:80 --network voting-net dockersamples/examplevotingapp_result

Lanciamo l'app con docker run
L'ordine è importante!

db	docker run -d --name db -e POSTGRES_USER=POSTGRES -e POSTGRES_PASSWORD=postgres --network voting-net postgres:15-alpine
redis	docker run -d --name redis --network voting-net redis:alpine
vote	docker run -d --name vote -p 8080:80 --network voting-net dockersamples/examplevotingapp_vote
result	docker run -d --name result -p 8081:80 --network voting-net dockersamples/examplevotingapp_result
worker	docker run -d --name worker --network voting-net dockersamples/examplevotingapp_worker

Traduciamo in docker compose

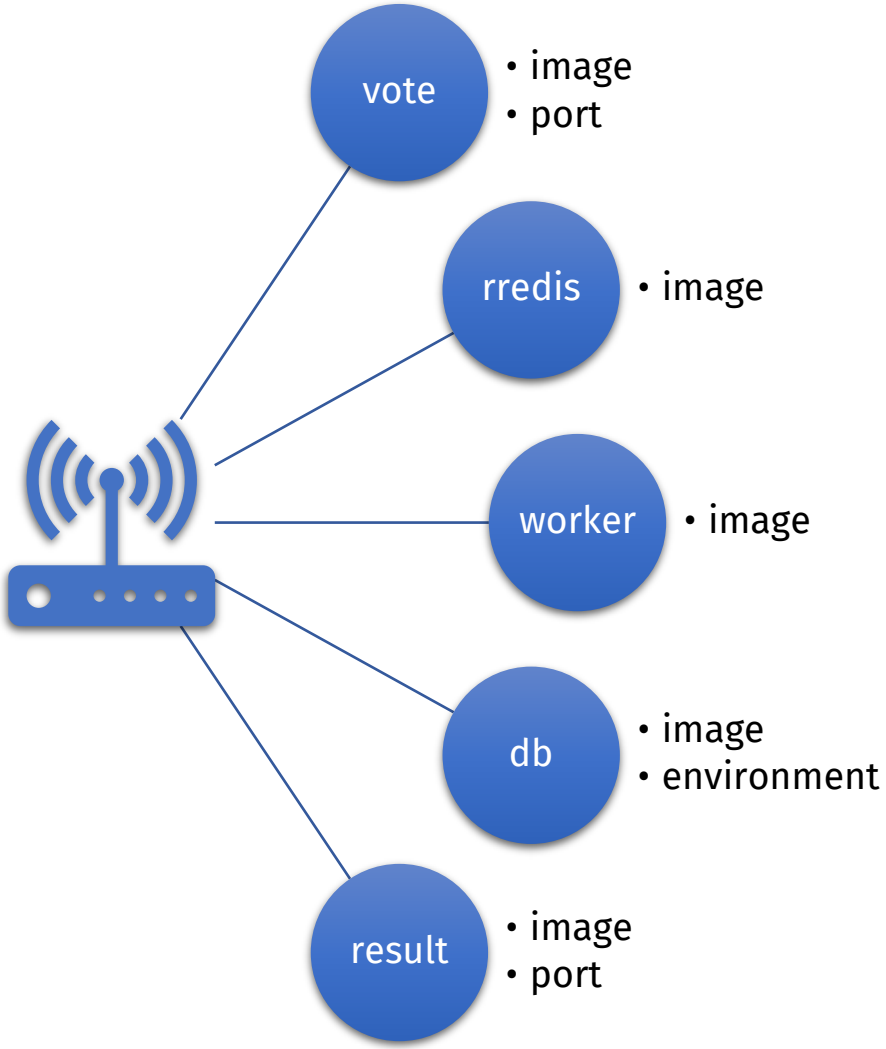



```yaml
services:
  vote:
    image: dockersamples/examplevotingapp_vote
    ports:
      - "8080:80"
  result:
    image: dockersamples/examplevotingapp_result
    ports:
      - "8081:80"
  worker:
    image: dockersamples/examplevotingapp_worker
  redis:
    image: redis:alpine
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
```

services

I container che compongono l'applicazione

Aggiungiamo la network



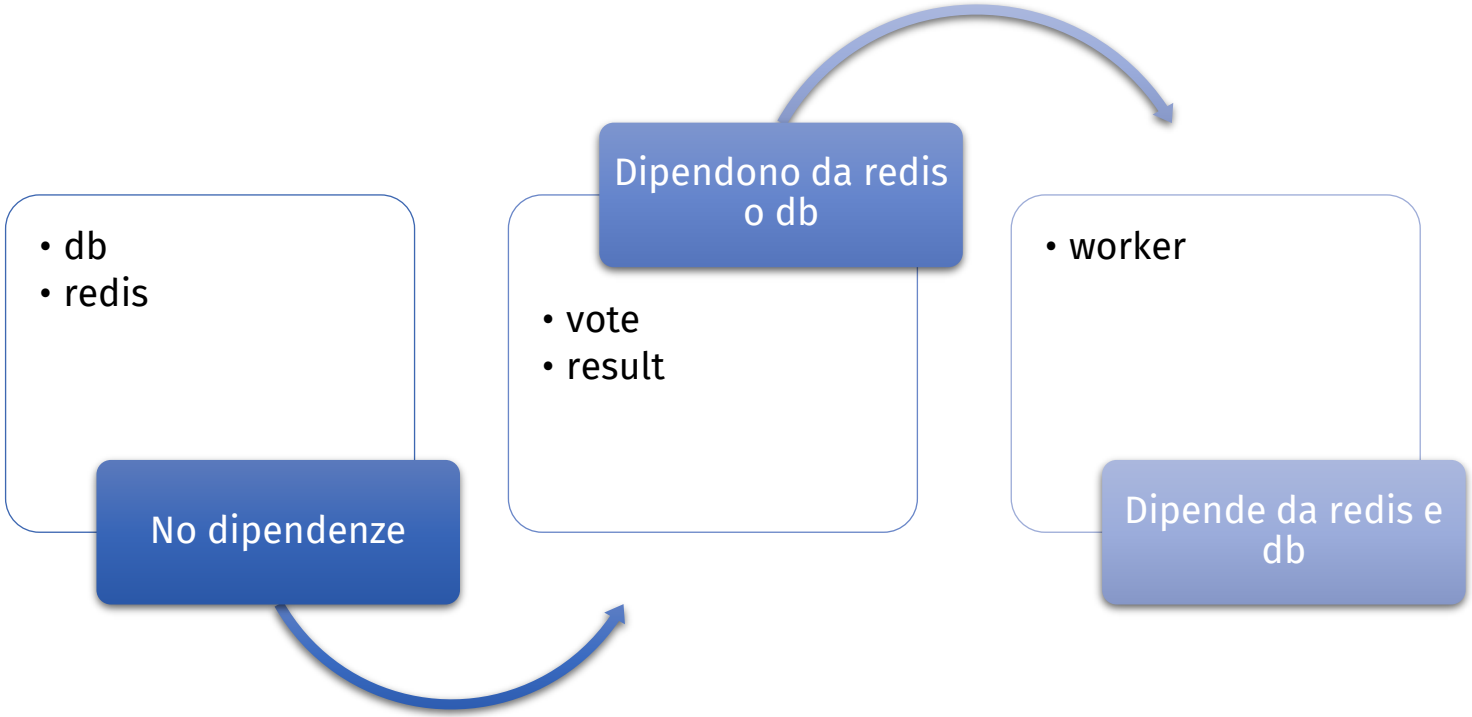

```yaml
services:
  vote:
    networks:
      - voting-net
  result:
    networks:
      - voting-net
  worker:
    networks:
      - voting-net
  redis:
    networks:
      - voting-net
  db:
    networks:
      - voting-net

networks:
  voting-net:
```

networks

Le reti virtuali che connettono i servizi

Creiamo le dipendenze

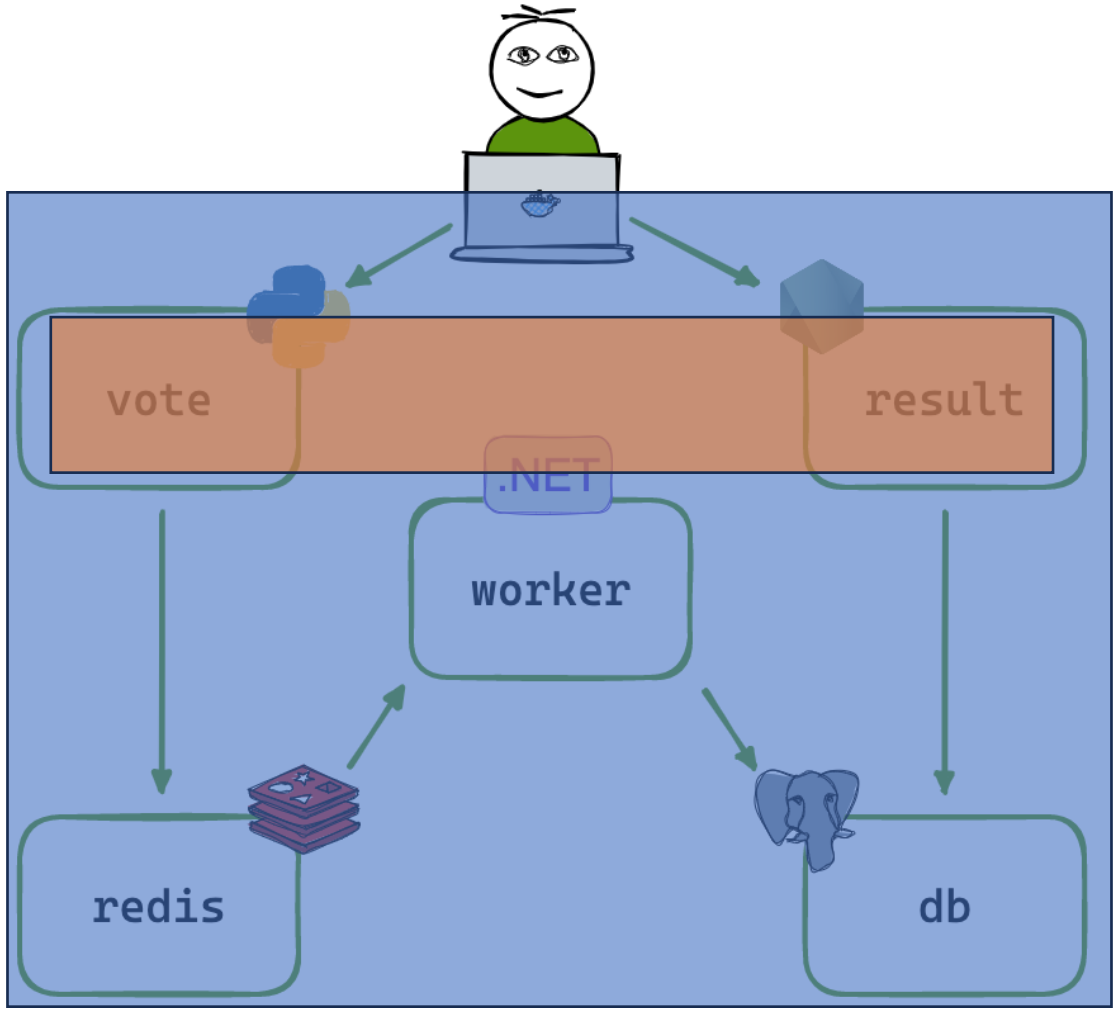


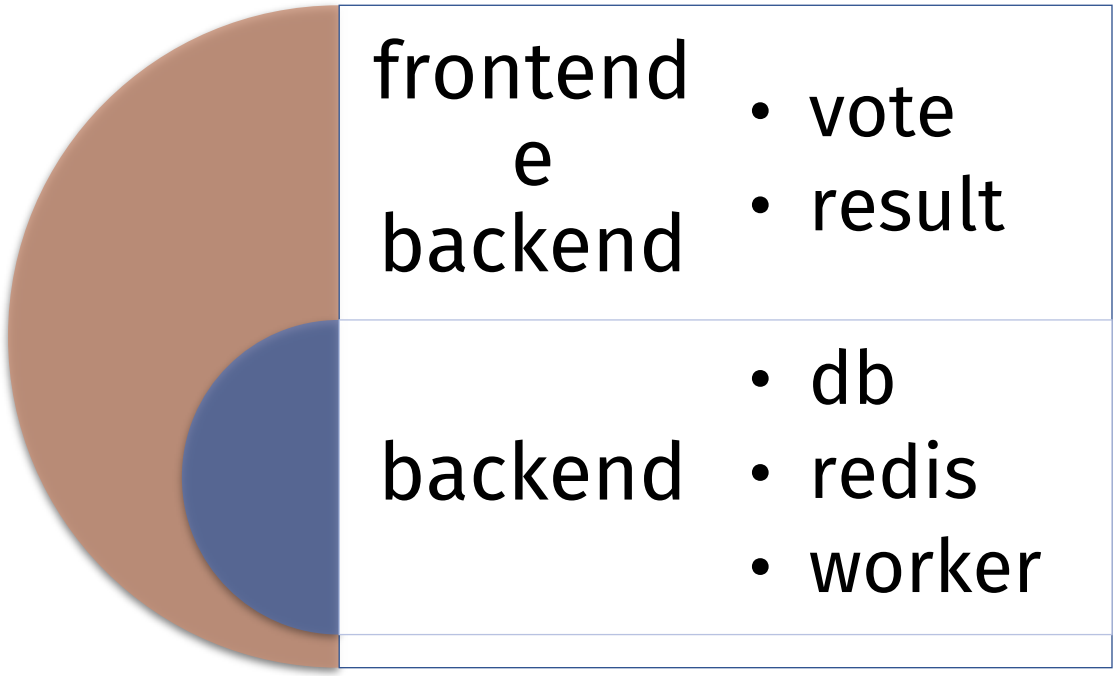

```yaml
services:
  vote:
    depends_on:
      - redis
  result:
    depends_on:
      - db
  worker:
    depends_on:
      - redis
      - db
  redis:
  db:
```

depends_on

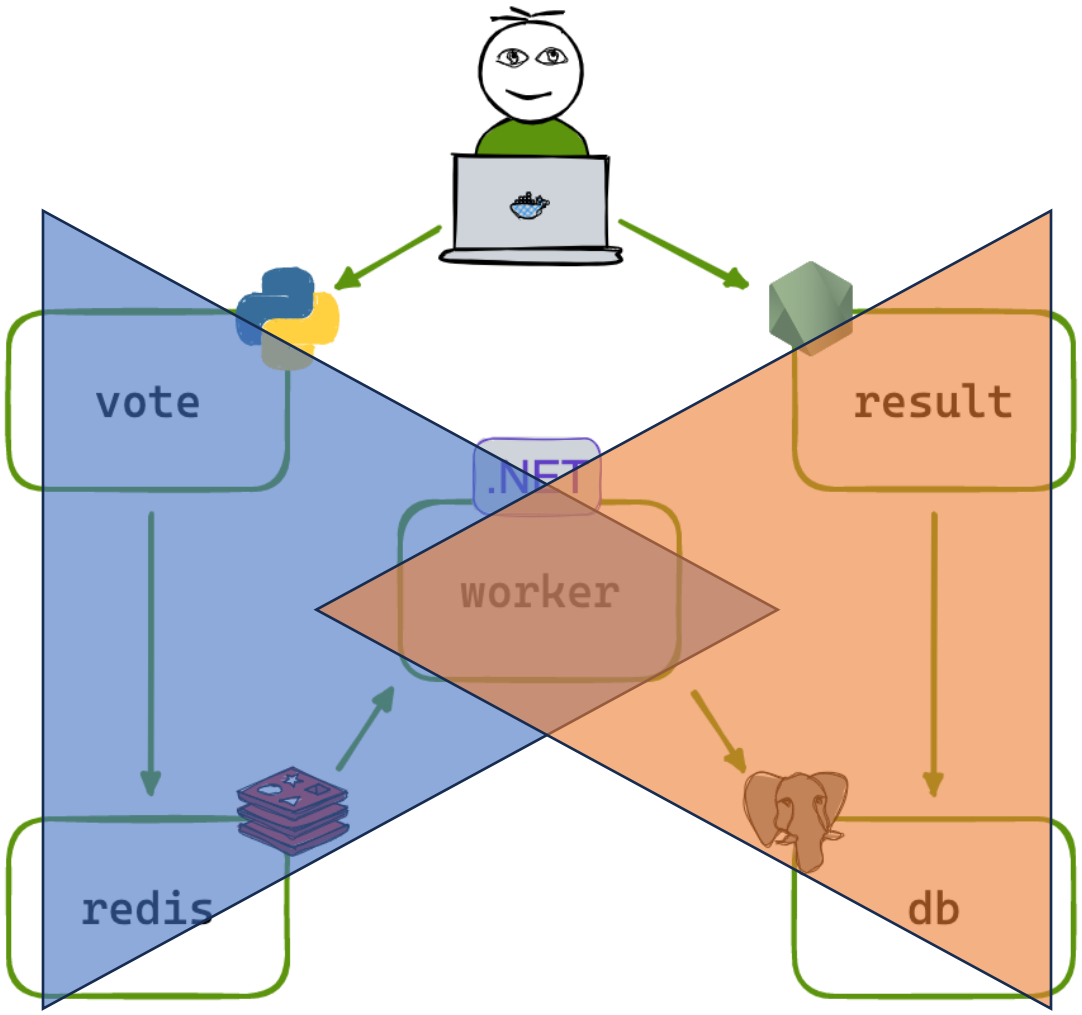
Specifica che un servizio dipende da un altro

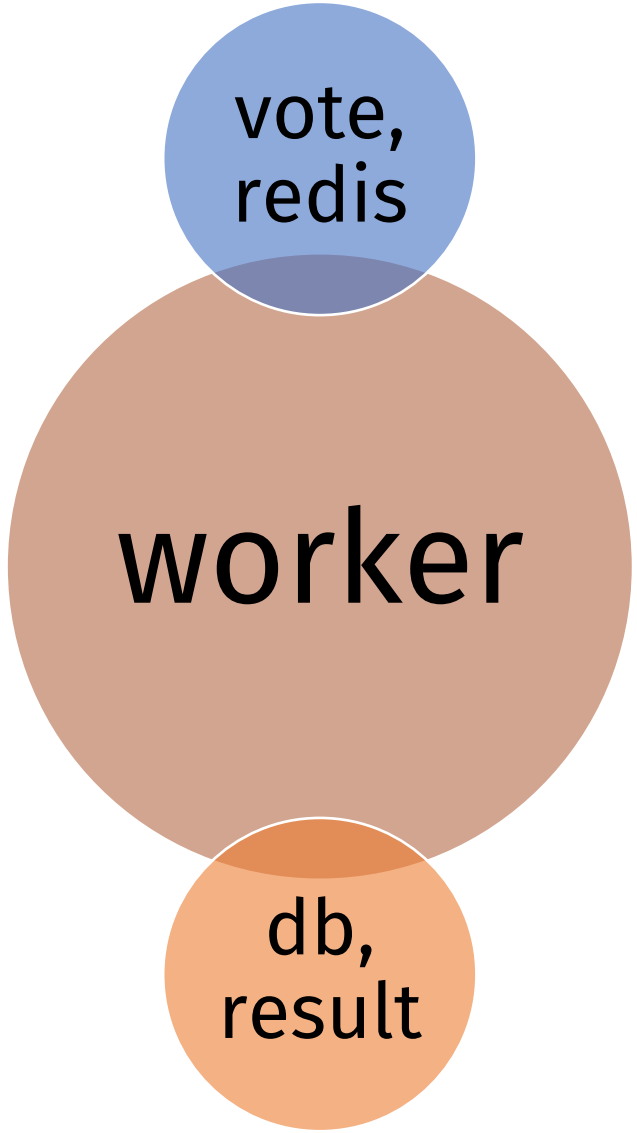
Network segregation





Network segregation





Volumes e healthcheck

```yaml
services:
  vote:
  result:
  worker:
  redis:
    volumes:
      - "./healthchecks:/healthchecks"
  db:
    volumes:
      - "db-data:/var/lib/postgresql/data"
      - "./healthchecks:/healthchecks"

volumes:
  db-data:
```

volumes

Persistenza e condivisione dei dati

```yaml
services:
  vote:
  result:
  worker:
  redis:
    healthcheck:
      test: /healthchecks/redis.sh
      interval: "5s"
  db:
    healthcheck:
      test: /healthchecks/postgres.sh
      interval: "5s"
```

healthcheck

Monitoraggio dello stato del container

Build e seed

```yaml
services:
  vote:
    build:
      context: ./vote
  result:
    build:
      context: ./result
  worker:
    build:
      context: ./worker
  redis:
    image: redis:alpine
  db:
    image: postgres:15-alpine
```

build

Definisce come costruire l'immagine a *runtime*

```yaml
services:
  # it won't run unless you specify
  # the "seed" profile
  # docker compose --profile seed up -d
  seed:
    build: ./seed-data
    profiles: ["seed"]
```

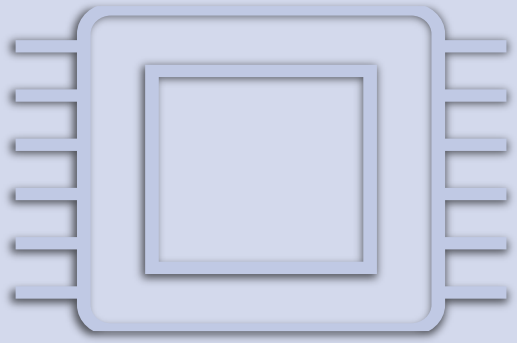
seed

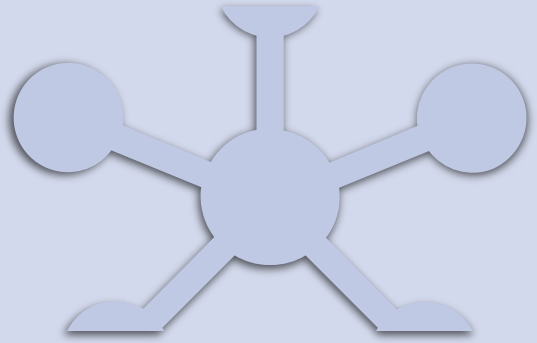
Permette di attivare alcuni servizi

Configurazione

Parametri principali del docker-compose.yaml

Parametri root





Services

Elenco dei container da avviare, ciascuno con la propria configurazione.

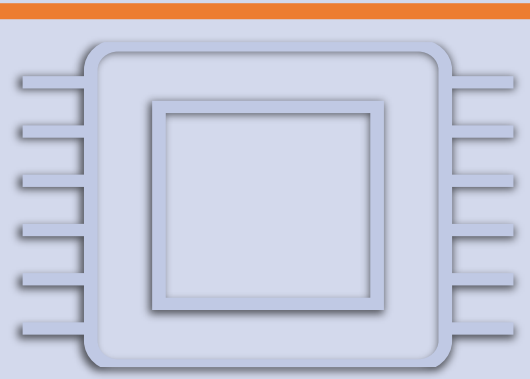
Volumes

- Definisce volumi condivisi e persistenti a livello globale.

Networks

- Definisce reti personalizzate peri servizi.

Parametri root

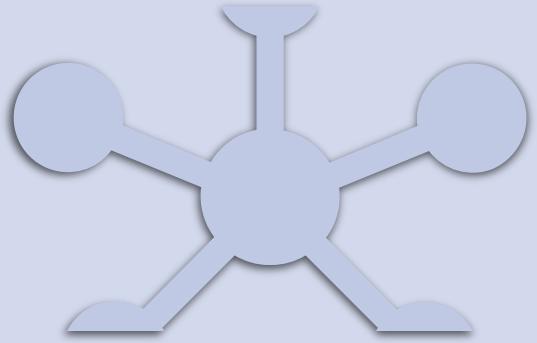


Services

Elenco dei container da avviare, ciascuno con la propria configurazione.

Volumes

- Definisce volumi condivisi e persistenti a livello globale.



Networks

- Definisce reti personalizzate peri servizi.

Dettaglio services

Servizio1

image: specifica l'immagine Docker da usare per il servizio.

build: Definisce come costruire l'immagine da una directory o Dockerfile.

ports: mappa le porte del container su quelle del sistema host.

volumes: monta volumi per persistenza del dato o per condividere file tra host e container.

networks: assegna il servizio ad una o più reti definite nel file.

depends_on: specifica l'ordine di avvio tra i servizi.

environment: imposta variabili d'ambiente nel container.

healthcheck: definisce come verificare lo stato di salute del container.

command: sovrascrive il comando di default dell'immagine.

restart: imposta la politica di avvio del container.

Esempio docker-compose.yaml

```
services:
  app1:
    image: <image_name>
    build:
      context: <path_to_build_context>
      dockerfile: <Dockerfile_name>
    ports:
      - "<host_port>:<container_port>"
    environment:
      - <ENV_VAR_NAME>=<value>
      - <ENV_VAR_NAME_2>=<value>
    volumes:
      - <vol_name>:<container_path>
    networks:
      - <net_name>
    depends_on:
      - <service_name>
    healthcheck:
      test: ["CMD", "<healthcheck_command>"]
      interval: <interval_in_sec>
      timeout: <timeout_in_sec>
      retries: <number_of_retries>
      start_period: <start_period_in_sec>
    command: <custom_command>
    restart: <restart_policy>
  db:
    image: <image_name>
    volumes:
      - <db_vol_name>:<db_data_path>
    environment:
      - <DB_ENV_VAR>=<value>
    networks:
      - <net_name>

volumes:
  <vol_name>:
    driver: <vol_driver>
  <db_vol_name>:

networks:
  <net_name>:
    driver: <net_driver>
```

Note sui placeholder

services

- <image_name>: es. nginx, node:18, myapp:latest
- <host_port>:<container_port>: es. 8080:80
- <ENV_VAR_NAME>: es. DEBUG, DB_USER
- <vol_name>: es. app-data
- <net_name>: es. fronted, backend
- <healthcheck_command>: es. curl –f http://localhost/ || exit 1
- <restart_policy>: no, always, on-failure, unless-stopped

volumes

- <vol_driver>: es. local

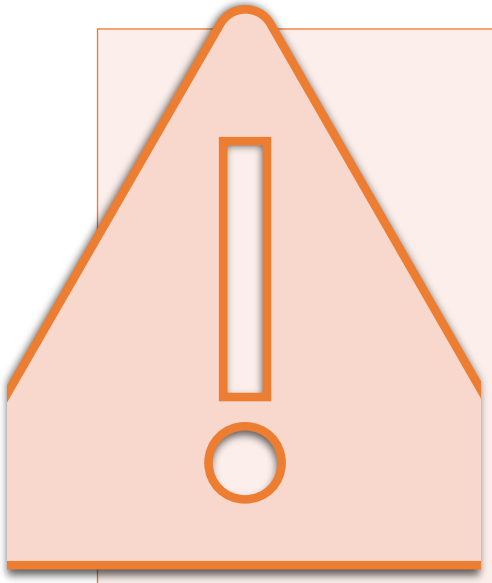
networks

- <net_driver>: es. bridge

CLI

Come usare il comando
docker compose

Naming convention



Il nome di default del file di configurazione sono i seguenti. In caso contrario, bisogna specificare il nome del file tramite il flag –f.

- docker-compose.yaml
- docker-compose.yml
- compose.yaml
- compose.yml

Avvio e gestione dei servizi

docker compose up

- Avvia i servizi definiti nel file docker-compose.yaml.
- Col flag –d avvia i servizi in background (detached mode).

docker compose down

- Ferma e rimuove container, reti e configurazioni.
- col flag –v rimuove anche i volumi associati.

docker compose restart

- Riavvia i container dei servizi.

docker compose stop

- Ferma i container senza rimuoverli.

docker compose start

- Riavvi container precedentemente fermati.

Monitoraggio e debug

docker compose ps

- Mostra lo stato dei container.

docker compose logs

- Visualizza i log dei servizi.
- col flag –f mostra i log in tempo reale (follow mode).

docker compose top

- Mostra i processi attivi nei container.

docker compose events

- Monitora in tempo reale gli eventi generati dai container.

Costruzione e configurazione

docker compose build

- Costruisce le immagini definite con build:

docker compose pull

- Scarica le immagini dai registry.

docker compose config

- Valida e mostra la configurazione finale.

Altri comandi

docker compose exec

- Esegue un comando all'interno di un container già in esecuzione.
- Es. docker compose exec app bash

docker compose version

- Mostra la versione di docker compose.

docker compose help

- Elenco completo dei comandi disponibili.

Link

- Docker Compose official guide
- Voting App
- Compose file
- Docker Compose CLI

YouTube Channels

- TechWorldWithNana
- That DevOps Guy
- Sistemista Italiano