Docker compose è un sistema di gestione di container Docker.  
A differenza di Kubernetes non offre autoscaling etc, pero’ comunque permette di fare molte delle cose viste con K8.  
  
Docker Compose, a differenza di docker plain, si usa quando abbiamo piu’ docker da tirare su, ciascuno con le sue immagini e configurazioni.   
Docker compose permette anche di creare una network interna per i docker in maniera tale che questi possano dialogare tra loro usando i nomi logici (dei servizi).  
  
Quando Si usa docker compose, ciascun applicativo deve comunque avere il suo docker file.  
Si puo’ prevedere di lanciare manualmente prima la docker build per creare l’immagine di ciascun applicativo prima di lanciare il docker-compose up, che sfrutta le immagini precedentemente prodotte per gli applicativi, oppure per semplicità si puo’ lanciare

Docker-compose up –build per far si che vengano generate le immagini docker dei docker file puntati nel docker compose.  
  
Es di **docker** file di progetto  
  
A screen shot of a computer

Description automatically generated  
  
Questo file va chiamato Dockerfile (senza estensione) e messo nella cartella del progetto.  
Poi nella cartella padre (es del repository) si crea **il docker-compose.yml** file  
  
A screen shot of a computer program

Description automatically generated

Spiegazione:  
nei **services** dichiariamo i container docker che docker-compose deve gestire.  
Il nome del service sarà anche il nome logico con il quale i container potranno essere contattati da altri container nella stessa sottorete.  
La sottorete creata è sotto il campo **networks**(in fondo) ed è etl-network.  
Stiamo creando un container maria\_db\_container, usando l’immagine definita (non la trova nel docker hub locale quindi la scarica dal docker hub registry online).

Ci sono 2 modi in cui si danno le variabili d’ambiente al container (quelle poi riferite negli applicativi). Tramite file esterno (**che deve chiamarsi .env per forza, altrimenti non funziona**) linkato con la property **env\_file:** e tramite la property **environment** con cui si listano e si danno direttamente le chiavi / valore. Qui stiamo usando entrambi gli approcci (il file .env che stiamo fornendo al docker-compose è unico per entrambi i docker che stiamo configurando e si trova nella stessa folder dove sta il docker compose).  
  
**NB:** mentre per il primo container, quello del db, puntiamo direttamente all’immagine nel docker registry , per il container applicativo vogliamo che sia docker-compose a mandare la generazione dell’immagine (usando il docker file dell’applicativo ) visto che stiamo usando il flag – build .  
Dobbiamo quindi definire qual è la cartella dell’applicativo e dove si trova il suo docker file.  
Visto che il file docker-compose.yml si trova nella cartella padre (es del repository) al cui interno c’e’ la cartella dell’applicazione etl-importer-updater (al cui interno c’e’ il docker file di etl-importer-updater) queste info le diamo con i campi  
**A screen shot of a computer

Description automatically generated**  
Anche per il container applicativo etl-importer-updater stiamo fornendo l’.env file e stiamo definendo delle variabili anche direttamente nel campo **environment** (**nello specifico stiamo settando anche il profilo spring di start dell’applicativo**  ).

(NB: $SPRING\_PROFILES\_ACTIVE è il nome della variabile da usare quando stiamo startando con docker e vogliamo settare quella che altrimenti avremmo indicato con

java -jar -Dspring.profiles.active=prod (se avessimo lanciato il jar manualmente)  
  
o con

mvn spring-boot:run -Dspring-boot.run.profiles=prod (se avessimo lanciato con mvn spring-boot)

)  
  
Come ultima istanza si noti che alla fine del docker-compose stiamo creando 2 volumes, uno che montiamo (richiamandolo) nel docker del db e uno nel docker applicativo (specificando quando montiamo dove vengono montati)  
  
A questo punto per startare il tutto si fa:  
  
Step1) builda applicativo etl-importer-updater   
 vai dove sta il pom.xml e chiama mvn clean install

Step2) vai dove sta il docker-compose.yml e lancia  
 docker-compose -f docker-compose.yml up –build   
  
Questo equivale ad aver lanciato 2 docker build sui docker file dei 2 container e ad averli runnati con docker run [immagine]