

Presentazione Progetto

Studente: Maiorani Simone - 346627

Professore: Reali Gianluca

Introduzione

Per questo progetto, ho realizzato una semplice applicazione client-server per la gestione di una lista di prodotti d'acquistare.

Nello specifico, l'applicazione tenta di emulare il comportamento di una lista della spesa, consentendo all'utente di inserire, recuperare ed eliminare i vari prodotti.

Nel dettaglio, l'applicazione è composta da tre microservizi:

- Un **database relazionale** per l'archiviazione dei dati.
- Un'**API REST** per gestire le comunicazioni tra il client ed il database.
- Un'**interfaccia web** per consentire all'utente di eseguire le varie operazioni.

I tre microservizi vengono fatti girare all'interno di tre distinti **container Docker** per una migliore distribuzione e gestione. Essendo in ambiente di sviluppo e avendo più container, ho definito un file **Docker Compose**. Quest'unico file di tipo YAML funge da "collante" e consente di definire tutti i microservizi e le configurazioni necessarie per l'esecuzione dell'applicazione, inclusi i container, la rete, il volume e le variabili d'ambiente.



The screenshot displays a web application interface for managing a shopping list. On the left side, there are two orange buttons: 'Get List' and 'Reset List'. The main content area is titled 'Inserisci i prodotti da acquistare' in a bold, brown font. Below this title, there is a form with three input fields: 'Nome Prodotto', 'Quantità', and 'Prezzo (€/int)'. Each field has a corresponding label above it. To the right of the form is a placeholder image showing a shopping cart in a supermarket aisle. At the bottom of the form is an orange button labeled 'Inserisci'.

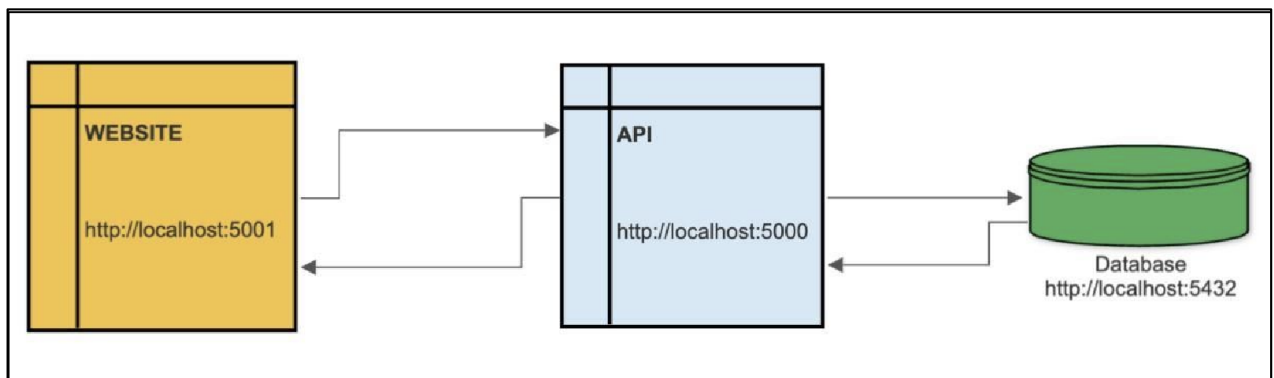
Interfaccia web dell'applicazione

Configurazione

Le tecnologie che sono state utilizzate per la realizzazione del progetto sono:

- **PostgreSQL** per la realizzazione del database.
- **Python (*Flask*)** per la realizzazione del front-end e del back-end.
- **Libreria SQLAlchemy** per l'esecuzione delle query sul database in Python.
- **HTML, CSS, Javascript** e jQuery per l'interazione dell'utente con il front-end.

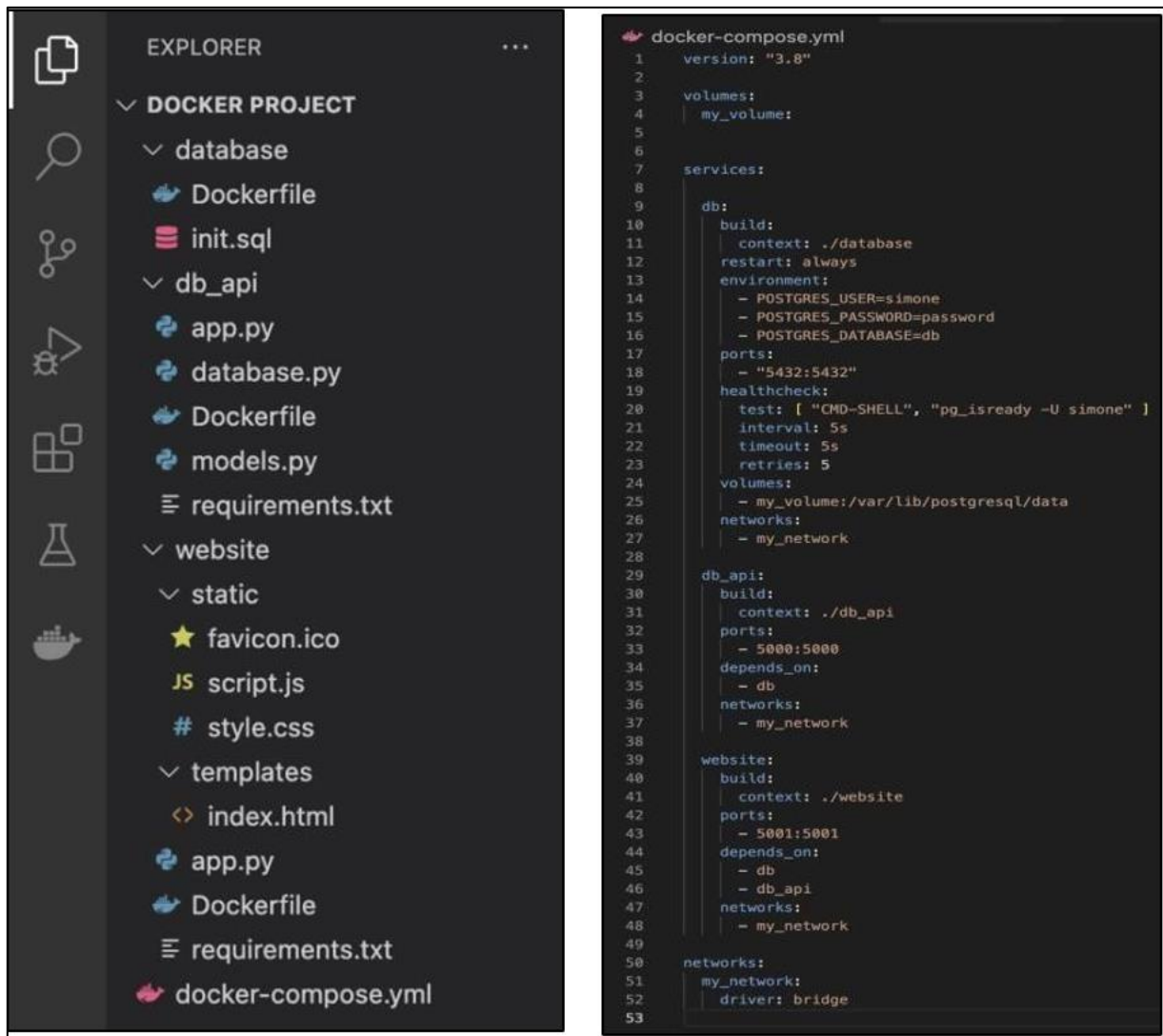
Per ogni microservizio ho definito un **DockerFile**, necessario per descrivere come l'immagine Docker dev'essere costruita. Infatti, all'interno del file abbiamo le librerie, le dipendenze e le configurazioni necessarie per far funzionare ogni microservizio.



```
db-api > Dockerfile
1  # Creazione di un Dockerfile che definisce un'immagine per il servizio db_api.
2
3  FROM python:3.9
4
5  WORKDIR /app
6  COPY ./ .
7  COPY requirements.txt requirements.txt
8
9  RUN pip install -r requirements.txt
10
11 ENTRYPOINT ["python"]
12 CMD ["app.py"]
13 |
```

Esempio di uno dei Dockerfile utilizzati

Esecuzione



Project Directory

File docker-compose.yml

Per eseguire correttamente il progetto è necessario posizionarsi correttamente nella directory di quest'ultimo ed eseguire da terminale il comando `docker compose up` che realizzerà e metterà in funzione ciascun servizio partendo dal rispettivo Dockerfile specificato all'interno di ciascuna sottocartella.

Fatto ciò, è possibile accedere all'interfaccia web dell'applicazione attraverso l'indirizzo <http://localhost:5001> per andare ad eseguire le varie operazioni di GET, POST e DELETE per la gestione di un'ipotetica lista della spesa.

Al container db, contenente il database, è stato affiancato un **volume** permanente per rendere i dati persistenti e utilizzabili anche a seguito della sua chiusura.

Inoltre, per il container db viene utilizzato un parametro di **healthcheck** per verificarne periodicamente lo stato di salute e accertarsi che stia funzionando correttamente. In caso di fallimento del container, vengono effettuati un massimo di 5 tentativi, ad un intervallo di 5 secondi e con un timeout di 5 secondi.

Il comando utilizzato è `pg_isready -U simone`, il quale va a verificare se il database PostgreSQL è pronto a ricevere connessioni.

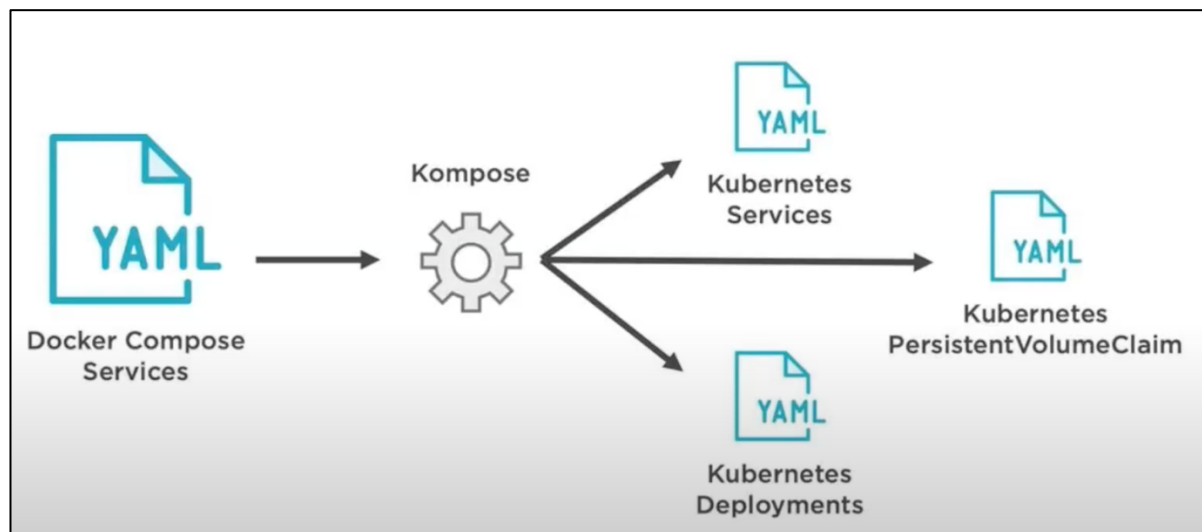
```
dockerproject-db-1 | 2023-05-05 22:16:29.682 UTC [66] LOG: checkpoint starting: time
dockerproject-db-1 | 2023-05-05 22:16:35.110 UTC [66] LOG: checkpoint complete: wrote 55 buffers (0.3%); 0 WAL file(s) added, 0 removed, 0 recycled; w
rite=5.409 s, sync=0.010 s, total=5.429 s; sync files=20, longest=0.006 s, average=0.001 s; distance=270 kB, estimate=270 kB
```

Output da terminale

Kubernetes

Il progetto prevede anche un'implementazione orchestrata da **Kubernetes**, che consente di gestire e scalare i container in modo efficace e automatizzato.

Per realizzare tale implementazione ho utilizzato il tool di conversione **Kompose** di Kubernetes (<https://kompose.io/>) che essenzialmente prende in input il file `docker-compose.yml` e restituisce in output i rispettivi file YAML di Kubernetes per il corretto funzionamento del programma.



Una volta installato il Kompose, è sufficiente posizionarsi correttamente nella directory del progetto ed eseguire il comando `kompose convert -f docker-compose.yml`.

Ciò restituisce in output i rispettivi file che, una volta apportate le dovute modifiche, possono essere lanciati e fermati insieme eseguendo da terminale rispettivamente i file `run.sh` e `stop.sh` presenti all'interno della cartella del progetto.






















```
1 #!/bin/bash
2
3 # Creazione delle immagini
4 docker build -t k8sdb ./database
5 docker build -t k8sapi ./db-api
6 docker build -t k8swebsite ./website
7
8 # Rimozione di tutti i pod, deployment e service
9 kubectl delete pod --all
10 kubectl delete deployment --all
11 kubectl delete service --all
12
13 # Applicazione dei file YAML
14 kubectl apply -f my-volume-persistentvolumeclaim.yaml
15 kubectl apply -f progetto2-my-network-networkpolicy.yaml
16
17 kubectl apply -f db-deployment.yaml
18 kubectl apply -f db-api-deployment.yaml
19 kubectl apply -f website-deployment.yaml
20
21 kubectl apply -f db-service.yaml
22 kubectl apply -f db-api-service.yaml
23 kubectl apply -f website-service.yaml
24
25 # Sleep per dare il tempo ai pod di avviarsi
26 sleep 7
27
28 # Port forwarding
29 kubectl port-forward service/db-api 5000:5000 &
30
31 sleep 2
32
33 kubectl port-forward service/website 5001:5001
34
```

run.sh

```
1 #!/bin/bash
2
3 # Rimozione di tutti i pod, deployment, service e persistent volume
4 kubectl delete -f db-deployment.yaml
5 kubectl delete -f db-api-deployment.yaml
6 kubectl delete -f website-deployment.yaml
7 kubectl delete -f db-service.yaml
8 kubectl delete -f db-api-service.yaml
9 kubectl delete -f website-service.yaml
10 kubectl delete -f progetto2-my-network-networkpolicy.yaml
11 kubectl delete -f my-volume-persistentvolumeclaim.yaml
12
```

stop.sh

Il funzionamento dell'applicazione è analogo.

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started ↑	Actions
<input type="checkbox"/>	 dockerproject		Running (3/3)		5 seconds ago	  
<input type="checkbox"/>	 db-1 2ae244b4f19e 	dockerproject-db	Running	5432:5432 	5 seconds ago	  
<input type="checkbox"/>	 db_api-1 7e36fa9fc2b8 	dockerproject-db_api	Running	5000:5000 	5 seconds ago	  
<input type="checkbox"/>	 website-1 cdca751ec640 	dockerproject-website	Running	5001:5001 	5 seconds ago	  

Struttura Docker

<input type="checkbox"/>	 k8s_POD_db-api-9fd8676b7-zm 56b2b730aac9 	registry.k8s.io/pause:3.1	Running		14 seconds ago	  
<input type="checkbox"/>	 k8s_POD_website-846cfcb87-rv be76aeffa102 	registry.k8s.io/pause:3.1	Running		14 seconds ago	  
<input type="checkbox"/>	 k8s_db-api_db-api-9fd8676b7-z 4d1b15728b9b 	sha256:b3e1da196384c	Running		13 seconds ago	  
<input type="checkbox"/>	 k8s_website_website-846cfcb87 a23a5c139b8c 	sha256:3fe0b48a301e6	Running		13 seconds ago	  
<input type="checkbox"/>	 k8s_POD_db-74768f6f49-4h2ms 002ae15f4338 	registry.k8s.io/pause:3.1	Running		0 seconds ago	  
<input type="checkbox"/>	 k8s_db_db-74768f6f49-4h2ms_c 35437bdf67d9 	sha256:3482c41d77c2c	Running		0 seconds ago	  

Struttura Kubernetes