DOCUMENTO DI PROGETTO - VIRTUAL NETWORKS AND CLOUD COMPUTING

Studente: Filippo Francisci

Matricola: 350395

E-mail: filippo.francisci@studenti.unipg.it

Introduzione

L'obiettivo del progetto è quello di sviluppare una semplice applicazione *client-server*, basandosi sull'utilizzo di *Kubernetes*. Per fare ciò si sviluppano i seguenti servizi:

- Un database preimpostato con un elenco di auto;
- Un'interfaccia *web* per permettere l'interazione dell'utente;
- Un'interfaccia per gestire la comunicazione tra l'utente e il *dataset*.

All'utente viene presentato un *form* online da compilare e in base ai campi immessi gli verrà restituita la lista delle auto che rispettano i termini di ricerca impostati. Il tutto viene implementato in *Kubernetes*, che permette di creare **POD** e di orchestrarli in modo adeguato, per consentire l'interazione tra i servizi realizzati. In particolare, viene definito un **POD** per ogni servizio, basandosi su delle immagini appositamente create tramite *Dockerfile*.

Implementazione

La logica dell'applicazione implementata è compresa nella directory ***src*** del progetto, dove si sono realizzati i singoli servizi in esame. Le principali tecnologie impiegate per la messa in opera degli stessi sono:

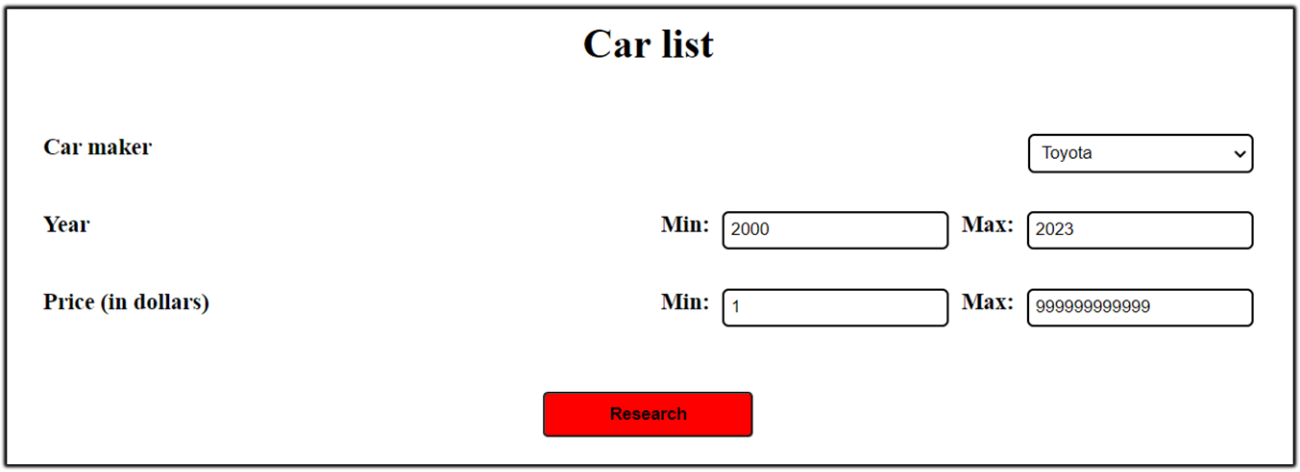
- **MySQL**: database usato per rappresentare il catalogo delle auto;
- **Flask**: framework scritto in *Python*, molto utile per realizzare semplici applicazioni *web*;
- Tutte le tecnologie utili allo sviluppo dell'interfaccia *web* del *client* (**HTML**, **CSS**, **Javascript**).

Per ogni servizio sono stati realizzati degli appositi *Dockerfile* per creare delle immagini personalizzate, utili nell'implementazione delle singole funzionalità dell'applicazione. Si fornisce di seguito un esempio del *Dockerfile* caratterizzante l'immagine dell'interfaccia *web* realizzata.

```
# Create a Dockerfile for the web page

FROM python:3.9

WORKDIR /Carapp

COPY ./static/style.css ./static/style.css

COPY ./templates/web-page.html ./templates/web-page.html

COPY ./web-app.py ./web-app.py

COPY  requirements.txt requirements.txt

RUN pip install -r requirements.txt

ENTRYPOINT ["python", "web-app.py"]
```

Esempio di uno dei *Dockerfile* realizzati

In questo caso si parte dall'immagine standard di *Python*, si installano le librerie di interesse e si esegue lo script che espone l'interfaccia web. Di seguito si riporta la pagina HTML realizzata.



Visualizzazione interfaccia *web*

L'utente è libero di inserire i parametri di suo interesse nei vari campi del *form* realizzato e ricercare le informazioni nel *database* cliccando l'apposito bottone "**Research**". Il risultato della ricerca verrà poi riportato in basso sotto forma di lista. Per mantenere i dati si è deciso di realizzare un *database* **MySQL**, il quale si è inizializzato con i valori del *dataset* di riferimento. I dati rappresentano diverse tipologie di automobili con parametri prestazionali e costo in dollari annessi. I campi definiti nel *dataset* sono i seguenti:

- **Id**: identifica incrementalmente i record del *dataset*;
- **Car Make**: produttore dell'auto;
- **Car Model**: modello di automobile caratterizzato;
- **Production Year**: anno in cui l'auto è stata prodotta;

- **Engine Size L**: fornisce informazioni sulla cilindrata del motore, misurata in litri (L);
- **Horsepower**: cavalli di un'auto che misurano la potenza del motore;
- **Torque lbft**: trazione che misura la forza esercitata dalle ruote dell'auto contro la superficie stradale per consentire all'auto di muoversi avanti o indietro;
- **0-60 MPH Time (seconds)**: rappresenta il tempo impiegato dall'auto per accelerare da 0 a 60 miglia orarie;
- **Price (in USD)**: prezzo dell'auto in dollari.

Anche qui si è creato il *Dockerfile* che prende l'immagine **MySQL** di base e crea semplicemente il *database* con le informazioni descritte precedentemente. Per poter recuperare le informazioni dallo stesso e passarle al *client*, si è utilizzata la libreria **SQLAlchemy** che offre delle efficienti funzionalità di connessione al *database*. Tramite l'impiego di *file* **YAML** sono stati istanziati i vari servizi, insieme ad un volume, utile per immagazzinare in modo persistente, sicuro ed efficiente i dati caricati. Si riporta come esempio il *file* **YAML** che permette di configurare il servizio che espone l'interfaccia *web*:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
        - name: web-app
          image: localhost:4000/app-image
          ports:
            - containerPort: 5000
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: web-app
  labels:
    app: web-app
spec:
  selector:
    app: web-app
  ports:
    - name: app-port
      port: 5000
      targetPort: 5000
  type: LoadBalancer
```

Esempio di un *file* YAML

Per poter mettere in opera in modo automatico l'applicazione sono stati creati anche degli appositi *script*. Inoltre, si è deciso di impiegare un registro su cui caricare le immagini create, utilizzate poi per istanziare i servizi di interesse. Lo *script* ***start.bat*** avvia l'applicazione, creando le immagini, caricandole sul registro sopra detto e costruendo i **POD** da mandare in esecuzione. Infine, tramite il comando ***port-forward***, si espone l'applicazione sulla porta 5000 (porta di *default* utilizzata da **Flask**). A questo punto connettendosi all'**URL** *http://localhost:5000* si accede all'interfaccia *web* da cui è possibile interagire con l'applicazione. Lo *script* ***stop.bat***, invece, semplicemente elimina il volume e tutti i **POD** messi in opera, oltre ad arrestare e rimuovere il registro su cui si erano caricate le immagini. Di seguito si riportano i due *script* realizzati.

```
# Build new images

docker build -t mysql-db ./src/mysql-db
docker build -t db-con ./src/db-con
docker build -t app-image ./src/web-app

# Create a container registry to keep all the images

docker run -d -p 4000:5000 --restart=always --name registry registry:2

#  Create a new tag for an image

docker tag mysql-db localhost:4000/mysql-db
docker tag db-con localhost:4000/db-con
docker tag app-image localhost:4000/app-image

# Push the images on the local registry

docker push localhost:4000/mysql-db
docker push localhost:4000/db-con
docker push localhost:4000/app-image

# Define all the yaml files to deploy the application

kubectl apply -f yaml-file/mysql-pvc.yml
kubectl apply -f yaml-file/mysql-deployment.yml
kubectl apply -f yaml-file/db-con-deployment.yml
kubectl apply -f yaml-file/web-app-deployment.yml

# Sleep command

timeout /t 10

# Port forwarding

kubectl port-forward service/web-app 5000:5000
```

script start.bat

```
# Remove all

kubectl delete -f yaml-file/mysql-deployment.yml
kubectl delete -f yaml-file/db-con-deployment.yml
kubectl delete -f yaml-file/web-app-deployment.yml
kubectl delete -f yaml-file/mysql-pvc.yml

# Remove the container registry

docker stop registry
docker rm registry
```

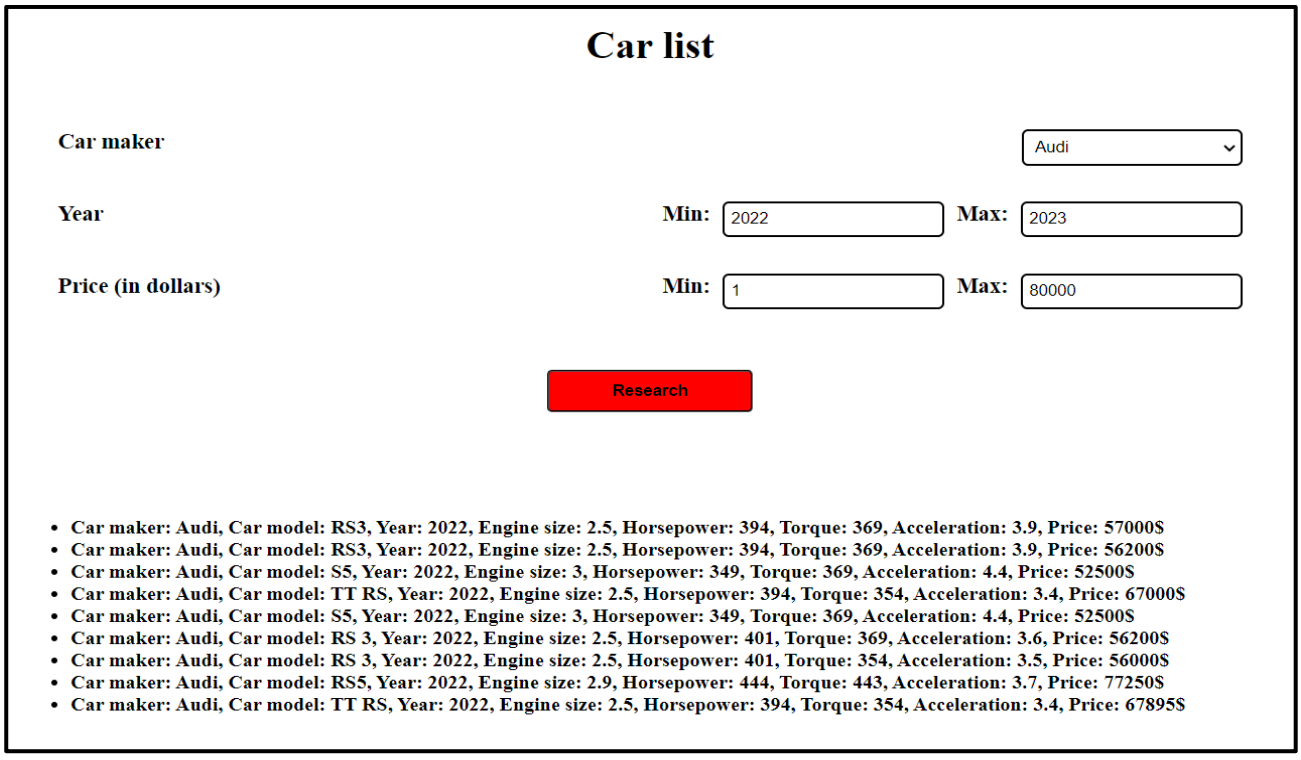
script stop.bat

Per verificare il corretto funzionamento dell'applicazione si usa il seguente comando: **kubectl get pods**.

```
C:\Windows\System32>kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
datab-866759589b-92bww     1/1     Running   0          90s
db-con-647f84788f-ph6gx    1/1     Running   0          89s
web-app-86cfdd87d-tgv2c    1/1     Running   0          88s
```

Visualizzazione *POD* in esecuzione

Dall'immagine riportata, si verifica la presenza di tre **POD** in stato "***Running***", che conferma il fatto che tutti i servizi sono attivi e funzionanti. A questo punto basta collegarsi all'interfaccia *web* dell'applicazione per fare le operazioni di interesse. Si presenta quindi una possibile ricerca che l'utente può effettuare.



Possibile utilizzo dell'applicazione

Conclusioni e sviluppi futuri

L'applicazione realizzata permette di verificare un possibile utilizzo di una tecnologia importante e diffusa come *Kubernetes*. *Kubernetes* permette di orchestrare *container* in ambienti molto più complessi di quello realizzato in questo progetto. Probabilmente sarebbe stato più opportuno implementare una tecnologia di tipo *docker-compose* che con un unico *file* **YAML** permette il *deployment* di tutti i *container* in modo semplice e veloce. L'impiego di *Kubernetes* per un'applicazione semplice come quella realizzata e con così pochi *container* da istanziare, rischia inutilmente di appesantire l'esecuzione della stessa, offrendo però al contempo funzionalità più avanzate, come la gestione dello stato di salute dei *container*. Lo scopo del progetto era comunque quello di verificare l'applicabilità, in un caso pratico, di una tecnologia come *Kubernetes* che è tra le più impiegate nel campo dell'informatica per le sue enormi qualità, tra cui il controllo dello stato di salute dei *container* e il loro successivo riavvio nel caso in cui qualcosa dovesse andare storto. L'applicazione potrebbe essere migliorata in vari modi, ad esempio aggiungendo più criteri di ricerca sull'interfaccia utente (come il modello dell'auto o anno di produzione della stessa) o nuovi campi d'informazioni nel *dataset* impiegato, come la quantità di prodotti disponibili in magazzino per ogni modello, con l'obiettivo di simulare il sito di una concessionaria di automobili.