



CA' FOSCARI UNIVERSITY

PROGETTO LABORATORIO E AMMINISTRAZIONE DI SISTEMA

**Workflow pratico all'installazione Docker e
creazione di Container e Immagini relativi a
un progetto dimostrativo.**

David Ambros
881443

February 3, 2023

Contents

1	Introduzione	2
2	Installazione	2
3	Container	3
3.1	Comandi utili	3
3.2	Come si accede ad un container?	4
4	Workflow creazione progetto demo	5
4.1	Overview progetto dimostrativo	5
4.2	Setup Container MongoDB	5
4.3	Setup Mongo-Express e comunicazione tra i due Container	6
4.4	Comunicazione tra Mongo-Express e localhost	6
4.5	Collegarsi in modo programmatico al database	8
4.6	Docker Compose	9
4.7	Dockerfile	10
5	Conclusione	11

LAS 2023

David Ambros

February 2023

1 Introduzione

In questa guida in stile "how to" vedremo come installare e usare docker, Docker Compose, Dockerfile, per creare un progetto pratico usando MongoDB e Mongo-Express per interfacciarsi al database.

La parte iniziale prevede la creazione delle componenti tramite comandi docker, e successivamente verrà dimostrato come ottenere lo stesso risultato tramite uso di Docker Compose.

Ci sarà anche una dimostrazione su come creare la nostra immagine relativa all'applicazione backend che useremo per interfacciarsi con mongoDB.

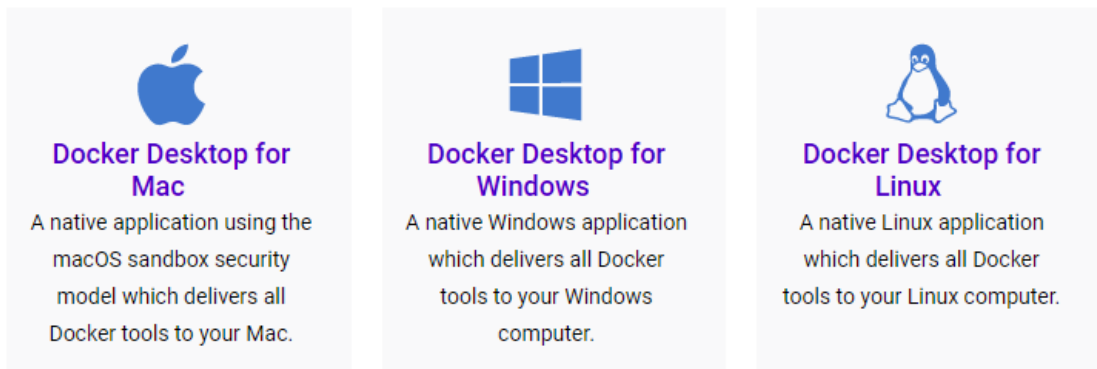
Quale è la differenza tra Docker e una normale VM? Perché Docker è conveniente in certe situazioni? La risposta è che Docker virtualizza il livello applicazione, e utilizza kernel e hardware del computer host, mentre una VM virtualizza sia il livello applicazione che il livello kernel quindi una VM ha un proprio sistema operativo, e usa solamente il hardware dell' host.

Questo implica che:

- La dimensione delle Docker image è molto inferiore.
- I container Docker sono molto più veloci nel processo di startup e run.
- Le Virtual Machine tuttavia evitano problemi di compatibilità tra sistemi operativi diversi, che si potrebbero incontrare usando Docker, ma facilmente risolvibili tramite l'uso di Docker Desktop che fa da "bridge" tra Docker e il kernel del host.

2 Installazione

Il primo passo per installare Docker è andare sul [sito ufficiale](#) e scegliere l'installer di Docker Desktop adatto al proprio sistema operativo:



Una volta scaricato ed eseguito attendiamo che l'installazione si completi:

Docker Desktop 4.16.3

Unpacking files...

```
Unpacking file: resources/services.tar
Unpacking file: resources/linux-daemon-options.json
Unpacking file: resources/lcow-kernel
Unpacking file: resources/lcow-initrd.img
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/bin/docker
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_icd.json
```

Una volta aperto, ci potrebbe venir detto di aggiornare wsl, in tal caso bisogna aprire il cmd e scrivere il seguente comando:

```
1 wsl --update
```

Ed eventualmente, in caso si ricevesse un errore di questo tipo:

```
1 "Hardware assisted virtualization and data execution protection must be enabled in the BIOS"
```

Bisogna aprire powershell come amministratore ed eseguire il comando:

```
1 dism.exe /Online /Enable-Feature:Microsoft-Hyper-V /All
```

Ora che l'installazione è completata, siamo pronti a creare il nostro primo container.

3 Container

3.1 Comandi utili

Che cos'è un container? Un container è l'istanza in esecuzione relativa a un immagine. Le immagini possono essere scaricate da un repository come DockerHub ad esempio.

- Per scaricare un immagine da DockerHub, bisogna eseguire il comando:

```
1 docker pull <nome-immagine>
```

- Per vedere la lista di tutte le immagini esistenti sul sistema locale:

```
1 docker images
```

- Per tirare su un container a partire da un immagine:

```
1 docker run <nome-immagine>
```

Tale comando eseguirà un immagine in un container, se l'immagine non esiste nel sistema, docker la andrà a scaricare da DockerHub. Quindi in un solo comando vengono eseguiti i seguenti due:

```
1 docker pull <nome-immagine>
2 docker start
```

```
C:\Users\Stryker>docker run redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
8740c948ffd4: Pull complete
a2271c958e57: Pull complete
9274cf84d443: Pull complete
27109f8012f4: Pull complete
17c6f4c2f686: Pull complete
9b9047e460ee: Pull complete
Digest: sha256:579dd1c5fc096aeac8d20bac605c95e9cbea11327243dec4da93e507af90c5a2
Status: Downloaded newer image for redis:latest
```

- Per tirare su un container a partire da un immagine, -d sta per "deattached" cioè il container non sarà vincolato dal terminale attuale:

```
1 docker run -d <nome-immagine>
```

- Per vedere la lista di tutti i container in esecuzione:

```
1 docker ps
```

- Il comando "docker ps", mostra tutti i container in esecuzione, il seguente comando invece mostra la lista di tutti i container sia in stato di esecuzione, che in stato di stop:

```
1 docker ps -a
```

- Per interrompere l'esecuzione di un container:

```
1 docker stop <id-container>
```

- Per eseguire un container interrotto, già esistente:

```
1 docker start <id-container>
```

Attenzione che la keyword non è più "run" ma "start"

- Visualizzare il terminale di un container in esecuzione:

```
1 docker exec -it <id-container> /bin/bash
```

"-it" sta per interactive terminal. Ora siamo dentro il container come root user.

3.2 Come si accede ad un container?

Ogni container ha una porta sulla quale sta in ascolto, che si può vedere tramite il comando "docker ps":

```
C:\Users\Stryker>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS               NAMES
9c4fada94852   redis    "docker-entrypoint.s..." 3 minutes ago  Up 3 minutes  6379/tcp            mystifying_darwin
```

Quello che dobbiamo fare è un binding tra una porta della nostra macchina locale e la porta del nostro container, in modo che le richieste inviate alla porta locale, siano spedite al nostro container grazie al binding.

Il binding va specificato nel momento in cui mandiamo in esecuzione un container tramite il comando:

```
1 docker run -p<host-port>:<container-port> -d <nome-immagine>
```

```
C:\Users\Stryker>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS               NAMES
27574765a7dd   redis    "docker-entrypoint.s..." 4 seconds ago  Up 3 seconds  0.0.0.0:6000->6379/tcp  frosty_yalow
```

4 Workflow creazione progetto demo

4.1 Overview progetto dimostrativo

Come anticipato in precedenza, questa sezione è dedicata all'uso pratico di Docker.

1. Per prima cosa scaricheremo mongo e mongo-express. Mongo è un Database non relazionale, Mongo-Express ci permetterà di avere un interfaccia grafica direttamente da browser per il nostro DB.
2. Vedremo come far comunicare i container di Mongo e Mongo-Express.
3. Vedremo come far comunicare Mongo-Express con la nostra macchina locale.
4. Vedremo come interfacciarci in modo programmatico al database Mongo senza l'uso di Mongo-Express, usando un applicazione NodeJs.
5. Vedremo come automatizzare tutto il processo di pull immagini, creazione container e creazione sottorete, usando Docker Compose.
6. E infine creeremo un immagine a partire dall'applicazione NodeJs in modo da poterla usare come server backend e creare un container a partire da essa, in modo da capire il workflow che viene fatto in un ambiente di sviluppo tradizionale.

4.2 Setup Container MongoDB

Per prima cosa scarichiamo mongod e mongo-express, che servirà per interfacciarci con il database

- ```
1 docker pull mongo
2 docker pull mongo-express
```

Ora dobbiamo creare una sottorete dedicata a tutti i container che avremo, in modo che essi possano comunicare tra di loro.

- ```
1 docker network create mongo-network
```

Usiamo il comando "docker run" per eseguire il container di mongo a partire dalla sua immagine, specificando che la porta sulla quale invieremo messaggi al container è la stessa porta di mongod, inoltre configuriamo le variabili d'ambiente usando la keywork "-e" e impostando un username e una password, diamo un nome al container attraverso "-name", e impostiamo la sottorete che abbiamo appena creato usando "-net":

- ```
1 docker run -p 27017:27017 -d \
2 -e MONGO_INITDB_ROOT_USERNAME=admin \
3 -e MONGO_INITDB_ROOT_PASSWORD=password \
4 --name mongod \
5 --net mongo-network mongo
```

Tramite il comando:

- ```
1 docker logs <id-container>
```

dovremmo vedere qualcosa di questo genere:

```
[{"t":{"$date":"2023-02-02T12:19:04.479400:00"},"s":"I", "c":"REPL", "id":5123808, "ctx":"main","msg":"Successfully registered PrimaryOnlyService", "attr":{"service":"ShardSplitDonorService","namespace":"config.tenantSplitDonors"}}]
[{"t":{"$date":"2023-02-02T12:19:04.479400:00"},"s":"I", "c":"CONTROL", "id":5945603, "ctx":"main","msg":"Multi threading initialized"}]
[{"t":{"$date":"2023-02-02T12:19:04.480400:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting", "attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","hostName":"22e0f3d6da32"}}]
[{"t":{"$date":"2023-02-02T12:19:04.480400:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info", "attr":{"buildInfo":{"version":"6.0.4","gitVersion":"44ff59461c135638a7e710f385a66bd2cf547","openSSLVersion":"OpenSSL 1.1.1f 31 Mar 2020","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu2004","distarch":"x86_64","target_arch":"x86_64"}}}}}
[{"t":{"$date":"2023-02-02T12:19:04.480400:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System", "attr":{"os":{"name":"Ubuntu","version":"20.04"}}}]
[{"t":{"$date":"2023-02-02T12:19:04.480400:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line", "attr":{"options":{"net":{"bindIp":"*"},"security":{"authorization":"enabled"}}}}}
[{"t":{"$date":"2023-02-02T12:19:04.481400:00"},"s":"I", "c":"STORAGE", "id":22270, "ctx":"initandlisten","msg":"Storage engine to use detected by data files", "attr":{"dbPath":"/data/db","storageEngine":"wiredtiger"}}]
[{"t":{"$date":"2023-02-02T12:19:04.481400:00"},"s":"I", "c":"STORAGE", "id":22297, "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://docs.mongodb.org/core/production-fs/#xfs"}, {"t":{"$date":"2023-02-02T12:19:04.481400:00"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Opening WiredTiger", "attr":{"config":{"create_cache_size=479M,session_max=33000,eviction:(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,remove=true,path=journal,compressor=snappy),builtin_extension_config=(zstd=(compression_level=6)),file_manager=(close_idle_time=600,close_scan_interval=10,close_handle_minimum=2000),statistics_log=(wait=0),json_output=(error,message),verbose={recovery_progress:1,checkpoint_progress:1,compact_progress:1,backup:0,checkpoint:0,compact:0,evict:0,histogram_stores:0,recovery:0,rs:0,snapshots:0,timestamp:0,transaction:0,verify:0,log:0}}}}}
[{"t":{"$date":"2023-02-02T12:19:05.543400:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened", "attr":{"durationMillis":1062}}]
[{"t":{"$date":"2023-02-02T12:19:05.544400:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WiredTiger recovery/timestamp", "attr":{"recoveryTimestamp":{"t":{"$date":"2023-02-02T12:19:05.544400:00"},"s":"I", "c":"CONTROL", "id":22170, "ctx":"initandlisten","msg":"sys/kernel/mem/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'.", "tag":"startupWarnings"}}]
[{"t":{"$date":"2023-02-02T12:19:05.555400:00"},"s":"W", "c":"CONTROL", "id":5123308, "ctx":"initandlisten","msg":"vm.max_map_count is too low", "attr":{"currentValue":65536,"recommendedMinimum":1677728,"maxConn":6883680},"tags":["startupWarnings"]}]]
[{"t":{"$date":"2023-02-02T12:19:05.558400:00"},"s":"I", "c":"NETWORK", "id":4915702, "ctx":"initandlisten","msg":"Updated wire specification", "attr":{"oldSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":17},"isInternalClient":true,"newSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInternalClient":{"minWireVersion":17,"maxWireVersion":17},"isInternalClient":true}}}
[{"t":{"$date":"2023-02-02T12:19:05.558400:00"},"s":"I", "c":"REPL", "id":5853300, "ctx":"initandlisten","msg":"Current featureCompatibilityVersion value", "attr":{"featureCompatibilityVersion":"6.0","context":"startup"}}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Clearing temp directory"}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"CONTROL", "id":20536, "ctx":"initandlisten","msg":"Flow Control is enabled on this deployment"}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"FTDC", "id":20625, "ctx":"initandlisten","msg":"Initializing full-time diagnostic data capture", "attr":{"dataDirectory":"/data/db/diagnostic.data"}}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"REPL", "id":6015317, "ctx":"initandlisten","msg":"Setting new configuration state", "attr":{"newState":"ConfigReplicationDisabled","oldState":"ConfigReplicationDisabled"}}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Timestamp monitor starting"}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"initandlisten","msg":"Listening on", "attr":{"address":"/tmp/mongod-27017.sock"}}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"initandlisten","msg":"Listening on", "attr":{"address":"0.0.0.0"}}]
[{"t":{"$date":"2023-02-02T12:19:05.564400:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"initandlisten","msg":"Waiting for connections", "attr":{"port":27017,"ssl":"off"}}]
C:\Users\Stryker>
```

4.3 Setup Mongo-Express e comunicazione tra i due Container

Ora tocca a mongo-express, che ci offrirà un UI per mongo. Se andiamo su DockerHub possiamo vedere che nelle variabili d'ambiente c'è ne è una in particolare chiamata:

```
1 ME_CONFIG_MONGODB_SERVER
```

Environment variables are passed to the `run` command for configuring a mongo-express container.

Name	Default	Description
ME_CONFIG_BASICAUTH_USERNAME	' '	mongo-express web username
ME_CONFIG_BASICAUTH_PASSWORD	' '	mongo-express web password
ME_CONFIG_MONGODB_ENABLE_ADMIN	'true'	Enable admin access to all databases. Send string
ME_CONFIG_MONGODB_ADMINUSERNAME	' '	MongoDB admin username
ME_CONFIG_MONGODB_ADMINPASSWORD	' '	MongoDB admin password
ME_CONFIG_MONGODB_PORT	27017	MongoDB port
ME_CONFIG_MONGODB_SERVER	'mongo'	MongoDB container name. Use comma delimited list
ME_CONFIG_OPTIONS_EDITORTHEME	'default'	mongo-express editor color theme, [more here](htt
ME_CONFIG_REQUEST_SIZE	'100kb'	Maximum payload size. CRUD operations above this
ME_CONFIG_SITE_BASEURL	'/'	Set the baseUrl to ease mounting at a subdirector
ME_CONFIG_SITE_COOKIESECRET	'cookiesecret'	String used by [cookie-parser middleware](https:/
ME_CONFIG_SITE_SESSIONSECRET	'sessionsecret'	String used to sign the session ID cookie by [exp
ME_CONFIG_SITE_SSL_ENABLED	'false'	Enable SSL.
ME_CONFIG_SITE_SSL_CRT_PATH	' '	SSL certificate file.
ME_CONFIG_SITE_SSL_KEY_PATH	' '	SSL key file.

Questa variabile fa in modo che mongo-express si possa connettere a mongodb, ma solo grazie al fatto che abbiamo configurato la sottorete in precedenza, quindi ora dobbiamo avviare mongo-express nella stessa sottorete:

```
1 docker run -d -p 8081:8081 \  
2 -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin \  
3 -e ME_CONFIG_MONGODB_ADMINPASSWORD=password \  
4 --net=mongo-network \  
5 --name mongo-express \  
6 -e ME_CONFIG_MONGODB_SERVER=mongodb \  
7 mongo-express
```

ATTENZIONE: il valore di

```
1 ME_CONFIG_MONGODB_SERVER
```

deve essere uguale al nome del container di mongodb avviato al punto prima.

4.4 Comunicazione tra Mongo-Express e localhost

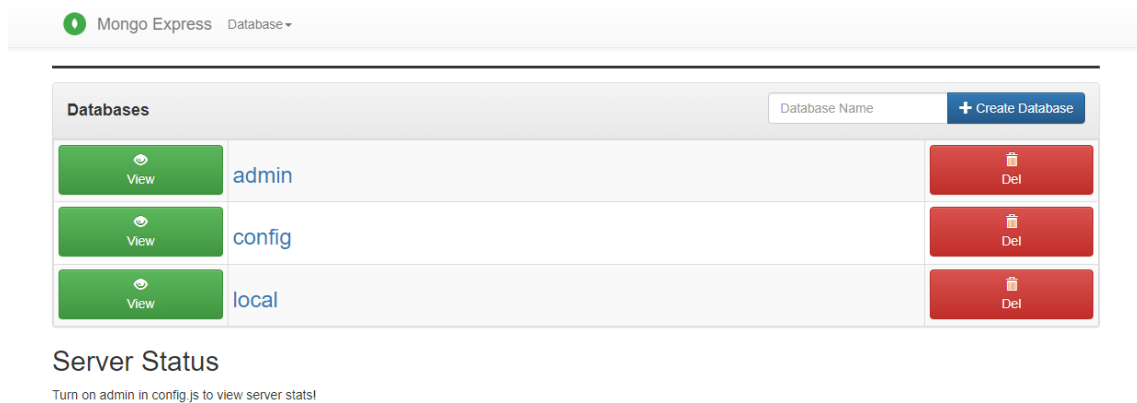
Guardando i log relativi a quest'ultimo container possiamo vedere che Mongo Express è in ascolto sulla porta 8081:

```
C:\Users\Stryker>docker logs 678275957e7ce6e235bd4e27e72c6d3b54fb1794c9b8a9fea89bf81d36ebcd8b  
Welcome to mongo-express  
-----  
  
(node:7) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.  
Mongo Express server listening at http://0.0.0.0:8081  
Server is open to allow connections from anyone (0.0.0.0)  
basicAuth credentials are "admin:pass", it is recommended you change this in your config.js!  
  
C:\Users\Stryker>
```

Rechiamoci nel browser a proviamo ad accedere a mongo-express tramite la porta 8081 che abbiamo bindato durante il run del container. Scriviamo quindi:

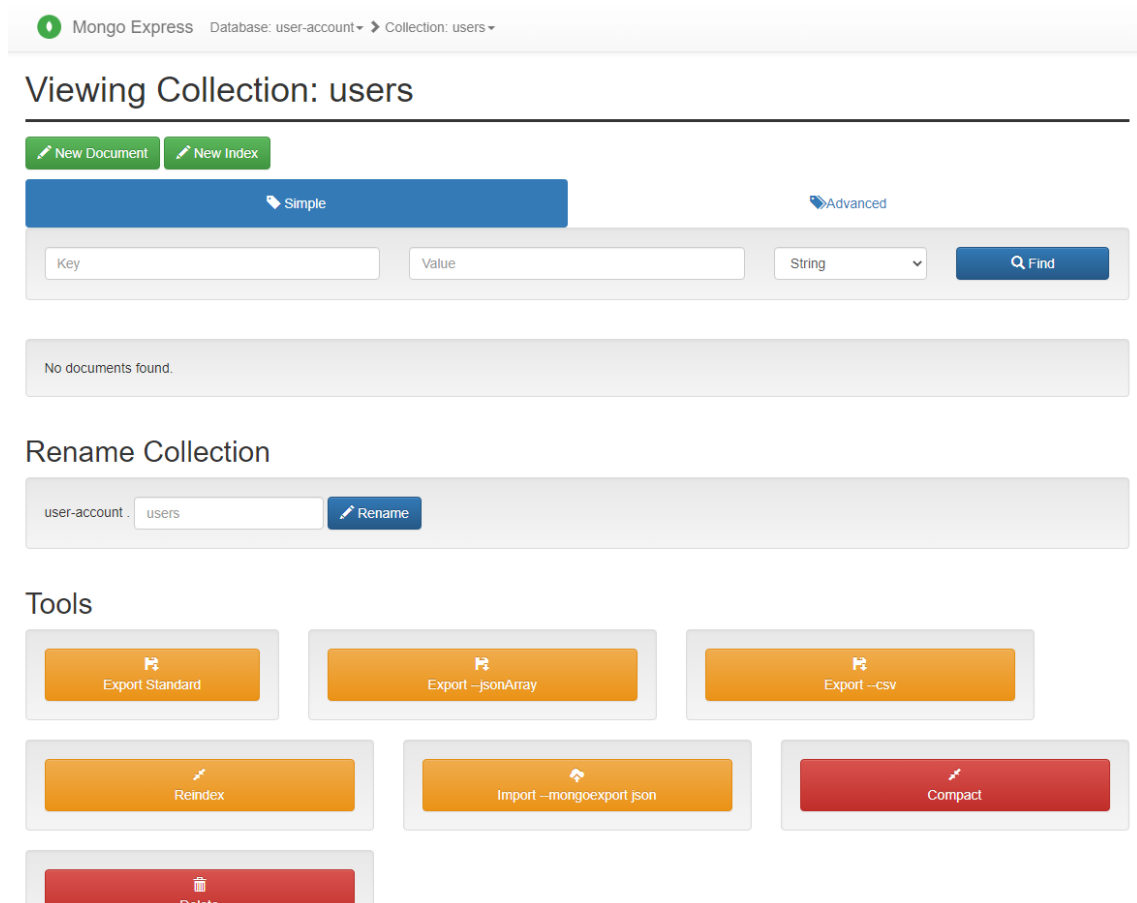
```
1 http://localhost:8081/
```

Il risultato è:



Questo ci permette di interfacciarci con mongodb e usare l'UI per creare e gestire il nostro database!

1. Creiamo quindi un database e chiamiamolo "user-account" usando il form in alto a destra.
2. Clicchiamo sul database appena creato e creiamo una collezione chiamata "users" sempre usando il form in alto a destra.



Congratulazioni!! Possiamo inserire, rimuovere, modificare utenti, creare indici, esportare e importare dati!

4.5 Collegarsi in modo programmatico al database

Questa sezione spiega brevemente come collegarsi al DB usando un server NodeJs express locale. La connessione puo essere fatta anche usando moltissime altre tecnologie come aws lambda, server OVH, server ec2 ecc. In tal caso bisognerà aprire le porte del proprio router per permettere la connessione.

In questa sezione ci collegheremo ed eseguiremo operazioni di get e post, per testare il nostro database in modo programmatico.

Supponiamo di volerci collegare al nostro database usando un applicazione express Nodejs, questi sono degli snapshot di codice di cui avremmo bisogno per interagire con il DB.

Le credenziali d'accesso sono:

```
1 let mongoUrl = "mongodb://admin:password@localhost:27017";
2 let databaseName = "user-account";
3
4 let express = require('express');
5 let app = express();
6 let MongoClient = require('mongodb').MongoClient;
7 let bodyParser = require('body-parser');
8 let path = require('path');
9 let fs = require('fs');
```

Se vogliamo eseguire un inserimento nel db:

```
1 app.post('/update-profile', function (req, res) {
2   let userObj = req.body;
3
4   MongoClient.connect(mongoUrl, function (err, client) {
5     if (err) throw err;
6
7     let db = client.db(databaseName);
8     userObj['userid'] = 1;
9
10    let myquery = { userid: 1 };
11    let newvalues = { $set: userObj };
12
13    db.collection("users").updateOne(myquery, newvalues, {upsert: true}, function(
14      err, res) {
15      if (err) throw err;
16      client.close();
17    });
18  });
19  res.send(userObj);
20 });
```

E in modo equivalente si possono fare richieste get per richiedere informazioni dalla nostra collezione:

```
1 app.get('/get-profile', function (req, res) {
2   let response = {};
3   MongoClient.connect(mongoUrlLocal, function (err, client) {
4     if (err) throw err;
5
6     let db = client.db(databaseName);
7
8     let myquery = { userid: 1 };
9
10    db.collection("users").findOne(myquery, function (err, result) {
11      if (err) throw err;
12      response = result;
13      client.close();
14
15      res.send(response ? response : {});
16    });
17  });
18 });
```

4.6 Docker Compose

Come abbiamo visto in precedenza i comandi di run possono essere molto lunghi e articolati, i comandi di run mostrati possono diventare ancora piu complicati in presenza di piu container e servizi che devono comunicare tra loro, e risulta difficile tenere traccia di tutti i comandi relativi e parametri per ogni container.

Docker Compose risolve tale problema attraverso la configurazione dei nostri container all interno di un file yaml, risparmiando all'utente di ripetere molte volte i comandi docker.

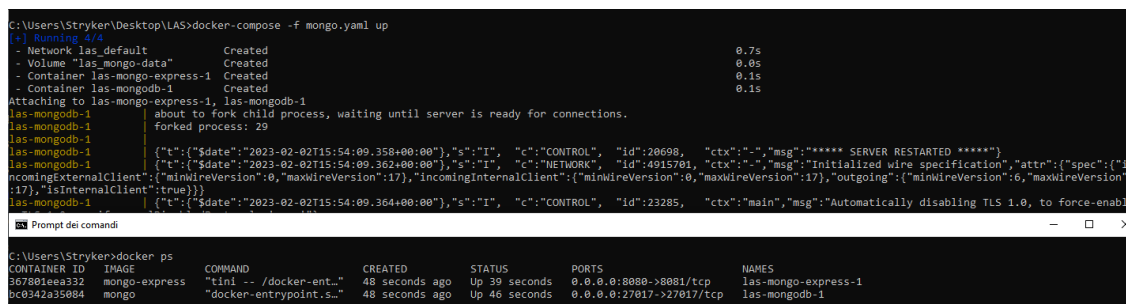
Un esempio di docker compose per il nostro workflow è questo:

```
1 version: '3'
2 services:
3   mongodb:
4     image: mongo
5     ports:
6       - 27017:27017
7     environment:
8       - MONGO_INITDB_ROOT_USERNAME=admin
9       - MONGO_INITDB_ROOT_PASSWORD=password
10    volumes:
11      - mongo-data:/data/db
12  mongo-express:
13    image: mongo-express
14    restart: always
15    ports:
16      - 8080:8081
17    environment:
18      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
19      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
20      - ME_CONFIG_MONGODB_SERVER=mongodb
21 volumes:
22   mongo-data:
23     driver: local
```

Come si puo notare non viene configurata la sottorete comune a entrambi i container, questo è perche Docker Compose lo fa in automatico per noi!






























Per eseguire un file docker-compose.yaml dobbiamo digitare il comando:

```
1 docker-compose -f <nome-file>.yaml up
```



```
C:\Users\Stryker\Desktop\LAS>docker-compose -f mongo.yaml up
[*] Running 4/4
- Network las_default Created 0.7s
- Volume "las-mongo-data" Created 0.0s
- Container las-mongo-express-1 Created 0.1s
- Container las-mongodb-1 Created 0.1s
Attaching to las-mongo-express-1, las-mongodb-1
las-mongo-express-1 | about to fork child process, waiting until server is ready for connections.
las-mongo-express-1 | forked process: 29
las-mongo-express-1 | {"t":{"$date":"2023-02-02T15:54:09.358+00:00"},"s":"I", "c":"CONTROL", "id":20698, "ctx":"-", "msg":"***** SERVER RESTARTED *****"}
las-mongo-express-1 | {"t":{"$date":"2023-02-02T15:54:09.362+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"-", "msg":"Initialized wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":17},"isInternalClient":true}}}}
las-mongo-express-1 | {"t":{"$date":"2023-02-02T15:54:09.364+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main", "msg":"Automatically disabling TLS 1.0, to force-enabl
```

Il risultato è identico a quello di prima, eseguito tramite comandi run indipendenti.

	Name	Image	Status	Port(s)	Started	Actions
<input checked="" type="checkbox"/>	 mongodb 22e0f3deda32 	mongo	Exited	27017:27017 		  
<input checked="" type="checkbox"/>	 mongo-express 678275957e7c 	mongo-express	Exited (143)	8081:8081 		  
<input type="checkbox"/>	 las 367801eea332 	-	Running (2/2)			  
<input type="checkbox"/>	 mongo-express-1 367801eea332 	mongo-express	Running	8080:8081 	4 minutes ago	  
<input type="checkbox"/>	 mongodb-1 bc0342a35084 	mongo	Running	27017:27017 	4 minutes ago	  

Per interrompere i container relativi a un file `docker-compose.yaml` dobbiamo digitare il comando:

Tale comando inoltre rimuoverà la rete comune ai due container creata automaticamente con il comando "UP".

4.7 Dockerfile

In questa ultima sezione proseguiremo la nostra demo creando un immagine docker. In questo modo potremo inserire la nostra applicazione all'interno di un docker container.

Per costruire un docker image a partire da un app, dobbiamo copiare il contenuto del applicazione dentro il dockerfile, che è un sistema a "template" che permette di creare immagini.

```

1 #Immagine d'appoggio, abbiamo bisogno di node
2 FROM node:13-alpine
3
4 #variabili d'ambiente
5 ENV MONGO_DB_USERNAME=admin \
6     MONGO_DB_PWD=password
7
8 #creiamo la cartella /home/app
9 RUN mkdir -p /home/app
10
11 #copiamo la nostra cartella relativa all'applicazione NodeJs con la quale in
12   precedenza ci siamo interfacciati al DB in modo programmatico all'interno del
13   container dentro: /home/app, questo e' solo un esempio, si puo usare qualsiasi
14   tipo di applicazione che faccia da backend (Java Spring, Python, Flask ecc)
15 COPY ./app /home/app
16
17 # imposta la cartella di default in modo che il prossimo comando sia eseguito
18   dentro: /home/app
19 WORKDIR /home/app
20
21 # eseguiamo npm install in /home/app grazie a WORKDIR
22 RUN npm install
23
24 # eseguiamo il comando "node server.js" dentro il container
25 CMD [ "node", "server.js" ]

```

E' importante che il nome del file al interno del quale inseriremo questo codice si chiami "Dockerfile".

Per eseguire la build del dockerfile e trasformarlo in un immagine dobbiamo digitare il comando:

```
1 docker build -t my-app:1.0 .
```

dove . sta per "cerca il dockerfile dentro la cartella corrente".

```
Microsoft Windows [Versione 10.0.19044.2466]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\Stryker\Desktop\LAS>progetto LASdocker build -t my-app:1.0 .

[+] Building 11.1s (10/10) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 494B
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [internal] load metadata for docker.io/library/node:13-alpine
[1/5] FROM docker.io/library/node:13-alpine#sha256:527c70f7481f7fef6b58558bc28de33459178ab72421f1fb7b63a281ab670258
-> resolve docker.io/library/node:13-alpine#sha256:527c70f7481f7fef6b58558bc28de33459178ab72421f1fb7b63a281ab670258
-> sha256:cdbde7a5bc2a134ca9ec91be5856ce97083738d6f1d835a1d20224dfca08_2.810kB / 2.81MB
-> sha256:780378ba4b1427477000ea5b1477bb7823376cabdded2103121852f5_35.30MB / 35.30MB
-> sha256:5d74bf112a7d1334af9f679cc9b75062377464235fac36e18952aa2b125d5d2_2.24MB / 2.24MB
-> sha256:527c70f7481f7ef6b58558bc28de33459178ab72421f1fb7b63a281ab670258_1.656kB / 1.656kB
-> sha256:4201b3cf8fd14b747600583f0dedcc9c8321383f84d3fedbd30b1148653_1.16kB / 1.16kB
-> sha256:82db6bf4583a5422edd579a3b77f66aFe1274Ac4bSeabcfe08591786b368_6.77kB / 6.77kB
-> extracting sha256:cdbde7a5bc2a134ca9ec91be5856ce97083738d6f1d835a1d20224dfca08_2.810kB / 2.81MB
-> sha256:40953642a1761567e57edd59ae9052573fc0e18c5458733a4093c23e1daf1ae_303B / 303B
-> sha256:780378ba4b1427477000ea5b1477bb7823376cabdded2103121852f5_35.30MB / 35.30MB
-> extracting sha256:5d74bf112a7d1334af9f679cc9b75062377464235fac36e18952aa2b125d5d2_2.24MB / 2.24MB
-> sha256:40953642a1761567e57edd59ae9052573fc0e18c5458733a4093c23e1daf1ae_303B / 303B
-> [internal] load build context
-> => transferring context: 6.09MB
[2/5] WORKDIR /home/app
[3/5] COPY --app /home/app
[4/5] RUN apt-get update
[5/5] RUN npm install
-> exporting to image
-> exporting layers
-> writing image sha256:d2d726e5Scb5a9b88d53ac97CB0ba847f80cF74f1E96c29a8S2b5b75e0
-> naming to docker.io/my-app:1.0
```

E ora per vedere la lista di immagini compresa la nuova, eseguiamo il comando "docker images":

```
C:\Users\Stryker\Desktop\LAS\progetto LAS>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-app	1.0	d2d726e55cb5	About a minute ago	126MB
mongo	latest	edacb1e1210b	21 hours ago	644MB
cdk-290503ffb826c854d1f8526e81eb293646e816c3d9afb442b51fb7206c417a4a	latest	c5855db5fd3a	5 days ago	1.72GB
mongo-express	latest	2d2fb2cab8f	15 months ago	136MB

Creare un container a partire da un applicazione permette di creare moltissime strutture, come ad esempio server rest collegato a un database, che comunica con un front-end situato sulla macchina locale.

Quello che è appena stato fatto, in un normale ambiente di lavoro viene eseguito da jenkins, che permette, a partire da un commit, di prendere il Dockerfile, costruire un immagine in base alle ultime modifiche del nostro progetto appena committate, e pusha questa immagine dentro un repository di immagini, dal quale gli sviluppatori possono scaricarle e usarle a loro piacere per dev o test. Oppure tale immagine può essere deployata direttamente in un ambiente di produzione.

Bisogna ricordarsi che ogni volta che un Dockerfile viene modificato, bisogna ri-buildare l'immagine.

5 Conclusione

Ora che sappiamo come creare la nostra immagine, scaricare immagini e tirare su container che comunicano tra di loro e fare comunicare container con l'ambiente esterno, abbiamo la base necessaria per sbizzarrirci nella creazione di infrastrutture usando docker.

Grazie per la lettura.

-David Ambros 02/2023

-Matricola 881443