Percorsi, installazioni e ambienti virtuali in python

Python Venv ovvero creare un progetto python al quale legare l'interprete, l'installatore e le sue librerie.

Partiamo da una domanda

- Abbiamo installato python
- Apriamo il terminale
- Lanciamo python –V e il terminale ci stampa a schermo la versione di python
- A questo punto facciamoci una domanda: cosa è successo veramente?
- Per capire la domanda scriviamo da prompt:

% pythonn –V

zsh: command not found: pythonn

Cosa sono i comandi? Perchè il sistema operativo trova python ma non trova pythonn?

Partiamo da una domanda

python è un file eseguibile. Ma dove è posizionato questo file, in quale folder del filesystem?

A questa domanda può rispondere il comando di Linux which. Da prompt di comandi

% which python

/usr/bin/python

La domanda allora diventa: perché il terminale va a pescare un file proprio da quella cartella? E se la volessimo cambiare?

Le variabili d'ambiente

Una variabile, così come abbiamo imparato nei linguaggi di programmazione, è un contenitore che ha un **nome** (es. var A), un **contenuto** (es. var A=3) e un **tipo** (A=3 ci dice che A è di tipo intero).

C'è un ulteriore caratteristica di una variabile, che è la sua visibilità o **scope**. Nei linguaggi di programmazione si parla di variabile globali oppure di variabili la cui visibilità è ristretta alla procedura che la crea.

Le variabili d'ambiente

Una variabile d'ambiente è una variabile il cui scope e' relativo

- al computer tutto e/o
- all'utente corrente e/o
- all'istanza corrente del terminale e/o

Le variabili d'ambiente

Le variabili d'ambiente esistono in tutti i sistemi operativi e sono molto simili. Da una finestra di terminale sotto Linux/MAC

%**env**

Fa una lista delle variabili d'ambiente associate al PC e all'utente corrente

%myvar=pippo

Crea la variabile d'ambiente myvar e gli assegna valore pippo.

%echo $myvar

Stampa a schermo il valore della variabile myvar

Le variabili d'ambiente

Le stesse operazioni possono essere fatte sotto MAC e, con comandi simili, sotto Windows.

La variabili d'ambiente possono essere usate negli script da terminale.

Le variabili d'ambiente possono essere usate nei pacchetti d'istallazione.

Le variabili d'ambiente possono essere usate dai programmi, es. in un nostro programma scritto in python possiamo usare le variabili d'ambiente per conoscere il nome dell'utente corrente.

Linux: dove sono definite le variabili d'ambiente

La definizione delle variabili d'ambiente avviene:

- nel file .bashrc che si trova nella cartella associata all'utente corrente.

- nei file .env che stanno nella root di un progetto. In tali file il formato di salvataggio è
  NOMEVAR=VALORE

Linux: cos’è e come funziona la variabile PATH

- Quando digitate il nome di un programma sul vostro terminale per richiederne l’esecuzione, di fatto state imponendo alla shell di cercare, in alcune directory ben precise, un file eseguibile con il nome da voi indicato.
- Ma quali sono queste directory?
- La prima è la directory corrente.
- Le altre sono indicate in una variabile d'ambiente. La variabile d'ambiente PATH.

Linux: cos'è e come funziona la variabile PATH

Facciamo un esempio. Posizioniamoci in una cartella vuota e digitiamo il cmd

%python –V

Python 2.7.16 ←

%echo $PATH

/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/andrea/Progetti/LibreriePython:/opt/X11/bin:/Library/Frameworks/Mono.framework/Versions/Current/Commands:/Applications/Wireshark.app/Contents/MacOS

In uno dei path indicati dalla variabile d'ambiente.

Python, pip, modules

python è il file eseguibile che interpreta il codice python e lo esegue.

Per alcuni usi particolari dobbiamo installare dei moduli. Questo lo facciamo con pip.

Poniamoci una domanda: data un'istanza di python quali sono i moduli attualmente installati ed ai quali l'istanza punta?

Python, pip, modules

Uno dei metodi è quello di digitare

pip list -v

A schermo compare la lista dei moduli installati e la loro directory.

Python, pip, modules

Possiamo avere installate più versioni di python. Queste le troviamo nelle cartelle:

/usr/bin

/usr/local/bin

Ogni versione di python ha la sua versione di pip che si trova nella stessa directory. Ed ogni versione di python può puntare ad una diversa cartella con i moduli installati.

Python, pip, modules: gestire il tutto a livello di progetto

Con il **virtual environment** possiamo gestire la versione di python e di pip ed i moduli associati a livello di singolo progetto.

Tutto ciò è utile e ordinato anche perché spesso i moduli non sono indipendenti fra loro e l'installazione di un modulo può causare problemi per il funzionamento di un altro modulo.

Il virtual environment

Supponiamo di dover scrivere un programma in python che invia mail.
Creiamo allora la cartella InviaMail (il nostro progetto) e li inseriamo il nostro codice python.

% pip list

ci darà i moduli installati nella directory di riferimento per la nostra installazione di python.

Vogliamo creare una cartella dove metteremo tutti i moduli necessari al nostro progetto e vogliamo che l'eseguibile python vada a pescare li i suoi moduli.

Il primo passo è installare il modulo virtualenv

% pip install virtualenv

Il virtual environment

Creiamo la cartella myenv che sarà il folder dove installeremo tutti i moduli ai quali punterà il nostro progetto:

```
% virtualenv myenv
% source myenv/bin/activate
```

Con gli ultimi 2 comandi lavoro nel myenv. Se installo un package lo installo li.

Uso i moduli che stanno li. Per provarlo, se ripetiamo

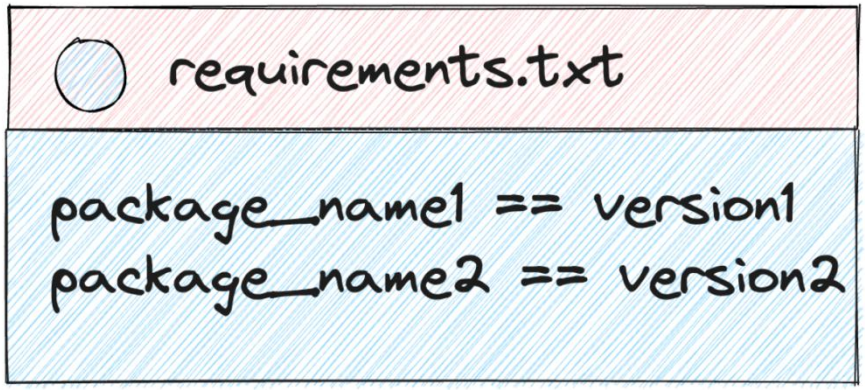
```
% pip list
```

la lista dei moduli installati è lunga 0, è vuota, visto che ancora non ho installato nulla per il mio progetto.

Il file requirements.txt

Un progetto ha associati i moduli/packages/librerie necessari che in parte sono inclusi nel pacchetto d'installazione di python ed in parte devono essere installati esplicitamente (**moduli aggiuntivi**).

Nel momento in cui dobbiamo distribuire il nostro progetto, piuttosto che fornire i moduli aggiuntivi possiamo fornire un file chiamato requirements.txt.



Questo file ha un formato preciso, definito dalla figura accanto e contiene la lista dei moduli aggiuntivi necessari al nostro progetto.

Il file requirements.txt

Dato un venv, nel momento in cui dobbiamo distribuire il nostro progetto dobbiamo creare il file requirements.txt. Il comando per farlo è:

```
$ pip freeze > requirements.txt
```

Colui che deve usare il nostro progetto riceverà i file python da noi sviluppati ed il file requirements.txt con la lista dei moduli aggiuntivi necessari. Per installare tali moduli deve eseguire il comando:

```
$ pip install -r requirements.txt
```

Docker, i containers, un filesystem costruito per l'applicazione: creare un ambiente per un'applicazione.

Il contesto di un'applicazione

Un programma python definisce un'APP che deve essere eseguita su un PC. Il PC di esecuzione interagisce con l'APP in due versi:

- **in input:** l'APP al momento dell'avvio e durante il suo ciclo di vita eredità lo stato del PC host, e quindi il suo filesystem, la sua configurazione di rete ed in generale tutte le sue caratteristiche
- **in output:** a sua volta l'APP esegue delle operazioni e quindi modifica lo stato del PC stesso. Per esempio salva alcuni dati su un file, consumo la RAM a disposizione del processo.

Il contesto di un'applicazione

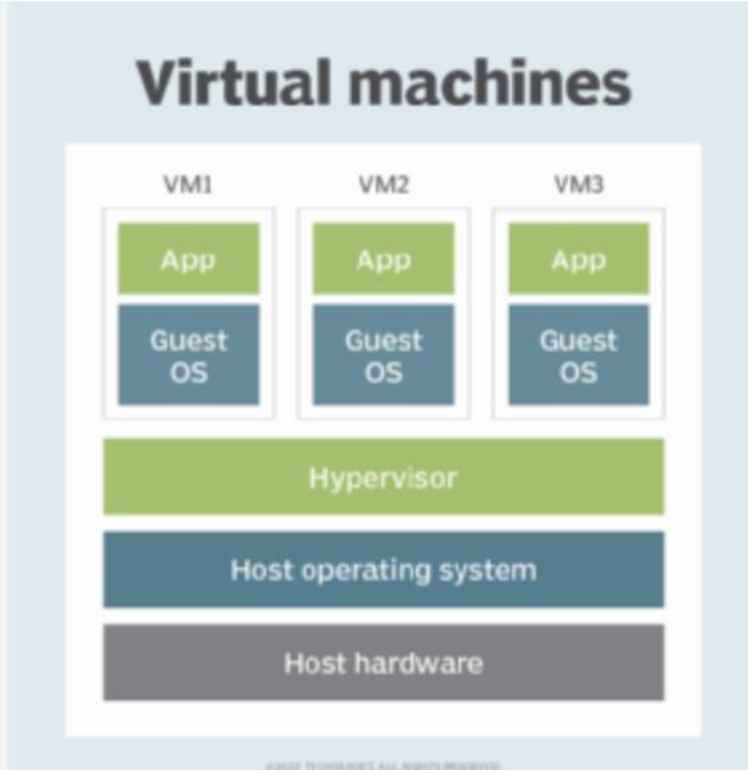
In alcuni casi, che dipendono dagli scopi della nostra APP, tali interazioni (sia in un senso che nell'altro) sono necessarie.

Es. dobbiamo costruire un'APP che processa in tempo reale il file di log di un server.

In altri casi l'APP non necessità di tali interazioni.

Esistono poi delle situazioni intermedie: per esempio dobbiamo interagire con il filesystem del PC host ma non con altre componenti.

Le macchine virtuali



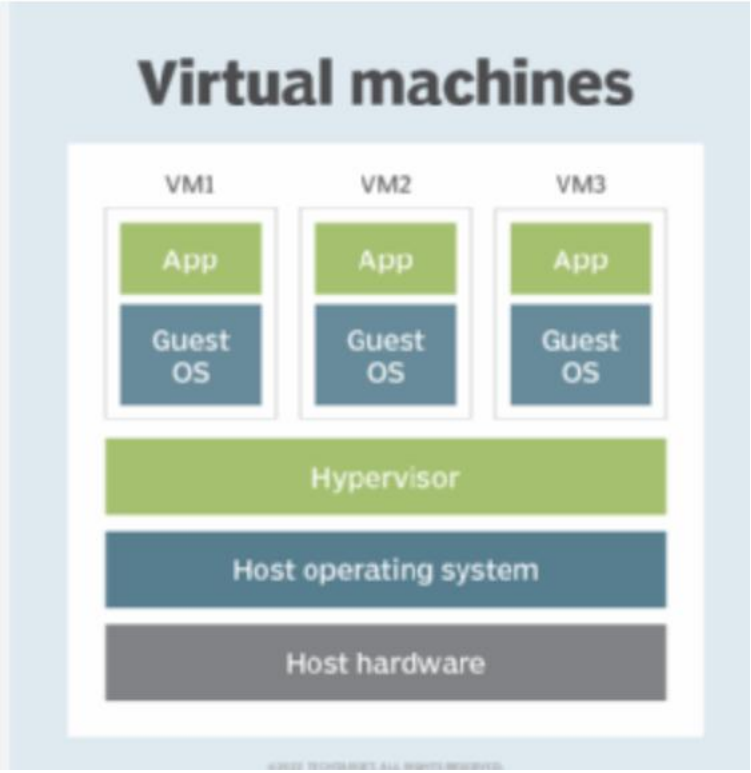
Un modo relativamente semplice di affrontare il problema è quello di installare sul nostro PC una virtual machine.

Il primo passo è installare un software che gestisce le VM.

Tale software ci permette di costruire una VM "personalizzata", rispetto a tutte le caratteristiche elencate.

Per esempio possiamo costruire una VM che ha una interfaccia di rete che è una scheda ethernet virtuale ma poi il PC host è connesso tramite Wifi.

Le macchine virtuali

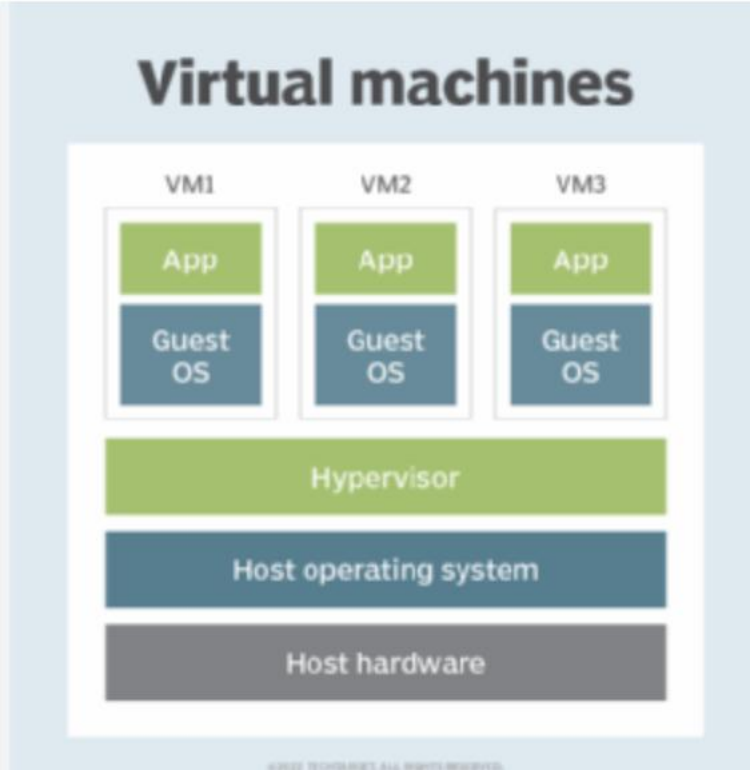


Per ospitare VMs in un computer è necessario uno specifico tipo di software chiamato **hypervisor**.

Il compito di un hypervisor è quello di gestire le risorse del PC host durante la creazione e durante il funzionamento delle VMs presenti su tale PC.

L'hypervisor consente anche il riallocamento delle risorse in funzione del carico di lavoro che in un certo momento sta sostenendo ciascuna specifica VM.

Le macchine virtuali

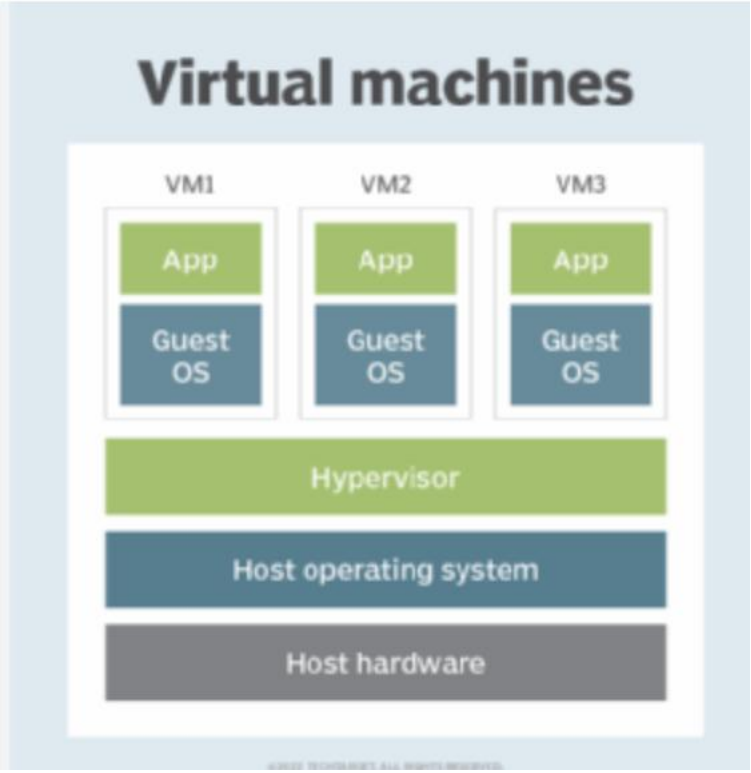


Una VM definisce un ambiente isolato rispetto al PC host, che ha un suo filesystem e un sistema di drivers virtuali.

In questo sistema isolato possiamo far girare la nostra APP.

In questo modo siamo sicuri che applicazioni esterne non interferiscano negativamente con la nostra APP. Al tempo stesso la nostra APP non è in grado di danneggiare altri elementi del PC.

Le macchine virtuali



Il principale inconveniente di una VM è la sua pesantezza. Le VM occupano centinaia di giga ed in generale creiamo tutta una serie di componenti che potrebbero non riguardare affatto la nostra APP.

Anche dal punto di vista della distribuzione di un'APP creare un installer che include una VM può essere funzionale da un punto di vista applicativo ma è molto pesante quantitativamente in molte fasi del processo di distribuzione dell'APP.

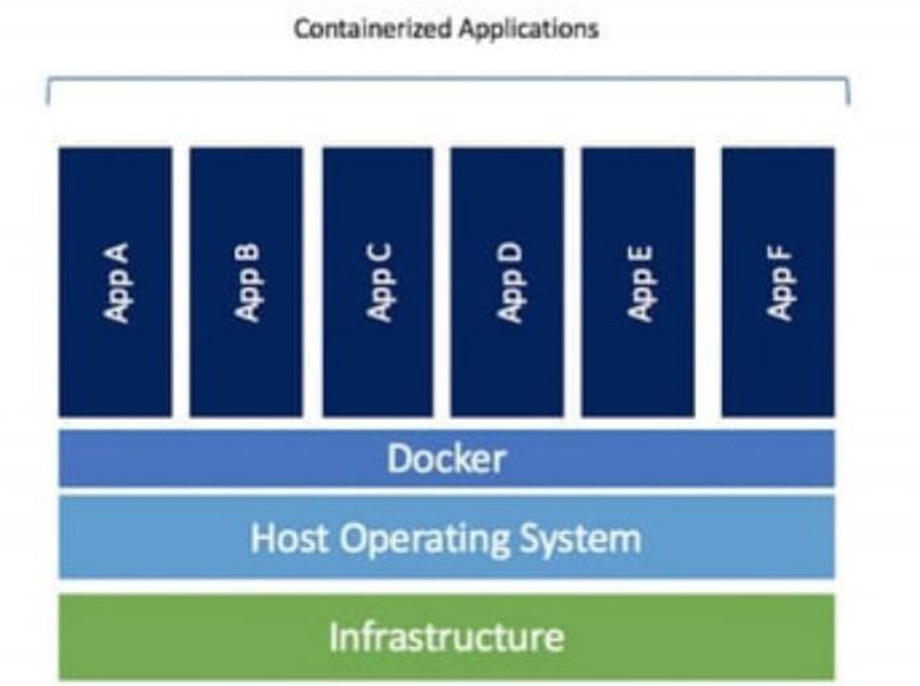
I containers

Mentre una VM punta a riprodurre un intero PC all'interno del quale può girare la nostra APP il container è un concetto diverso.

Esso parte dall'APP stessa e punta a costruire un ambiente virtuale con tutte e sole le risorse necessarie all'APP.

Il risultato è un nuovo processo, che da una parte definisce un ambiente virtuale per la nostra APP ma, al tempo stesso, gira nel kernel del PC host, senza replicare parti dello stesso.

Docker



Docker è un software development tool ed una tecnologia di virtualizzazione il cui scopo è quello di semplificare lo sviluppo, la distribuzione e la gestione di applicazioni.

Docker è un gestore di containers.

Installiamo Docker Desktop

Facciamo un piccolo esercizio

1) Da VSCode creiamo la nostra cartella/progetto e dentro mettiamoci un file **myapp.py** con semplicemente una print.
2) Il secondo passo è quello di costruire il container per la nostra applicazione. Creiamo nella stessa cartella di progetto un file che chiamiamo Dockerfile.

Facciamo un piccolo esercizio

3) Avvio Docker Desktop

4) Preparo il Dockerfile

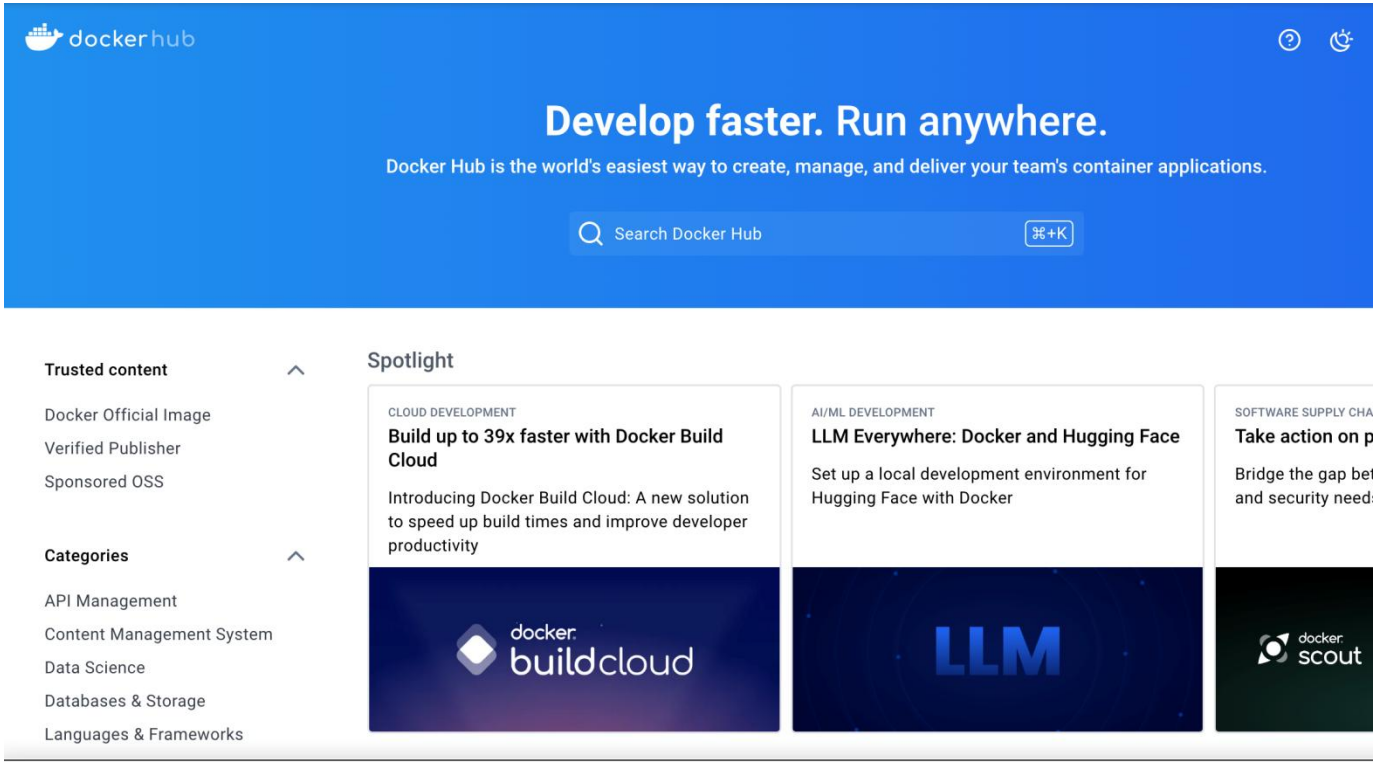
Il Dockefile è un file di testo con l'insieme di istruzioni necessarie a creare il container della nostra semplice applicazione **myapp.py**.

Il Dockerfile

```
FROM python:3.10-alpine
ADD myapp.py .
CMD ["python","./myapp.py"]
```

FROM: ogni Dockerfile in genere inizia con l'istruzione FROM. Questa istruzione specifica l’immagine base da cui partire per costruirne una di nuova. In questo caso stiamo usando l'immagine python . Il tag 3.10-alpine specifica quali tra le diverse immagini python utilizzare. L'idea è di definire una mini macchina virtuale pensata in modo essenziale per la nostra applicazione.

Hub.docker.com



E' il repository di file image per docker. Un motore di ricerca ci aiuta a cercare l'immagine più adatta per la nostra app.

Il Dockerfile: ADD

```
FROM python:3.10-alpine
ADD myapp.py .
CMD ["python","./myapp.py"]
```

ADD: è usato per copiare file o directory all'interno di una docker image. La sintassi prevede 2 parametri:

ADD source destination

Il source può essere un file oppure una URL.

Il Dockerfile: CMD

```
FROM python:3.10-alpine
ADD myapp.py .
CMD ["python","./myapp.py"]
```

CMD: è usato per specificare quale comando deve eseguire il container quando viene lanciato.

Nel nostro esempio il comando è "python" con il parametro "./myapp.py".

Facciamo un piccolo esercizio

5) Scrivo:

docker build -t myapp-container .

La build serve per costruire il container. Il puntino serve per dire dove prendere i file. Il dockefile di default è Dockerfile nella directory indicata dal puntino. Altrimenti usare –f per specificare un differente Dockerfile.

6) **docker** run myapp-container

Alcuni comandi

$ docker ps

Fornisce la lista dei container in esecuzione ed i loro **id**

$ docker stop id

Interrompe l'esecuzione di un container

Facciamo un piccolo esercizio

E' possibile lanciare un container scrivendo da prompt il comando e quindi sovrascrivendo il CMD del Dockerfile.

Es.

docker run myappcont python myapp.py

Il risultato è lo stesso. Se nel Dockerfile ADD diversi file python (es. app1.py app2.py) possiamo poi decidere quale app lanciare.

Docker compose

Docker Compose è uno strumento per costruire ed eseguire applicazioni multi-container. Ambienti complessi costituiti per esempio da un server web e un database possono essere ridefiniti come multi-container applications.

Docker Compose semplifica il controllo dell'applicazione distribuita, facilitando la gestione dei servizi, delle reti, dei volumi utilizzando un unico YAML configuration file. Scritto il configuration file, con un unico comando è possibile creare ed avviare tutti I servizi e quindi avviare il sistema distribuito definito nel configuration file.

Docker Compose può essere utilizzato nella fase di sviluppo, nella fase di test e di deployment di un'applicazione distribuita.

Docker Compose dispone dei comandi per gestire l'intero ciclo di vita dell'applicazione distribuita:

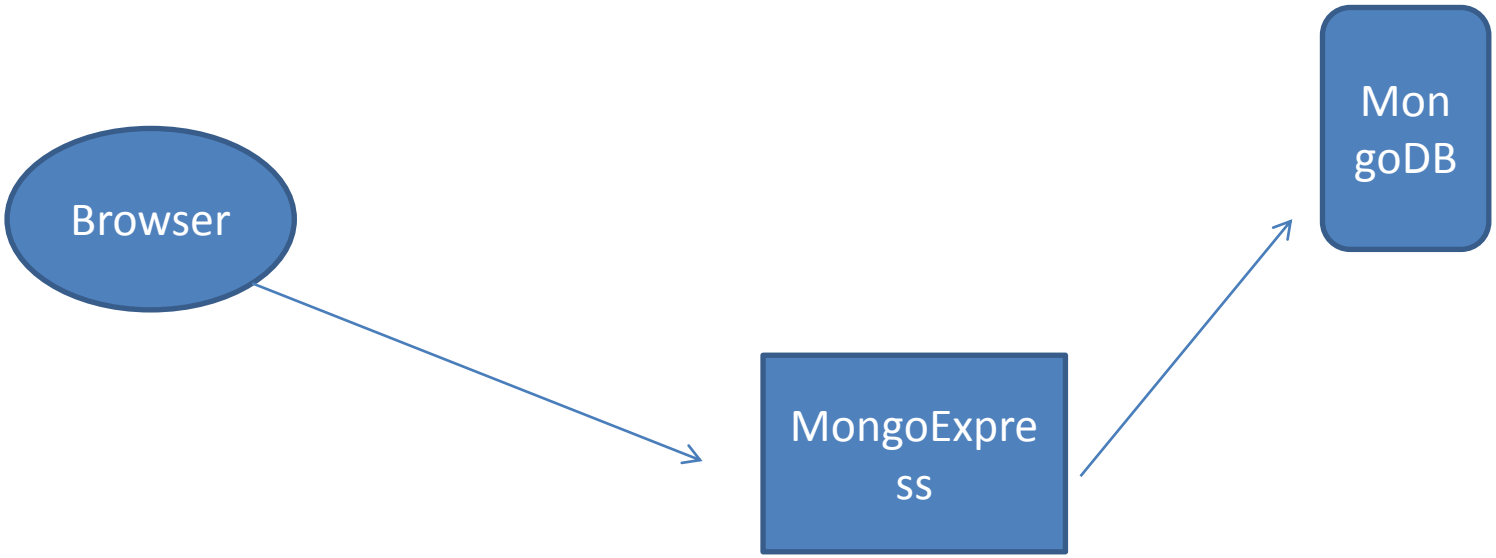
- start, stop and rebuild services
- view the status of running services
- stream the log output of running services
- run a one-off command on a service

Facciamo un esercizio

https://www.deso.tech/cose-il-docker-compose-un-esempio-pratico/

Facciamo un esercizio

Mettiamo su un'architettura client/server dove il server è costituito dal DB non relazionale MongoDB e il client è un'interfaccia web per la gestione del DB stesso. Questa interfaccia si chiama Mongo Express.



Facciamo un esercizio

Data l'architettura della slide precedente realizziamo il tutto attraverso due docker container, il primo che contiene il database MongoDB ed il secondo che contiene Mongo Express.

I due container definiscono due applicazioni che devono dialogare tra loro. Per permettere questo usiamo docker compose.

Il primo passo è creare una cartella di progetto e dentro la stessa creare un file che chiamiamo docker-compose.yml.

- La prima cosa da fare è specificare la versione di Docker Compose che intendiamo utilizzare, usiamo la sintassi *version: '3'* per usare la versione 3 del formato Docker Compose.
- Successivamente creiamo la sezione "services:" in cui dichiarare i container che vogliamo creare, in questo caso useremo due container: "*mongo*" e "*mongo-express*".

Docker-compose.yml

```
version: '3'

services:
        mongo:

        mongo-express:
```

Il container mongo

Nel servizio ***mongo*** specifichiamo l'immagine *mongo* da utilizzare per il container.

Usando poi la coppia chiave/valore

restart: always

possiamo assicurarci che il container verrà riavviato automaticamente in caso di errore.

Inoltre, definiamo un volume chiamato *localdatabase* che verrà montato nella directory */data/db* del container per persistere i dati del db.

Infine, associamo il container alla rete *composenetwork* che definiremo in un'altra sezione del file.

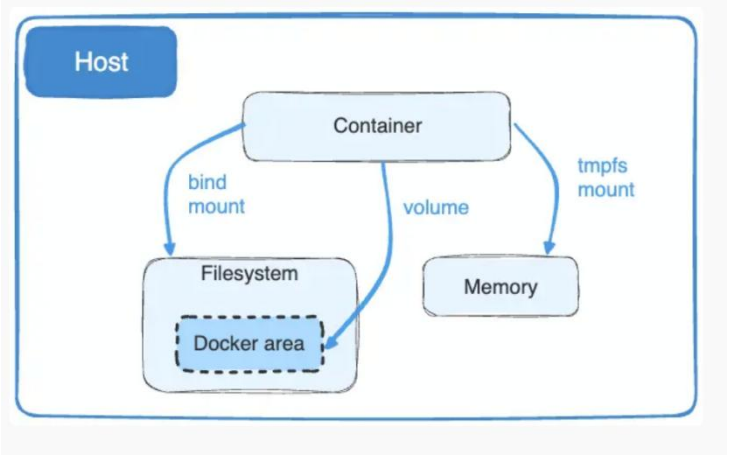
Docker-compose.yml

```
version: '3'

services:
        mongo:
            image: mongo
            restart: always
            volumes:
            - localdatabase:/data/db
            networks:
            - composenetwork

mongo-express:

volumes:
localdatabase:

networks:
composenetwork:
driver: bridge
```

Volumes

Le applicazioni che vogliamo eseguire con i docker container possono aver bisogno di dati persistenti. In questo caso dobbiamo associare al container un volume.

I **Volumes** sono dei meccanismi che ci permettono di gestire dati persistenti all'esterno del container. Tutti I volumes sono gestiti da Docker e piazzati in una directory dedicata del computer host, usualmente /var/lib/docker/volumes per Linux systems.



Un container può avere una memoria persistente associata oppure una memoria temporanea.

Volumes

Possiamo associare dello spazio su disco (volume) ad un container in tre modi:

- specificando il volume quando facciamo docker run
- specificando il volume nel Dockerfile
- specificando il volume nel file yml in caso di uso di docker-compose

Volumes

Nell'esempio, con la chiave volumes in fondo al file

volumes:

    localdatabase:

dichiariamo il volume con nome localdatabase. Quando invece nel container mongo scriviamo

volumes:

    - localdatabase:/data/db

stiamo mappando il volume localdatabase nella cartella /data/db del filesystem del container. Questo significa che quello che viene scritto da MongoDB nella cartella /data/db è persistente e finisce nel volume localdatabase.

Il container mongo.express

Per il servizio *mongo-express*:

- specifichiamo l'immagine *mongo-express* da usare per il container
- mappiamo la porta locale 8081 alla porta 8081 del container per consentire l'accesso all'interfaccia web di Mongo Express
- impostiamo le variabili d'ambiente "*ME_CONFIG_MONGODB_SERVER*" e "*ME_CONFIG_MONGODB_PORT*" per indicare a Mongo Express di connettersi al servizio *mongo* sulla porta predefinita di MongoDB.
- ci assicuriamo che il servizio sia avviato dopo *mongo* usando la sezione *depends_on*.

Accanto le voci del servizio mongo-express nel file di docker-compose.

```
mongo-express:
        image: mongo-express
        restart: always
        ports:
                - 8081:8081
        environment:
- ME_CONFIG_MONGODB_SERVER=mongo
- ME_CONFIG_MONGODB_PORT=27017
        networks:
                - composenetwork
        depends_on:
                - mongo
```

Il file yml completo

```yaml
version: '3'

services:
  mongo:
    image: mongo
    restart: always
    volumes:
      - localdatabase:/data/db
    networks:
      - composenetwork

  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8081:8081
    environment:
      - ME_CONFIG_MONGODB_SERVER=mongo
      - ME_CONFIG_MONGODB_PORT=27017
    networks:
      - composenetwork
    depends_on:
      - mongo

volumes:
  localdatabase:

networks:
  composenetwork:
    driver: bridge
```

Avviamo l'insieme dei container

Pronto il file di configurazione possiamo avviare tutti i container usando il comando

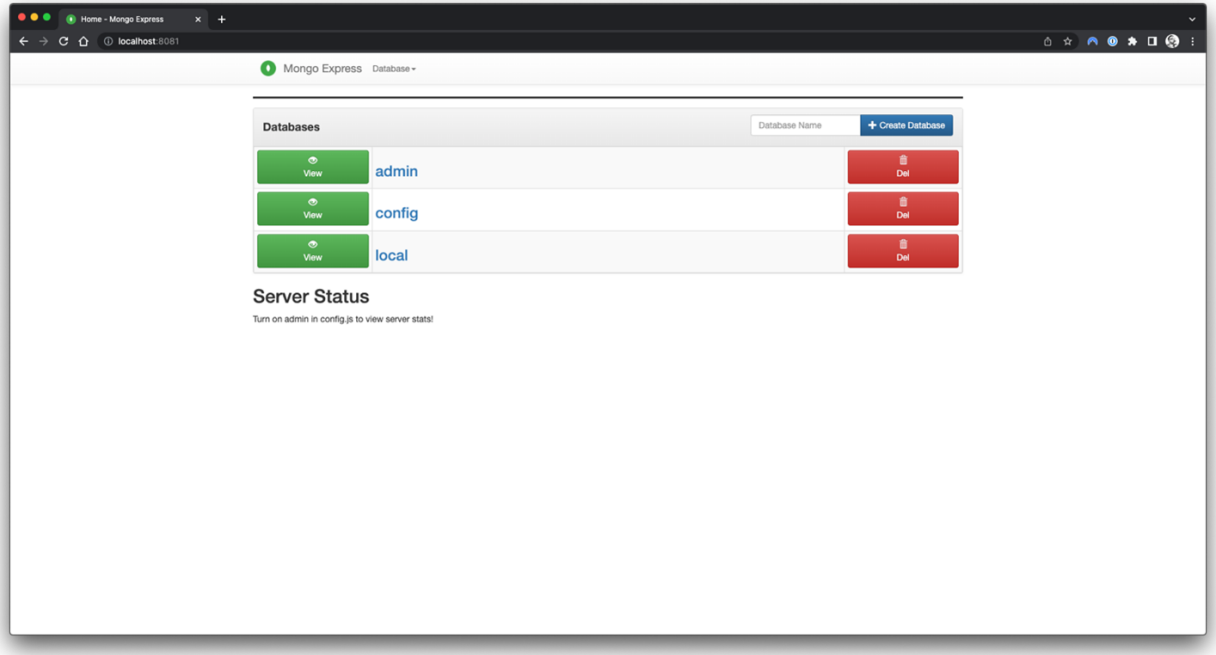
$ docker compose up -d

Questo comando leggerà il contenuto del file e avvierà i container in modalità detached, vale a dire in background senza che il terminale si blocchi, e così facendo il controllo viene restituito al terminale in modo da poterlo usare per altre attività.

Docker Compose si occuperà automaticamente di scaricare le immagini dei container, creare i container e configurare reti e volumi.

Testiamo l'APP distribuita

A questo punto è possibile verificare il corretto funzionamento dell'applicazione aprendo il browser e visitando *localhost:8081*. Otterremo in questo modo l'interfaccia di Mongo Express.



Infine…..

Anche se il file Docker Compose può diventare più complesso e con più servizi, questo non incide sulla facilità di creazione e gestione dell'ambiente. Se vogliamo mettere in pausa l'ambiente possiamo semplicemente eseguire il comando

$ docker compose stop

Per riprendere basta dare il comando

$ docker compose start

Se poi abbiamo esigenza di fare pulizia oltre a stoppare il tutto possiamo eseguire il comando

$ docker compose down