Installare Docker

- Caveat: eventuali versioni più antiche non sono più supportate e vanno rimosse
- Seguendo l'ottima documentazione in https://docs.docker.com/ si installerà *docker-ce,* Docker Community Edition, garantita come "edizione" ufficiale e più recente, cf. https://docs.docker.com/install/ (sezione *About Docker CE*)
- Esiste anche l'edizione Docker EE (Enterprise edition), a pagamento, mentre la Community Edition è gratuita
- Altra guida (rapida e efficace): https://computingforgeeks.com/installing-docker-ce-ubuntu-debian-fedora-arch-centos/
- Fino a pag. 5, si illustra l'installazione su CentOS, versione free di Red Hat, e, un tempo, distro di "prima scelta" per Docker
- Al 2021, però, Docker gira egregiamente su Debian/Ubuntu e CentOS è *discontinued*; si consiglia quindi di saltare a pag. 6

Installazione e avvio di Docker-CE: in a nutshell

- Per altre distro (Ubuntu, Debian, ...) e per il cloud (AWS, Azure), seguire: https://docs.docker.com/install/
  - Per Ubuntu... **non** installare con *apt* prima di aver letto la guida sopra!
- Ottima sintesi, concisa ed efficace, per installare Docker CE:
  https://computingforgeeks.com/installing-docker-ce-ubuntu-debian-fedora-arch-centos/
- Installato Docker-CE, si può avviare (da root) il servizio Docker:

```
# systemctl start docker
# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor
preset: disabled)
   Active: active (running) since Fri 2018-06-01 03:28:10 UTC; 30s ago
```

- Il daemon Docker è, a volte, "delicato". Se risponde in modo strano o non risponde, sempre da super-user:

```
# systemctl restart docker  # e, se ancora non bastasse:
# killall dockerd           # e, se ancora non bastasse: killall -9 dockerd
```

Immagine/Container *Hello world!*

- Lanciandolo, verifichiamo il buon funzionamento di Docker (da utente comune, ma appartenente al gruppo *docker*):

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world          # immagine scaricata dall'hub di Docker
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

- Alcune *keyword*: Docker daemon, client, image, container

Elementi di base: CLI e daemon/servizio

Da riga di comando, come utente comune nel gruppo docker, si invoca il client (CLI) `docker`:

```
$ docker
Usage:   docker [OPTIONS] COMMAND
A self-sufficient runtime for containers
Options:
  -H, --host list        Daemon socket(s) to connect to
 ...
Management Commands:
  container   Manage containers
  ...
Commands:
  attach      Attach local standard input, output, and error streams to a running container
...
```

Verifichiamo ora la presenza del daemon `dockerd` e del servizio docker:

```
$ ps -A | grep dockerd
  367 ?        00:06:04 dockerd
```

```
$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-06-07 13:56:13 CEST; 1 day 18h ago
 Main PID: 367 (dockerd)
    Tasks: 24 (limit: 2370)
   Memory: 176.2M
   CGroup: /system.slice/docker.service
           ├─367 /usr/bin/dockerd -H fd://
           ├─375 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
```

Interazione di base

Come spiegato dall'output di *docker run hello-world*:

- il cliente CLI *docker* si rivolge al daemon/servizio/engine *dockerd*
- questo crea un container (dal template/immagine *hello-world*)
- il container esegue il codice dell'app incapsulata nel container
- il daemon *dockerd* media l'input/output tra il container in esecuzione e il client CLI *docker*, che a sua volta comunica con il terminale dell'utente
  - nel caso di *hello-world*, l'output generato dall'app/container va al daemon *dockerd*, che lo passa al client CLI *docker*, che lo emette sulla standard output dell'utente

La comunicazione daemon-client avviene via una socket del kernel:

```
$ ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0  7 giu 13.55 /var/run/docker.sock
```

Permessi e ownership implicano che solo gli utenti nel gruppo *docker* possono interagire direttamente col daemon (v. prossima slide)

Docker come utente non-root

Durante l'installazione, sarà stato definito il gruppo *docker*:

```
$ grep docker /etc/group
docker:x:999:user1,user2
```

Se l'utente corrente non è nel gruppo *docker*, si vedrà l'output:

```
$ groups | tr ' ' '\n' | grep docker
$
```

Se invece l'utente corrente è nel gruppo *docker*, si vedrà l'output:

```
$ groups | tr ' ' '\n' | grep docker
docker
$ sudo deluser $USER docker   # ora logout e login...
```

Qui sopra si è temporaneamente eliminato dal gruppo *docker* l'utente corrente, che non potrà più interagire col daemon/engine di Docker:

```
$ docker run hello-world
docker: permission denied trying to connect to Docker daemon socket at unix:///var/run/docker.sock...
```

Rimettiamo quindi l'utente corrente nel gruppo *docker*, con un comando tra:

```
$ sudo adduser $USER docker      # rimettiamo l'utente nel gruppo docker (logout e login di nuovo)
```

```
$ sudo usermod -aG docker $USER   # rimettiamo l'utente nel gruppo docker (logout e login di nuovo)
```

Immagine/container *Ubuntu*

```
$ docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
a48c500ed24e: Pull complete
1e1de00ff7e1: Pull complete
0330ca45a200: Pull complete
471db38bcfbf: Pull complete
0b4aba487617: Pull complete
Digest:
sha256:c8c275751219dadad8fa56b3ac41ca6cb22219ff117ca98fe82b42f24e1ba64e
Status: Downloaded newer image for ubuntu:latest
root@911b926d6c41:/#
```

- Da *man docker-run*:

  -i, --interactive, Keep container's STDIN open even if not attached

  -t, Allocate a pseudo-TTY. The default is false. When true, Docker can allocate a pseudo-tty and attach to the standard input of any container

- L'immagine per il container *Ubuntu* non esisteva localmente, quindi è stato scaricata da https://store.docker.com (o *hub.docker.com*)

Immagini e container

- L’immagine è un template (stampo) di container, da cui si possono generare più container in esecuzione nell’engine Docker
  - (cf. programma (file eseguibile) vs. processo (in esecuzione))
- Un’immagine ha una struttura interna stratificata (*layered*) che riflette la sua tipica creazione *bottom-up*, che parte da uno strato iniziale, esegue dei comandi di modifica dello stato di questo, aggiungendo quindi altri strati…
- Il sito microbadger consente di visualizzare la struttura dei container...

Immagini/Container: gestione da riga di comando

```
$ docker images    # immagini disponibili (scaricate prima dall'hub di Docker)
REPOSITORY          TAG                 IMAGE ID            CREATED
    SIZE
ubuntu              latest              452a96d81c30        4 weeks ago
    79.6MB
hello-world         latest              e38bc07ac18e        7 weeks ago
    1.85kB
$ docker ps        # container attivi
CONTAINER ID        IMAGE               COMMAND             CREATED
    STATUS
$ docker container ls      # come il precedente: container attivi
CONTAINER ID        IMAGE               COMMAND             CREATED
# ls /var/lib/docker/image/overlay2/imagedb/content/sha256/
452a96d81c30a1e426bc250428263ac9ca3f47c9bf086f876d11cb39cf57aeec
e38bc07ac18ee64e6d59cf2eafcdddf9cec2364dfe129fe0af75f1b0194e0c96
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
    SIZE
ubuntu              latest              452a96d81c30        4 weeks ago
    79.6MB
hello-world         latest              e38bc07ac18e        7 weeks ago
    1.85kB
```

Immagini: informazione (visibili per *root*)

- Pretty-printing del json con *json.tool* di Python o (va installato) *jq*:

```
# cat /var/lib/docker/image/overlay2/imagedb/content/sha256/
e38bc07ac18ee64e6d59cf2eafcdddf9cec2364dfe129fe0af75f1b0194e0c96 | python -m json.tool
{
    "architecture": "amd64",
    "config": {
 ...
      "Image": "sha256:c96933136c89265af0ba6c49ebfb11db8633c76518a0cf479a015781598c8e0b",
 ...
    },
    "container": "190fa62181e74c25cd9f430ea198d477abf6040e1004a64d61fb4a85c1ac082b",
...
    "os": "linux",
...
# apt install jq
...
# cat/var/lib/docker/image/overlay2/imagedb/content/sha256/
e38bc07ac18ee64e6d59cf2eafcdddf9cec2364dfe129fe0af75f1b0194e0c96 | jq
{
  "architecture": "amd64",
  "config": {
...
```

History di un'immagine: come è stata creata

```
# cat /var/lib/docker/image/overlay2/imagedb/content/sha256/
e38bc07ac18ee64e6d59cf2eafcdddf9cec2364dfe129fe0af75f1b0194e0c96 | python -m json.tool
... # stesso comando di prima, cerchiamo la sezione history:
"history": [
        {   "created": "2018-04-27T23:28:32.571994329Z",
            "created_by": "/bin/sh -c #(nop) ADD
file:81813d6023adb66b80fe163bc7db464004673838d17195b9d84aade4f8961b71 in / "
        },
        {   "created": "2018-04-27T23:28:33.469439435Z",
            "created_by": "/bin/sh -c set -xe \t\t&& echo '#!/bin/sh' > /usr/sbin/policy-rc.d \t&&
echo 'exit 101' >> /usr/sbin/policy-rc.d \t&& chmod +x /usr/sbin/policy-rc.d \t\t&& dpkg-divert --
local --rename --add /sbin/initctl \t&& cp -a /usr/sbin/policy-rc.d /sbin/initctl \t&& sed -i
's/^exit.*/exit 0/' /sbin/initctl \t\t&& echo 'force-unsafe-io' > /etc/dpkg/dpkg.cfg.d/docker-apt-
speedup \t\t&& echo 'DPkg::Post-Invoke { \"rm -f /var/cache/apt/archives/*.deb
/var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin || true\"; };' >
/etc/apt/apt.conf.d/docker-clean \t&& echo 'APT::Update::Post-Invoke { \"rm -f
/var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin ||
true\"; };' >> /etc/apt/apt.conf.d/docker-clean \t&& echo 'Dir::Cache::pkgcache \"\";
Dir::Cache::srcpkgcache \"\";' >> /etc/apt/apt.conf.d/docker-clean \t\t&& echo 'Acquire::Languages
\"none\";' > /etc/apt/apt.conf.d/docker-no-languages \t\t&& echo 'Acquire::GzipIndexes \"true\";
Acquire::CompressionTypes::Order:: \"gz\";' > /etc/apt/apt.conf.d/docker-gzip-indexes \t\t&& echo
'Apt::AutoRemove::SuggestsImportant \"false\";' > /etc/apt/apt.conf.d/docker-autoremove-suggests"
        }, ...
    ],
```

- ogni comando crea un *layer*, logicamente sopra i precedenti (fisicamente, però, vengono memorizzate solo le *diff*!)
- anche *docker inspect ubuntu* mostra la struttura dell'immagine, ma non i comandi usati per crearla, che troviamo invece nella sezione *history*

Gestione container da riga di comando

```
$ docker run -it ubuntu /bin/bash
root@f7d33d06a2db:/# echo "  prova" > /home/prova.txt
root@f7d33d06a2db:/# cat /home/prova.txt
  prova
root@f7d33d06a2db:/# exit     # vedremo che il file creato non è persistente!

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES

$ docker run -it ubuntu /bin/bash             # avvia un nuovo container, in cui, giustamente,
root@8d7464dfdb59:/# cat /home/prova.txt      # non si ritrova il file creato nella sessione per l'altro container
cat: /home/prova.txt: No such file or directory
root@8d7464dfdb59:/# exit

$ docker ps -a
CONTAINER ID    IMAGE     COMMAND        CREATED             STATUS                      PORTS
NAMES
8d7464dfdb59    ubuntu    "/bin/bash"    About a minute ago  Exited (1) 6 seconds ago
vigilant_pare
f7d33d06a2db    ubuntu    "/bin/bash"    4 minutes ago       Exited (1) 3 minutes ago
competent_bhabha

$ docker start competent_bhabha
competent_bhabha

$ docker ps
CONTAINER ID    IMAGE      COMMAND       CREATED             STATUS                      PORTS
NAMES
f7d33d06a2db    ubuntu     "/bin/bash"   10 minutes ago      Up 34 seconds
competent_bhabha
```

Gestione container

```
$ docker container ls -a
CONTAINER ID     IMAGE          COMMAND          CREATED          STATUS                      PORTS
  NAMES
8d7464dfdb59     ubuntu         "/bin/bash"      14 minutes ago   Exited (1) 12 minutes ago
 vigilant_pare
f7d33d06a2db     ubuntu         "/bin/bash"      17 minutes ago   Exited (0) 6 minutes ago
 competent_bhabha
6525c979ac5f     hello-world    "/hello"         3 hours ago      Exited (0) 3 hours ago
 infallible_yalow
$ docker [container] rm infallible_yalow
infallible_yalow
$ docker ps -a      # equivale a: docker container ls -a
CONTAINER ID     IMAGE          COMMAND          CREATED          STATUS                      PORTS
  NAMES
8d7464dfdb59     ubuntu         "/bin/bash"      14 minutes ago   Exited (1) 12 minutes ago
 vigilant_pare
f7d33d06a2db     ubuntu         "/bin/bash"      17 minutes ago   Exited (0) 6 minutes ago
 competent_bhabha
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
8d7464dfdb594e669178a5a449e7557ea26253c36e6b0138ea5038dc99fbca4f
f7d33d06a2db27c89844315f3b9bb7501c7ec1323d7c1ed0168dd6f7b485260c
Total reclaimed space: 119B
$ docker ps -a
CONTAINER ID     IMAGE          COMMAND          CREATED          STATUS                      PORTS
  NAMES
$ docker image ls   # equivale a: docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

Creazione di un'immagine

- Si vuole un container quanto più semplice possibile
- Il container eseguirà un classico *hello* (compilato da un nostro *hello.c*)
- Per default, il linking di un file eseguibile è *dinamico*, cioè avviene rispetto a DLL che dovrebbero poi far parte del file system del container

```
$ gcc -o hello hello.c
$ ldd hello   # mostra librerie dinamiche (DLL) linkate con l'eseguibile
   linux-vdso.so.1 (0x00007fff705dc000)
   libc.so.6 => /usr/lib/libc.so.6 (0x00007fd61584c000)
   /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
(0x00007fd615a25000)
```

- Bisogna invece compilare staticamente:

```
$ gcc -static -o hello hello.c
/usr/bin/ld: cannot find -lc
collect2: error: ld returned 1 exit status
```

- E, potrebbe servire (almeno per CentOS), installare il supporto necessario:

```
$ gcc -static -o hello hello.c
/usr/bin/ld: cannot find -lc
collect2: error: ld returned 1 exit status
$ sudo yum install glibc-static.x86_64
...
$ gcc -static -o hello hello.c
$ nm -C hello | grep ' T '    # mostra simboli di funzioni (da librerie statiche) linkate dentro l'eseguibile
```

- Ora si può costruire l'immagine per il container

Creazione immagine da *Dockerfile*

- Le immagini di Docker vengono create a partire da file di specifica, che, per default, si dovrebbero chiamare *Dockerfile*
  - ogni immagine è costruita aggiungendo *layer* a una base detta *scratch*
  - è possibile trasferire file dal sistema host all'immagine da costruire, con il comando *COPY*

```
~ # cat Dockerfile
FROM scratch
COPY hello /
CMD ["/hello"]
```

- Ora il comando *docker build .* , eseguito nella directory del *Dockerfile*, costruisce l'immagine con ID 51962532f387

```
$ docker build .    # genera immagine con eseguibile hello
Sending build context to Docker daemon  34.82kB
Step 1/3 : FROM scratch --->
Step 2/3 : COPY hello /
 ---> fb40ca4adec3
Step 3/3 : CMD [/hello]
 ---> Running in c69a6900d039
Removing intermediate container c69a6900d039
 ---> 51962532f387
Successfully built 51962532f387
```

- Nel *Dockerfile*, il comando *CMD ["Eseguibile"]* specifica che il container eseguirà il file *Eseguibile*, che si suppone sia incluso nell'immagine
- In alternativa a *CMD*, esiste il comando *ENTRYPOINT* (cf. https://stackoverflow.com/questions/21553353 e https://docs.docker.com/engine/reference/builder/#understand-how-cmd-and-entrypoint-interact)

Immagini: ID e alias

- Le immagini hanno come chiave univoca un *IMAGE ID* (esadecimale, parte di un hash, corrisponde ai nomi dei file di supporto in */var/lib/docker/image*...)
- Possono anche avere nomi più descrittivi, costruiti dall'utente come stringhe
  - per chiarezza, li diremo *alias*, riservando il termine "nome" per altro
- Ogni immagine quindi ha un solo *ID*, ma può avere più alias
- Questa informazione è visualizzata con *docker images*, che, però, chiama l'alias "*REPOSITORY*" (impropriamente)!

```
$ docker images     # l'immagine generata non ha nome (REPOSITORY) o TAG, ma solo IMAGE-ID
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              51962532f387        12 seconds ago
8.44kB
ubuntu              latest              452a96d81c30        4 weeks ago
79.6MB
hello-world         latest              e38bc07ac18e        7 weeks ago
1.85kB

$ docker rmi 51962532f387   # remove unnamed image
Deleted: sha256:51962532f3872729c660fa24ea8d442ee0cb28079b962dd8e0945866d5746d1d2
Deleted: sha256:fb40ca4adec3b81ac9348f37e3bf650a15a09ae5d8cf6bc344338f9c4d9b66eb
Deleted: sha256:059e6802e435a6e90934dd95d2f62be8e67646406b5fd967876d23a548b2d22e
```

- Rimuoviamo l'immagine per ricrearla con un alias (per facilitare i riferimenti)
- In alternativa, con il comando *docker tag* si sarebbe potuto dare il nome all'immagine appena creata "anonima"

Creazione immagini: meglio con nome

```
$ docker build -t my-hello-img .     # -t: dà un alias alla nuova image
Sending build context to Docker daemon  34.82kB
Step 1/3 : FROM scratch
 --->
Step 2/3 : COPY hello /
 ---> cddbf51ca48d
Step 3/3 : CMD [/hello]
 ---> Running in a22c52271c71
Removing intermediate container a22c52271c71
 ---> bd26e03c2ba0
Successfully built ba0b5dc15106
Successfully tagged my-hello-img:latest

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
my-hello-img        latest              ba0b5dc15106        About a minute ago
8.44kB
ubuntu              latest              452a96d81c30        4 weeks ago
79.6MB
hello-world         latest              e38bc07ac18e        7 weeks ago
1.85kB
$ docker run --rm my-hello-img     # --rm: rimuove il container dopo la sua terminazione
Hello mondo!
```

NB: l'immagine ha un solo *IMAGE-ID* (numerico), ma può avere più alias con struttura *nome-immagine*[*:tag*] (le quadre [...] delimitano una parte opzionale)

- il *tag*, per default, vale "*latest*"; per lo più lo si usa per indicare una versione
- semplificando, nel seguito il *nome-immagine* sarà semplice (senza slash) o avrà prefisso *username/* , con *username* valido sul registry registry-1.docker.io
- si potrebbero usare anche altri registry, come discusso nella prossima slide

Alias di immagini: *docker tag* e precisazioni

```
$ docker images my-hello-img
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-hello-img        latest              ba0b5dc15106        About a minute ago
8.44kB$
```

Come detto, individuiamo le immagini con alias con struttura (semplificata)
[*registry-host*:*port*/][*username*/]*nome-semplice*[*:tag*]

- il *tag* dovrebbe indicare una versione e, per default, vale "*latest*"
- il *registry-host* per default è registry-1.docker.io (non lo si indica, a meno che si sia organizzato il proprio registry)
- *username* dev'essere un utente di *registry-host* e si dice anche "repository" (si intende cioè il repository dell'utente *username*)

  per default il repository è "*library*" (vi si trovano le immagini pubbliche standard, tipo debian o ubuntu)

docker tag è il comando per introdurre nuovi alias e non (come potrebbe sembrare) solo un tag (che è solo una parte dell'alias).

```
$ docker tag ba0b5dc15106 my-hello

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-hello-img        latest              ba0b5dc15106        35 seconds ago      857kB
my-hello            latest              ba0b5dc15106        35 seconds ago      857kB
```

Alias di immagini con repository

- In vista di un *push* (upload sul registry pubblico, discusso in altre slide), è opportuno che gli alias contengano lo *username* (repository) appropriato, p.es.:

```
$ docker tag my-hello-img gpappala/my-hello   # registry docker (default), username/repository gpappala
$ docker tag my-hello-img registry.gitlab.com/gpappala/myproj/my-hello # altro registry docker
$ docker images
REPOSITORY                                    TAG     IMAGE ID      CREATED          SIZE
my-hello-img                                  latest  ba0b5dc15106  About an hour ago  857kB
my-hello                                      latest  ba0b5dc15106  About an hour ago  857kB
gpappala/my-hello                             latest  ba0b5dc15106  About an hour ago  857kB
registry.gitlab.com/gpappala/myproj/my-hello  latest  ba0b5dc15106  About an hour ago  857kB
```

Commit: creazione immagine da container

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              452a96d81c30        5 weeks ago
79.6MB

$ docker run -it ubuntu /bin/bash
root@eb07824b0791:/# apt-get update           # siamo dentro il container Ubuntu
...
root@eb07824b0791:/# apt-get upgrade
...
root@eb07824b0791:/# apt-get install python   # python not installed in standard Ubuntu image
...
root@eb07824b0791:/# python --version
Python 2.7.15rc1
root@eb07824b0791:/# exit

$ docker ps -a
CONTAINER ID        IMAGE    COMMAND        CREATED             STATUS                         PORTS
  NAMES
eb07824b0791        ubuntu   "/bin/bash"    44 minutes ago      Exited (0) 39 minutes ago
  vigorous_haibt

$ docker commit vigorous_haibt ubuntu-python-img    # genera immagine ubuntu-python-img da
container
sha256:46dc2d8b0e2f5df0ad198b6d39d1a75bace387fb9acc17c446ebc79918c9e670

$ docker images    # ubuntu-python-img è la nuova immagine da container
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu-python-img   latest              46dc2d8b0e2f        5 seconds ago       161MB
ubuntu              latest              452a96d81c30        5 weeks ago
79.6MB
```

- Così, si mette a punto il container interagendo con la shell, come per configurare un sistema e, quando si è soddisfatti, con *docker commit* si crea l'immagine dal container
- L'alternativa, più scomoda, è installare python "FROM ubuntu" con comandi di un apposito *Dockerfile*

Immagine con sessione interattiva

```
$ docker run ubuntu-python-img        # la nuova immagine non va avviata così: occorre sessione interattiva!
^C
# run -t instructs Docker to allocate (in the host) a pseudo-TTY connected to the container's stdin;
# run -i Keep STDIN open even if not attached creating an interactive bash shell in the container
$ docker run -it ubuntu-python-img   # interattiva!
root@12de96dbdaa6:/# python --version
Python 2.7.15rc1
root@12de96dbdaa6:/# apt install python2-flask   # installa framework Web per python
...
root@12de96dbdaa6:/# exit
exit
$ docker ps -a
CONTAINER ID   IMAGE              COMMAND       CREATED             STATUS
NAMES
12de96dbdaa6   ubuntu-python-img  "/bin/bash"   4 minutes ago       Exited (0) 4 minutes ago
festive_gates
32964665ffb1   ubuntu-python-img  "/bin/bash"   5 minutes ago       Exited (0) 5 minutes ago
elated_tereshkova
eb07824b0791   ubuntu             "/bin/bash"   About an hour ago   Exited (0) About an hour ago
vigorous_haibt
$ docker container rm elated_tereshkova    # il primo container avviato non interattivamente da ubuntu-
python-img
                                           #   non serve più, mentre da festive_gates, in cui si è installato
flask ...
$ docker commit festive_gates ubuntu-python-flask    # si genera immagine ubuntu-python-flask
da container
sha256:6a7f23498ab7012ab54f573de726ab98cf61582a826bc8343a8aa8bc5d822
$ docker images
```

Immagine con avvio diretto Python

```
$ docker ps -a
CONTAINER ID     IMAGE    COMMAND        CREATED            STATUS                        PORTS
NAMES
eb07824b0791     ubuntu   "/bin/bash"    44 minutes ago     Exited (0) 39 minutes ago
festive_gates
```

Si vuole generare una nuova immagine attraverso "editing" del container *festive_gates* o, per meglio dire, della sua immagine:

- la nuova immagine avvierà direttamente python eseguendo il comando python "*import this*" (*this* è un "easter egg").

La nuova immagine, generata ancora con *docker commit*, potrebbe aggiungersi alle precedenti o, come qui sotto, sovrascrivere l'esistente *ubuntu-python-img*

```
$ docker commit --change='CMD ["python", "-c", "import this"]' festive-gates ubuntu-
python-img
sha256:d5698ef036e601902aebd26aaff98d22f632868a1ab6c1f8fba2e352a4ddacdd

# si noti l'array ["python", "-c", "import this"]' con comando "python" e argomenti da eseguire

# --change=... specifica che nel (l'immagine del) container festive_gates cambierà un elemento

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu-python-img   latest              d5698ef036e6        12 seconds ago      161MB
ubuntu              latest              452a96d81c30        5 weeks ago
79.6MB
hello-world         latest              e38bc07ac18e        7 weeks ago
1.85kB

$ docker run ubuntu-python-img    # lanciamo container da nuova immagine, frutto del commit con editing
The Zen of Python, by Tim Peters ...

$ docker ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS
```

Immagini e container: considerazioni

Vale la pena, a questo punto, di riportare alcune osservazioni sulla relazione tra immagini e container:

- un'immagine è come uno "stampo", un "template" da cui vengono creati i container (con *docker run*)
- all'inverso, da un container terminato (e probabilmente variato rispetto all'avvio, p.es. dopo installazioni su una distro) è possibile (*docker commit*) costruire "per estrazione" un'immagine
- quindi, benché i container terminati occupino risorse, possono essere utili
- se però non servono, conviene evitare che restino disponibili (avviandoli con *docker run -rm*) o eliminarli tutti, con *docker container prune* o selettivamente con *docker rm <container>*
- non è possibile eliminare un container se non è terminato
  - per fermare un container in esecuzione: *docker stop <container>*
- non è possibile eliminare un'immagine (con *docker rmi*) fintantoché vi sono ancora in giro container che la istanziano

Sintassi dei comandi CLI docker

È molto regolare: *docker classe-di-risorse comando sottocomando…* ; p.es.:

docker container [ls|rm|start|stop|commit|inspect|run|…] CONTAINER…

A volte la classe di risorse, specie *container* si può omettere, quindi:

docker [rm|start|stop|commit|inspect|run|…] CONTAINER…

NB: *docker ps [-a]* equivale a *docker container ls [-a]*

Oppure, per le risorse *image*:

docker images equivale a *docker image ls*

docker rmi equivale a *docker image rm*

docker build equivale a *docker image build*

docker save equivale a *docker image save*