Attività 3.a: architettura di Docker e principali funzionalità

Esercitazione

Prerequisiti: Macchina Linux con Docker, sshd, gcc, Python e pip installati

(accertarsi di aver aggiunto l'utente con il quale si lavora al gruppo *docker* per evitare di usare *sudo* ogni volta)

Obiettivo: creare un'immagine contenente un solo programma eseguibile e verificare l'architettura di Docker

Fase 1: creare la prima immagine e il primo container

1. Creare una cartella di lavoro ed un programma in C al suo interno.
2. Il programma si limiterà a stampare un messaggio. Andrà compilato chiedendo l'inclusione statica delle librerie nel modulo eseguibile. Se il programma si chiama *prova1.c*:

```
$ gcc –o prova1.exe --static prova1.c
```

   Cambiarne i diritti e provarne il funzionamento:

```
$ chmod 777 prova1.exe
$ ./prova1.exe
```

3. Creare un *Dockerfile* per la generazione di un'immagine partendo da un'immagine di base vuota *(scratch)*

```
FROM scratch
COPY prova1.exe /
CMD ["/prova1.exe"]
```

4. Generare un'immagine con nome *img1* a partire dal *Dockerfile* nella directory corrente; verrà creata un'immagine intermedia (dopo la COPY) e un'immagine finale.

```
$ docker build –t img1 .
```

5. Elencare le immagini presenti:

```
$ docker image ls -a
```

6. Eseguire un container associato all'immagine *img1,* assegnandogli un nome *(--name*), chiedendone la rimozione al termine dell'esecuzione del programma *(--rm*) e associandogli una console (*-it*)

```
$ docker container run --name cont1 --rm –it img1
```

Fase 2: lista dei container attivi

1. Modificare il programma C, in modo da avere la stampa di due messaggi, intervallati da un delay (*sleep()* in *unistd.h*) di un minuto e generare un'immagine.
2. Aprire una finestra *ssh* (opzione *–l* per specificare l'utente) o *putty*
3. Eseguire un container sull'immagine creata dalla finestra principale
4. Sull'altra finestra elencare i container attivi

```
$ docker container ps -a
```

5. Sempre sull'altra finestra elencare i processi attivi (*pstree -p*): si noterà la presenza di un processo *containerd-shim* che ha come figlio l'eseguibile ottenuto dal programma C; notare anche (comando ps) che tale processo è attivato all'interno di un *namespace* il cui codice è lo stesso assegnato al container.

Fase 3: costruzione di un'immagine basata su quella creata precedentemente

1. In una nuova cartella creare un programma *prova2.c* che stampi un messaggio e che poi esegua il programma della fase precedente (funzione *execl*; vedi qui)

```
        execl("prova1.exe","prova1.exe",NULL);
```

2. Creare un'immagine *img2* che contenga il nuovo programma, basandosi sull'immagine precedente: tale nuova immagine conterrà quindi entrambi i programmi creati che verranno poi eseguiti nella sequenza *prova2-prova1*; lanciare il container e verificare

```
FROM img1
COPY prova2.exe /
CMD ["/prova2.exe"]
```

3. Esaminare il file XML che contiene, fra le altre cose, la descrizione della seconda immagine; notare la presenza di un *overlay file system*: la *MergeDir* sarà generata al lancio del container mentre la *LowerDir* e la *UpperDir* corrispondono rispettivamente all'immagine base *img1* e *img2.* Verificare il contenuto delle due cartelle. (accedere come root: *sudo su*)

```
$ docker image inspect img2
```

4. Modificare il Dockerfile di *img2*, chiedendo l'esecuzione di *prova1.exe*: notare che la *build*, al passo 2, utilizza l'immagine intermedia generata nella generazione dell'immagine precedente: questo in quanto il passo 1 e il passo 2 non sono cambiati rispetto all'immagine precedente.

5. Eliminare tutte le immagini generate

```
$ docker image prune –a
$ docker image rm $(docker image ls -aq)
```