



Università degli Studi di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea in Informatica Triennale

Luca Steccanella



CPNLab

Un tool didattico per la gestione di reti
con i Docker Container

Relazione progetto finale

Relatore

Chiar.mo. Prof. Salvatore Riccobene

Correlatore

Dott. Federico Santoro

“Simplicity is prerequisite for reliability” EDSGER W. DIJKSTRA

“I computer sono incredibilmente veloci, accurati e stupidi. Gli uomini sono incredibilmente lenti, inaccurati e intelligenti. L'insieme dei due costituisce una forza incalcolabile.” ALBERT EINSTEIN

“The purpose of computing is insight, not numbers” RICHARD HAMMING

*“Il computer non è una macchina intelligente
che aiuta le persone stupide,
anzi è una macchina stupida che funziona solo
nelle mani delle persone intelligenti”* UMBERTO ECO

SOMMARIO

Prefazione	6
1 Introduzione	7
1.1 Le topologie di rete virtuali ad oggi	7
1.2 Le problematiche	8
1.3 La soluzione precedente: VB-ANT	10
1.4 La nostra soluzione: CPNLab	11
2 Panoramica di Docker.....	13
2.1 Cos'è Docker	13
2.2 Container VS Virtual Machine	14
2.3 Installazione.....	15
2.3.1 Windows & MacOS	15
2.3.2 CentOS.....	15
2.3.3 Fedora	15
2.3.4 Debian	16
2.3.5 Ubuntu.....	16
2.4 La gestione dei container e immagini	17
2.5 I network.....	22
2.5.1 Le reti di base	22
2.5.2 I comandi	22
2.6 La formattazione.....	26
3 Altri strumenti, linguaggi e librerie utilizzate	27
3.1 C++14.....	27
3.2 Qt5.....	28
3.2.1 Panoramica.....	28
3.2.2 Com'è composto un widget	29
3.3 CLion	34
3.4 Astah UML	35

4	Il backend: la comunicazione con Docker	36
4.1	Premesse	36
4.2	La classe dockInt	36
4.2.1	Panoramica	36
4.2.2	Header	37
4.2.3	La definizione delle funzioni.....	38
4.3	La classe manager	42
4.3.1	Panoramica	42
4.3.2	L'header	42
4.3.3	Definizione delle funzioni.....	43
5	Il frontend: Come funziona la GUI	48
5.1	Introduzione	48
5.2	Integrazione con Docker	48
5.3	La comunicazione fra i widget.....	48
5.4	La mainWIndow	50
5.4.1	File .ui	50
5.4.2	Header	52
5.4.3	Implementazione	53
5.5	Panoramica dei widget.....	56
6	Requisiti, installazione e distribuzione.....	57
7	Guida all'uso.....	58
7.1	Avvio	58
7.2	Aggiungere un network.....	58
7.3	Aggiungere un nuovo container (nodo)	59
7.4	Connettere e rimuovere network dai container.....	59
7.5	L'interfaccia dedicata al container	60
7.6	Il terminale.....	61
8	Conclusioni	62
8.1	Efficacia.....	62
8.2	Sviluppi futuri.....	62

8.3	Ringraziamenti	62
8.4	Sitografia.....	66

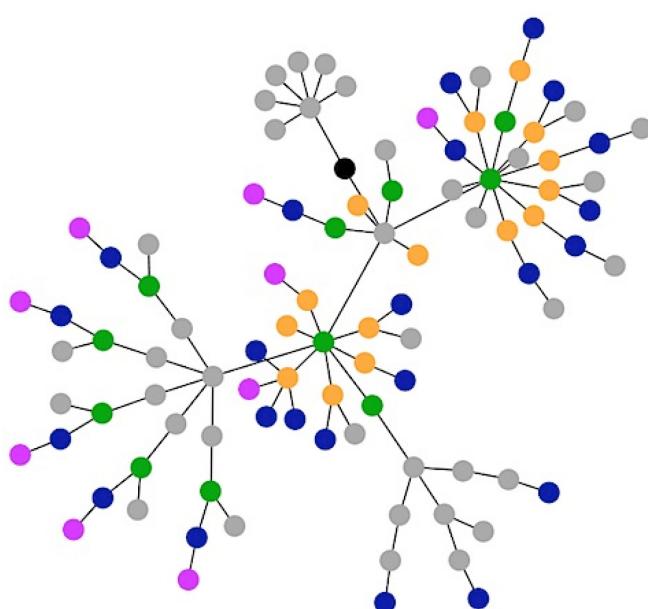
PREFAZIONE

Nell'informatica moderna si presenta molto spesso la necessità di realizzare topologie di rete per i più disparati motivi e scenari d'uso: uno studente od un appassionato alle prime armi che vuole muovere i primi passi nel mondo della gestione delle reti, un amministratore di rete che deve testare delle configurazioni di rete oppure un team che deve testare un software, effettuare un assessment di sicurezza su un ambiente specifico o creare una rete all'interno di un'unica macchina fisica.

Il più delle volte è conveniente creare queste topologie virtualmente, poiché non si ha a disposizione l'hardware fisico o non è conveniente usarlo. Immaginiamo ad esempio uno studente alle prime armi, è improbabile che egli abbia la possibilità di creare una topologia usando nodi fisici. Creare nodi e reti virtuali sarà senz'altro più alla portata delle sue risorse.

A tal scopo tutto l'hardware e le LAN (Local Area Network) vengono virtualizzati tramite l'uso di macchine virtuali, ma la virtualizzazione non è l'unica soluzione, ad esempio vi sono i Docker Container e le Docker Network che discuteremo in modo dettagliato più avanti.

Ad ogni modo anche le topologie di reti virtuali presentano alcune problematiche spinose, ed è proprio lo scopo di questa tesi proporre, sviluppare e discuterne una soluzione.

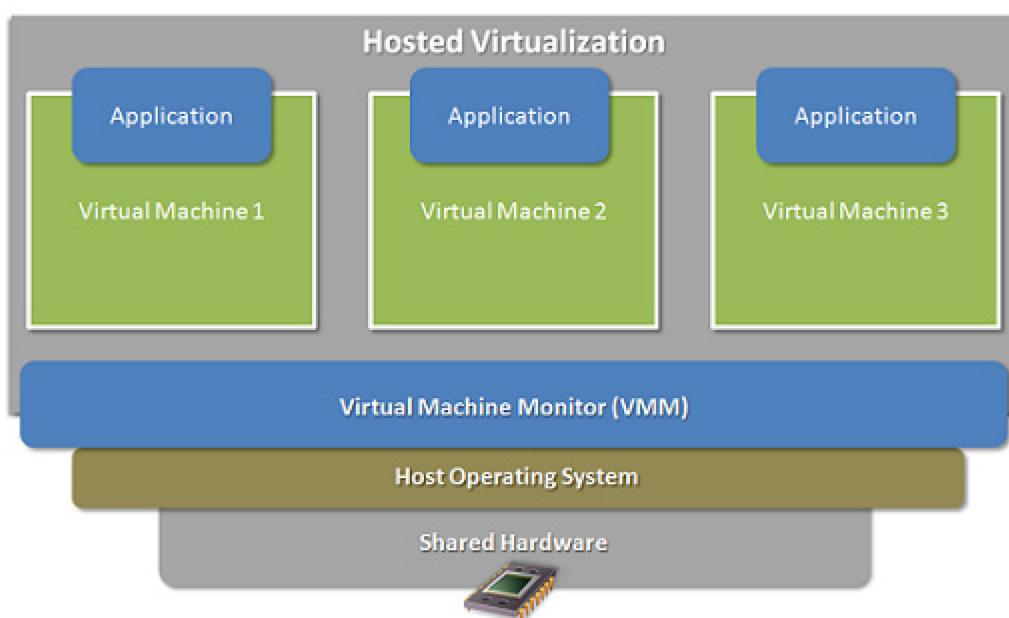


1 INTRODUZIONE

1.1 LE TOPOLOGIE DI RETE VIRTUALI OGGI

Nella stragrande maggioranza dei casi le topologie di rete vengono realizzate e gestite tramite l'uso di macchine virtuali.

Con l'ausilio di software come VMware o VirtualBox è infatti possibile creare delle macchine virtuali che si comportano esattamente come dei nodi fisici; è possibile scegliere sistema operativo, servizi, applicativi, schede di rete e altro ancora, per poi procedere, partendo da zero, alla configurazione del singolo nodo e delle singole reti.

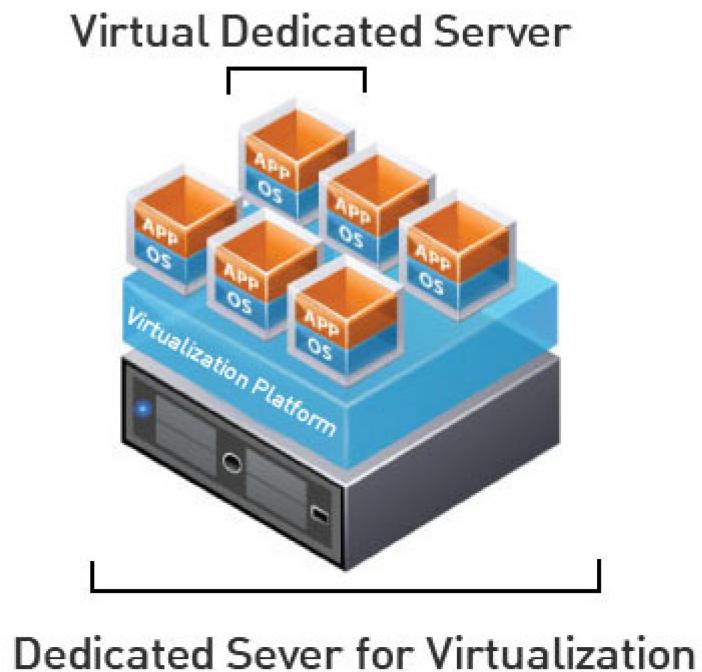


Il vantaggio primario dell'uso di macchine virtuali a tale scopo è dovuto al fatto che, tramite una singola macchina fisica, è possibile creare un numero arbitrario di nodi configurabili sotto ogni aspetto, sia hardware che software.

Nel caso didattico, ad esempio, un singolo studente tramite il proprio PC può creare intere topologie di rete e configurarle come meglio crede. Un Professore universitario, d'altro canto, può configurare un ambiente d'esame efficace su un singolo PC di laboratorio.

Nel mondo enterprise la virtualizzazione ha preso molto piede, ad esempio, immaginiamo di avere a disposizione una macchina con 256GB di memoria virtuale, 24 core fisici e 48 virtuali. È ragionevole ipotizzare che tramite

L'utilizzo di questa singola macchina ed un Hypervisor di primo livello è possibile creare un'intera infrastruttura di rete dotata di tutto il necessario.



Dedicated Sever for Virtualization

Per i motivi sopra citati le funzionalità legate al networking dei vari software di virtualizzazione si sono evolute notevolmente nel tempo. Ciò permette una grande varietà di configurazioni, che in alcuni scenari più semplici possono essere automatizzate completamente, richiedendo di conseguenza minori interventi da parte dell'utente; ad esempio la connessione alla rete della macchina ospitante può essere effettuata in maniera trasparente ed in varie modalità, ad ogni modo l'isolamento di entrambe le reti resta possibile.

1.2 LE PROBLEMATICHE

L'utilizzo di macchine virtuali, soprattutto nel mondo didattico, non è sempre così semplice. Di seguito analizzeremo le varie problematiche.

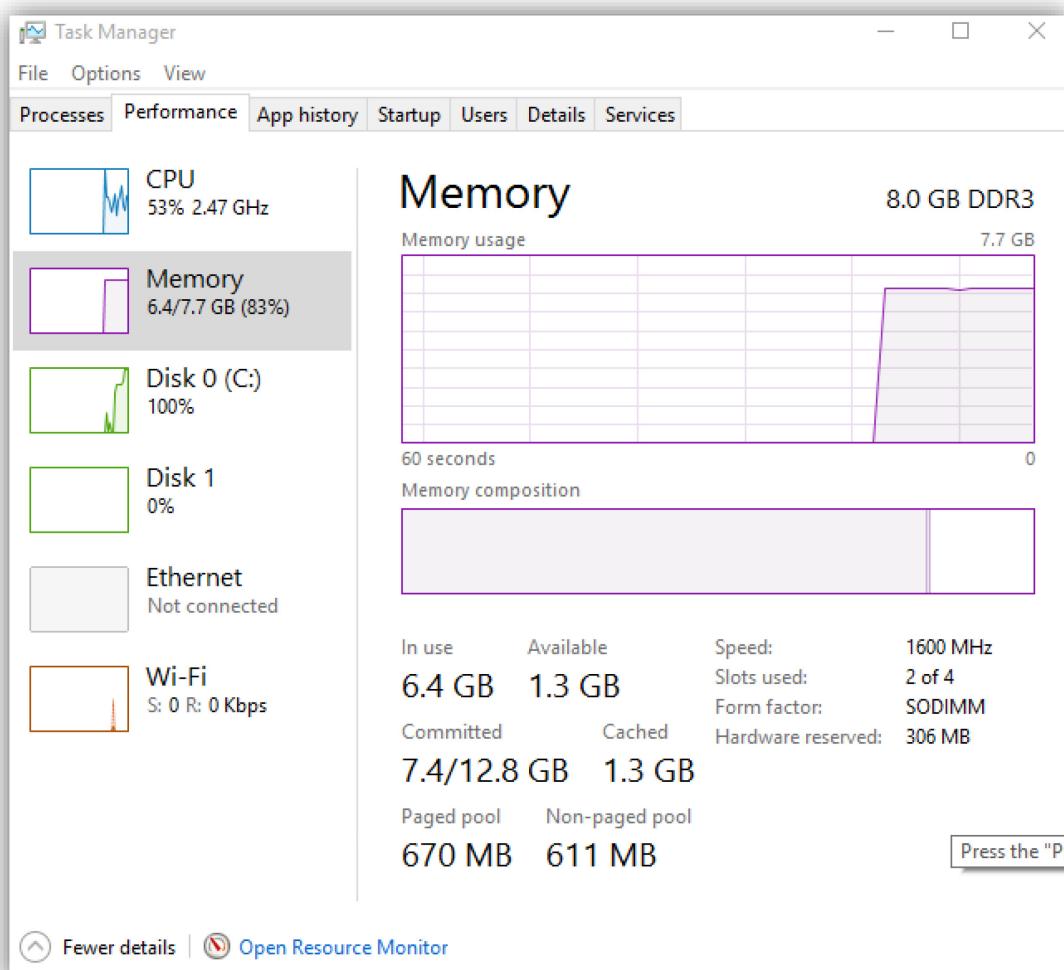
Per cominciare l'utilizzo di macchine virtuali a tale scopo richiede comunque un quantitativo di risorse adeguato; ogni nodo della rete deve virtualizzare per intero hardware e sistema operativo, ciò consuma risorse.

In particolare ogni nodo deve avere un sistema operativo completo, in modo da non porre alcun limite allo studente in termini di configurazione.

Tutte le funzionalità base di networking ed i tool annessi devono essere presenti e funzionanti.

Ogni nodo deve essere in grado di effettuare l'ip forwarding ed essere dotato di applicativi come ifconfig, ip, route e traceroute; supportando anche IPv6. La distribuzione scelta deve dunque derivare da una base affidabile e rodata, come Debian, sistemi RHEL o Ubuntu.

Ipotizziamo adesso di avere uno studente dotato di un PC non particolarmente performante e con un quantitativo di RAM che ammonta a 4GB. Tralasciando i calcoli richiesti alla CPU e lo spazio sul disco occupato da ogni macchina, fattori che comunque non sono da sottovalutare, l'ammontare di memoria richiesto da ogni nodo sopra descritto potrebbe raggiungere almeno i 500MB. Nel caso del nostro studente avviare più di 4 nodi occuperebbe memoria sufficiente da mettere in crisi il sistema operativo ospitante.



Creare un nodo con questa metodologia corrisponde ad istanziare una nuova macchina virtuale e, sebbene si possa preparare un'immagine pronta, ciò richiede dei tempi di inizializzazione e configurazione dell'hardware virtuale.

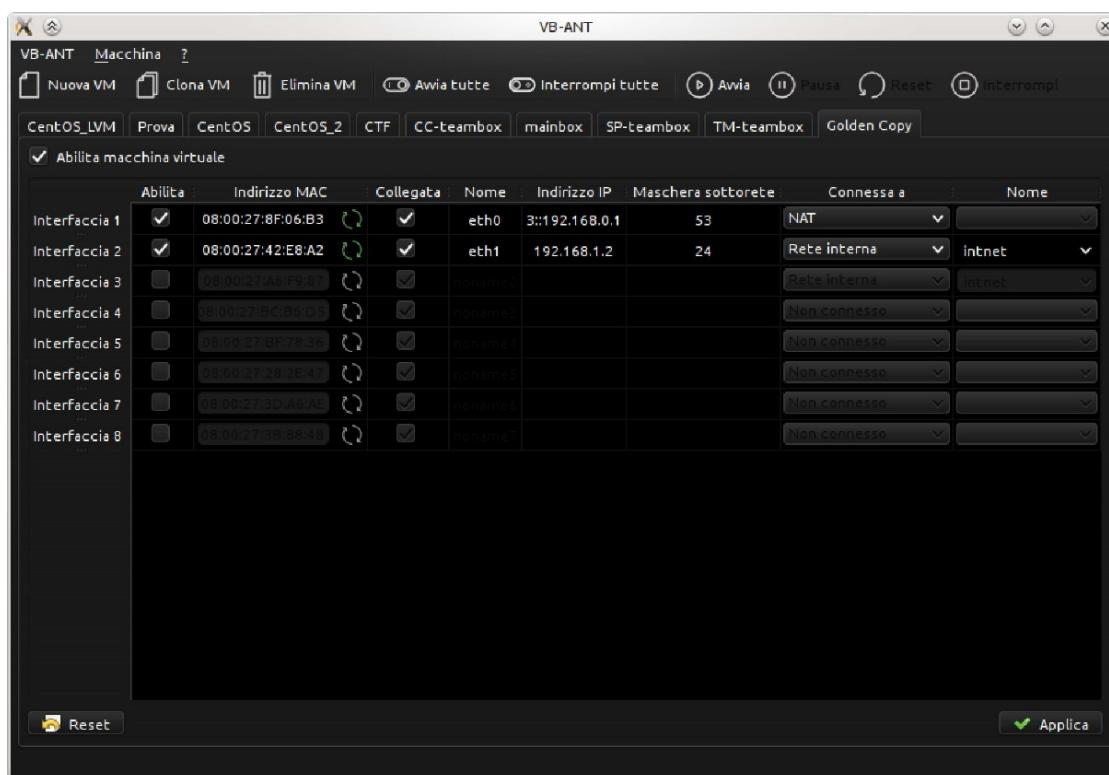
Inoltre date le varie configurazioni hardware e software a disposizione degli studenti si potrebbe incorrere in ulteriori complicazioni.

Fra l'altro il docente, in sede di esame, preferirebbe affidarsi ad un sistema più controllabile al fine di assicurarsi che l'esame venga svolto nei termini imposti dallo stesso.

Per tali motivi si è voluto dare il via allo sviluppo di un software che potesse risolvere o perlomeno limitare tali problematiche e che fosse integrabile in un ambiente completo e pronto all'uso.

1.3 LA SOLUZIONE PRECEDENTE: VB-ANT

Un primo approccio risolutivo alla questione è il software VB-ANT che permette di gestire tramite le API di VirtualBox le macchine controllate da quest'ultimo.



Tale software, sebbene fornisse tramite un'interfaccia unica la gestione completa delle macchine virtuali ed in particolare le configurazioni relative alla rete ed al routing, si è rivelato inadatto allo scopo.

Sfortunatamente le API di VirtualBox non funzionano benissimo ed il montaggio delle macchine virtuali non risulta affatto semplice. Realizzare un software in grado di comunicare con VirtualBox e che svolgesse tutte le operazioni desiderate si è rivelato complesso ed ha prodotto un software che necessita di molte dipendenze e tool esterni. A dispetto del grande lavoro di sviluppo, ciò ha reso il software instabile e troppo dipendente da fattori esterni.

Inoltre il software si può rivelare troppo complesso per uno studente alle prime armi e, soprattutto, usando le macchine virtuali gestite tramite VirtualBox le problematiche prestazionali sono state soltanto mitigate tramite l'uso di un'immagine studiata per lo scopo.

È dunque apparso evidente, dopo questa analisi, che andava trovata una nuova soluzione.

1.4 LA NOSTRA SOLUZIONE: CPNLAB

Durante la fase di brainstorming si è compreso che era necessario abbandonare del tutto l'idea di usare VirtualBox e la virtualizzazione in generale e dunque passare a qualcosa di più moderno, qualcosa che avrebbe risolto sia le problematiche di sviluppo sia quelle prestazionali.

Questo "qualcosa" è Docker: un sistema software che avvalendosi delle funzionalità di isolamento delle risorse del Kernel Linux, come ad esempio *cgroups* e *namespaces*, consente a "container" indipendenti di coesistere sulla stessa istanza di Linux, evitando la creazione, configurazione e mantenimento di una macchina virtuale.

In questo modo le risorse occupate da ogni singolo container sono esigue. Un'immagine di Debian dotata dei tool di networking, dunque perfetta per il nostro scopo, istanziata su una macchina fisica, può occupare al massimo circa 50MB di memoria RAM e impiega sia molto meno spazio sul disco che cicli di CPU rispetto ad una macchina virtuale. Questo è dovuto al fatto che, tramite l'uso dei Docker container, i programmi eseguiti all'interno di questi ultimi richiedono le stesse risorse di un comune processo eseguito direttamente all'interno della macchina ospitante, evitando, fra l'altro, di dover gestire un'intera macchina virtuale con hardware e sistemi operativi propri. Le prestazioni risultanti sono pertanto a favore di Docker.

Inoltre Docker permette con semplicità, tramite la propria CLI (Command Line Interface), di creare container (dunque nuovi nodi) e network in maniera immediata e di estrarre e formattare informazioni relative a questi ultimi in maniera semplice, veloce ed efficace. La comunicazione con Docker dunque, anche a livello di programmazione, appare da subito molto più chiara e trasparente rispetto a VirtualBox.

Visti i vantaggi certi ed evidenti, si procede immediatamente allo sviluppo di un software che si avvalga dei Docker container e network al fine di raggiungere l'obiettivo preposto.

Questo software è il tool **CPNLab**, acronimo per *Container Powered Networking Laboratory*, che incarnerà la soluzione alle problematiche emerse durante la nostra analisi.

2 PANORAMICA DI DOCKER



2.1 Cos'è DOCKER

Prima di procedere è bene comprendere al meglio la tecnologia su cui si basa CPNLab.

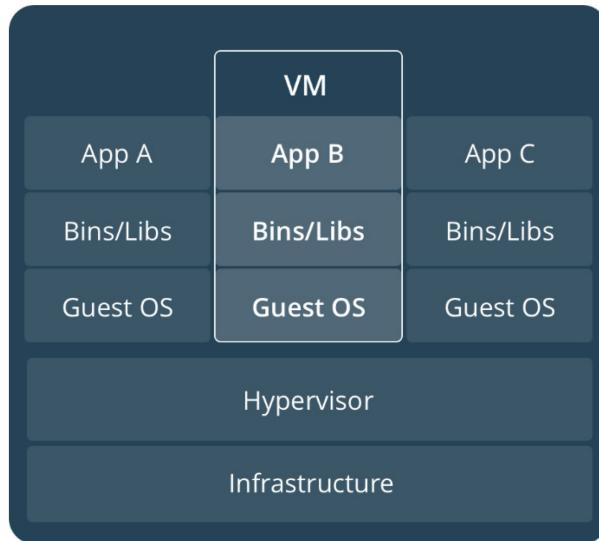
Docker è un progetto open-source che automatizza il rilascio all'utente finale di applicazioni all'interno di appositi container, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. Come scritto in precedenza Docker utilizza varie funzioni del kernel Linux al fine di ottenere un isolamento totale di ogni container.

Attraverso i namespace del kernel Linux si isola l'ambiente operativo di ogni container, incluso l'albero dei processi, la rete, gli ID utente ed i file system montati. I cgroups forniscono invece l'isolamento delle risorse, inclusa la CPU, la memoria, i dispositivi di I/O a blocchi e la rete.

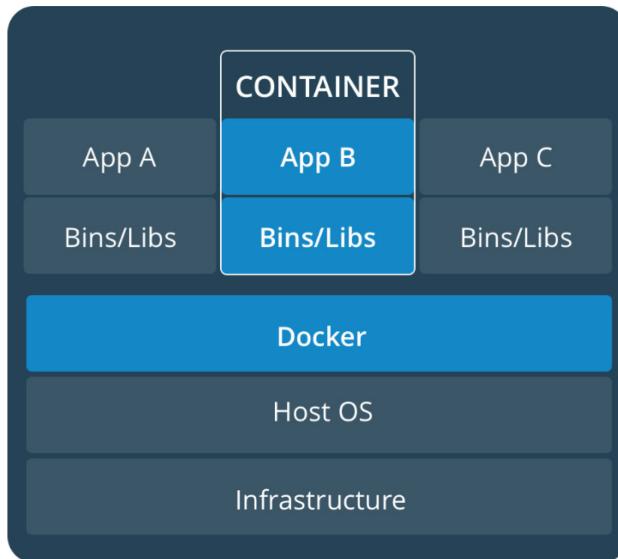
Inoltre, dalla versione 0.9, Docker include la libreria *libcontainer* per poter utilizzare direttamente le funzionalità di virtualizzazione del kernel Linux, in aggiunta alle librerie dedicate preesistenti come *libvirt*, *LXC* e *systemd-nspawn*.

Lo scopo di tutto ciò è produrre delle applicazioni, dette container, che siano in grado di essere eseguite in maniera totalmente isolata rispetto al resto del sistema e, all'occorrenza, comportando delle macchine indipendenti. Il tutto occupando le stesse risorse di un qualunque altro processo.

2.2 CONTAINER VS VIRTUAL MACHINE



Le macchine virtuali sono un'astrazione dell'hardware fisico ed eseguono un intero sistema operativo dotato di applicazioni, librerie, driver, aggiornamenti e molti altri dati. Le macchine virtuali trasformano una singola macchina in una moltitudine di macchine, occupano dunque molto spazio e risorse. Inoltre è più facile perderne i dati ivi contenuti.



I container invece sono un'astrazione a livello applicativo che condividono un unico kernel e le stesse risorse. Sono totalmente isolati fra loro e vengono eseguiti come comuni processi. Essi contengono solo gli eseguibili e le eventuali dipendenze. Dunque occupano molto meno spazio e le risorse richieste da un container sono sostanzialmente inferiori ad una macchina virtuale. Inoltre i container non necessitano di alcuna configurazione aggiuntiva e funzioneranno su ogni macchina in cui verranno eseguiti.

Lo sviluppatore può dunque costruire un'immagine di Docker che include esattamente e solamente tutto il necessario. Si parte dalla base e si aggiungono i componenti richiesti.

Le macchine virtuali invece funzionano al contrario, si parte con un sistema completo e si tenta di eliminare il superfluo.

A causa dei sopracitati motivi l'uso di Docker sta prendendo molto piede nel mondo enterprise poiché consente di ottimizzare sensibilmente il workflow.

2.3 INSTALLAZIONE

L'installazione di Docker è molto semplice e supporta una grande varietà di sistemi operativi.

2.3.1 Windows & MacOS

Nei sistemi Windows e MacOS è possibile utilizzare gli installers reperibili nel sito ufficiale.

Per tutte le varie distribuzioni Linux primarie come CentOS, Fedora, Debian e Ubuntu è possibile installarlo tramite i package manager apt, apt-get, yum e dnf.

2.3.2 CentOS

```
$ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
$ sudo yum makecache fast
$ sudo yum install docker-ce
```

2.3.3 Fedora

```
$ sudo dnf -y install dnf-plugins-core
$ sudo dnf config-manager \
    --add-repo \
    https://download.docker.com/linux/fedora/docker-ce.repo
$ sudo dnf makecache fast
$ sudo dnf install docker-ce
```

2.3.4 Debian

```
$ sudo apt-get update  
$ sudo apt-get install \  
    apt-transport-https \  
    ca-certificates \  
    curl \  
    gnupg2 \  
    software-properties-common  
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -  
$ sudo apt-key fingerprint 0EBFCD88  
  
pub   4096R/0EBFCD88 2017-02-22  
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  
uid            Docker Release (CE deb) <docker@docker.com>  
sub   4096R/F273FCD8 2017-02-22  
$ sudo add-apt-repository \  
    "deb [arch=amd64] https://download.docker.com/linux/debian \  
    $(lsb_release -cs) \  
    stable"  
$ sudo apt-get update  
$ sudo apt-get install docker-ce
```

2.3.5 Ubuntu

```
$ sudo apt update  
$ sudo apt install \  
    apt-transport-https \  
    ca-certificates \  
    curl \  
    software-properties-common  
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
$ sudo apt-key fingerprint 0EBFCD88  
  
pub   4096R/0EBFCD88 2017-02-22  
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  
uid            Docker Release (CE deb) <docker@docker.com>  
sub   4096R/F273FCD8 2017-02-22
```

```
$ sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
$ sudo apt update
$ sudo apt install docker-ce
```

2.4 LA GESTIONE DEI CONTAINER E IMMAGINI

I container vengono creati a partire da un'immagine che può essere creata dall'utente o essere prelevata tramite il repository ufficiale di Docker, che contiene sia le immagini ufficiali sia quelle create dalla community.

Le immagini possono essere definite tramite degli appositi dockerfile e pacchettizzate in appositi archivi.

Ad esempio è possibile creare un container partendo da un'immagine contenente un sistema Debian 8, ma nel nostro caso quest'immagine è stata lievemente modificata per adattarla ai nostri scopi. Esattamente come è previsto dalla filosofia su cui si basa Docker, si ha una base e vi si aggiunge soltanto il necessario.

I comandi principali che permettono di manipolare le immagini ed i container sono i seguenti.

docker images

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Questo comando mostra tutte le immagini presenti nella propria macchina, includendo il repository, i tag e le dimensioni.

Esempi:

```
$ docker images
```

REPOSITORY SIZE	TAG	IMAGE ID	CREATED
<none> 1.089 GB	<none>	77af4d6b9913	19 hours ago
committ 1.089 GB	latest	b6fa739cedf5	19 hours ago

<none>	<none>	78a85c484f71	19 hours ago
1.089 GB			
docker	latest	30557a29d5ab	20 hours ago
1.089 GB			
<none>	<none>	5ed6274db6ce	24 hours ago
1.089 GB			
postgres	9	746b819f315e	4 days ago
213.4 MB			
postgres	9.3	746b819f315e	4 days ago
213.4 MB			
postgres	9.3.5	746b819f315e	4 days ago
213.4 MB			
postgres	latest	746b819f315e	4 days ago
213.4 MB			

Si può restringere la lista indicando nome del repository ed un eventuale tag.

```
$ docker images java
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java	8	308e519aac60	6 days ago	
824.5 MB				
java	7	493d82594c15	3 months ago	
656.3 MB				
java	latest	2711b1d6f3aa	5 months ago	
603.9 MB				

```
$ docker images java:8
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java	8	308e519aac60	6 days ago	
824.5 MB				

Docker build

```
docker build [OPTIONS] PATH | URL | -
```

Questo comando costruisce un'immagine a partire da un Dockerfile e un contesto, è possibile indicare un PATH o un URL che punta ad un repository git.

Ad esempio:

```
$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz

Downloading context: http://server/ctx.tar.gz [=====] 240
B/240 B

Step 1/3 : FROM busybox
--> 8c2e06607696

Step 2/3 : ADD ctx/container.cfg /
--> e7829950cee3

Removing intermediate container b35224abf821

Step 3/3 : CMD /bin/ls
--> Running in fbc63d321d73
--> 3286931702ad

Removing intermediate container fbc63d321d73

Successfully built 377c409b35e4
```

docker import

```
docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]
```

Importa il contenuto di un archivio tar al fine di creare un'immagine.

Ad esempio:

```
$ docker import http://example.com/exampleimage.tgz
```

docker save

```
docker save [OPTIONS] IMAGE [IMAGE...]
```

Salva una o più immagini in un archivio tar.

docker ps

```
docker ps [OPTIONS]
```

Questo comando elenca tutti i container in esecuzione nella macchina. Indicando id, nome, stato, processo e altre informazioni utili.

Tramite il flag -a verranno mostrati anche i container che non sono stati avviati, poiché di default il comando mostra solo quelli in esecuzione.

Ad esempio:

```
$ docker ps -a
```

docker inspect

```
docker inspect [OPTIONS] NAME|ID [NAME|ID...]
```

Restituisce un array JSON contenente tutte le informazioni relative ad un oggetto docker (valido sia per i container che i network).

docker create

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Crea un container a partire da un'immagine, attraverso i numerosi flag è possibile specificare molti aspetti relativi al comportamento del container.

In questo esempio viene creato e successivamente avviato un container.

```
$ docker create -t -i fedora bash  
6d8af538ec541dd581ebc2a24153a28329acb5268abe5ef868c1f1a261221752  
$ docker start -a -i 6d8af538ec5  
bash-4.2#
```

docker run

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Docker run è un comando simile a docker create, ma oltre a creare il container specificato lo avvia usando il comando specificato.

Tramite il flag `docker run --network=<NETWORK>` è possibile connettere direttamente il container ad una rete personalizzata.

```
$ docker run --network=isolated_nw -itd --name=container3 busybox  
8c1a0a5be480921d669a073393ade66a3fc49933f08bcc5515b37b8144f6d47c
```

Nell'esempio seguente viene eseguito un container creato da un'immagine di debian e tramite il flag -it docker alloca una pseudo-TTY connessa al container. Così facendo si ottiene una bash interattiva collegata al container.

```
$ docker run --name test -it debian  
root@d6c0fe130dba:/#
```

docker rm

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

Elimina il container specificato. Il container non deve essere in esecuzione.

docker start

```
docker start [OPTIONS] CONTAINER [CONTAINER...]
```

Avvia un container.

docker stop

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

Ferma un container.

docker attach

```
docker attach [OPTIONS] CONTAINER
```

Collega l'emulatore terminale corrente agli stream del container.

docker cp

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-  
docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
```

Copia contenuti fra il container e il file system locale.

docker exec

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Esegue un comando all'interno del container restituendone l'output.

Ad esempio:

```
$ docker exec container1 ifconfig  
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02  
          inet addr:172.17.0.2  Bcast:0.0.0.0  Mask:255.255.0.0  
            inet6 addr: fe80::42:acff:fe11:2/64 Scope:Link  
              UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1  
              RX packets:16 errors:0 dropped:0 overruns:0 frame:0  
              TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
```

```
    collisions:0 txqueuelen:0
    RX bytes:1296 (1.2 KiB) TX bytes:648 (648.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

docker export

```
docker export [OPTIONS] CONTAINER
```

Esporta il container in un archivio tar, che potrà successivamente essere importato tramite il comando import.

2.5 | NETWORK

In Docker è possibile definire delle LAN e specificarne il comportamento e dettagli. Una volta fatto è possibile usare queste reti per connettere i container fra loro. Di seguito vedremo come gestire, creare nuove reti e quali sono quelle preesistenti.

2.5.1 Le reti di base

Una volta installato Docker l'utente troverà a sua disposizione tre reti pre configurate: bridge, host e none.

- **Bridge:** è un network che collega il container alla rete interna di Docker che a sua volta è connessa alla macchina fisica;
- **Host:** collega direttamente il container alla rete della macchina fisica;
- **None:** è una rete vuota, non connette il container ad alcuna interfaccia.

2.5.2 I comandi

Docker network ls

```
docker network ls [OPTIONS]
```

Riporta la lista dei network presenti nel proprio sistema.

Ad esempio:

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER
7fca4eb8c647	bridge	bridge
9f904ee27bf5	none	null
cf03ee007fb4	host	host

docker network inspect

```
docker network inspect [OPTIONS] NETWORK [NETWORK...]
```

Riporta informazioni dettagliate relative alla rete sotto forma di array JSON.

Ad esempio:

```
$ docker network inspect isolated_nw
```

```
[  
  {  
    "Name": "isolated_nw",  
    "Id": "1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",  
    "Scope": "local",  
    "Driver": "bridge",  
    "IPAM": {  
      "Driver": "default",  
      "Config": [  
        {  
          "Subnet": "172.21.0.0/16",  
          "Gateway": "172.21.0.1/16"  
        }  
      ]  
    },  
    "Containers": {}  
  },  
  {  
    "Name": "host",  
    "Id": "cf03ee007fb4",  
    "Scope": "host",  
    "Driver": "host",  
    "IPAM": {},  
    "Containers": {}  
  }]
```

```
        "Options": {},
        "Labels": {}
    }
]
```

docker network create

```
docker network create [OPTIONS] NETWORK
```

Crea una nuova rete. Durante questa fase si specifica anche quale driver viene utilizzato dal network; il più comune è il driver bridge che permette di creare una rete simile al bridge di default ma più personalizzabile, in particolare è possibile isolare una rete di questo tipo dalla macchina fisica.

In fase di creazione della macchina è possibile assegnare una subnet ed un gateway, se non viene fatto verranno assegnati in automatico.

Il gateway viene gestito in maniera automatica da Docker.

Nell'esempio seguente viene creata una nuova rete con driver bridge e successivamente viene ispezionata.

```
$ docker network create --driver bridge isolated_nw
1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b

$ docker network inspect isolated_nw
[
    {
        "Name": "isolated_nw",
        "Id": "1196a4c5af43a21ae38ef34515b6af19236a3fc48122cf585e3f3054d509679b",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Config": [
                {
                    "Subnet": "172.21.0.0/16",
                    "Gateway": "172.21.0.1/16"
                }
            ]
        }
    }
]
```

```
        }
    ],
},
"Containers": {},
"Options": {},
"Labels": {}
}
]

$ docker network ls

NETWORK ID      NAME      DRIVER
9f904ee27bf5    none      null
cf03ee007fb4    host      host
7fca4eb8c647    bridge    bridge
c5ee82f76de3   isolated_nw  bridge
```

docker network prune

```
docker network prune [OPTIONS]
```

Elimina tutti i network inutilizzati.

docker network connect

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

Connette un container alla rete, creando una nuova interfaccia ed assegnando un ip adatto.

Questo comando può essere eseguito a caldo, ossia anche se il container è in esecuzione.

docker network disconnect

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

Disconnette la rete dal container, eliminando l'interfaccia relativa.

2.6 LA FORMATTAZIONE

Utilizzando la sintassi del Go è possibile formattare l'output di alcuni comandi come *docker ps* o *docker inspect*.

Ad esempio, nel caso del comando *docker inspect*, è possibile parserizzare l'output in JSON in modo da estrarre una singola stringa contenente l'informazione che ci serve.

Quest'operazione viene effettuata aggiungendo il flag *-f* o *--format* seguito dalla sintassi di Go.

Nell'esempio seguente viene lanciato un comando che permette di estrapolare tutti gli indirizzi ip di un container.

```
$ docker inspect --format='{{range  
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $INSTANCE_ID  
10.0.1.5  
10.0.2.3  
10.0.3.7
```

Questa funzionalità di Docker si è rivelata utilissima durante lo sviluppo del software, poiché ha sgravato lo sviluppatore dalle problematiche legate all'estrazione e alla parserizzazione delle informazioni necessarie al funzionamento dello stesso.

3 ALTRI STRUMENTI, LINGUAGGI E LIBRERIE UTILIZZATE

3.1 C++14



Per lo sviluppo del tool è stato scelto di utilizzare il C++ nella sua versione 2014.

Si tratta di un linguaggio nato nel 1983 dalla mente di Bjarne Stroustrup e standardizzato nel 1998 come un miglioramento del C. Esso mantiene la potenza del C ma al contrario di quest'ultimo è fortemente tipizzato ed orientato agli oggetti, rendendolo molto più adatto allo sviluppo di software più complessi e moderni.

Il C++ ha ispirato ed influenzato molti altri linguaggi di uso comune, fra questi Java.

Nel tempo lo standard C++ è stato aggiornato tre volte nel 2003 (C++03), nel 2011 (C++11) e nel 2014 (C++14) è previsto un ulteriore aggiornamento alla fine del 2017.

Grazie agli ultimi aggiornamenti dello standard, in particolare quello del 2011, e al nuovo sistema che prevede un aggiornamento ogni 3 anni, il C++, dopo una piccola fase di stallo, è tornato ad essere un linguaggio moderno, più facile da imparare e con molte funzionalità che favoriscono lo sviluppo software.

Inoltre grazie ai numerosi compilatori disponibili, un applicativo C++ è compilabile ed eseguibile in numerosi sistemi operativi ed architetture.

Oltre a questi motivi, la principale ragione per la quale è stato scelto il C++ per lo sviluppo del tool è che il C++, come il C, è in grado di interfacciarsi direttamente con il kernel del sistema operativo ed utilizzare puntatori alla memoria. Caratteristica indispensabile al fine di utilizzare la CLI di Docker nel modo più efficiente possibile.

3.2 QT5



3.2.1 Panoramica

Per lo sviluppo dell’interfaccia grafica è stata scelta la libreria QT in versione 5.9.

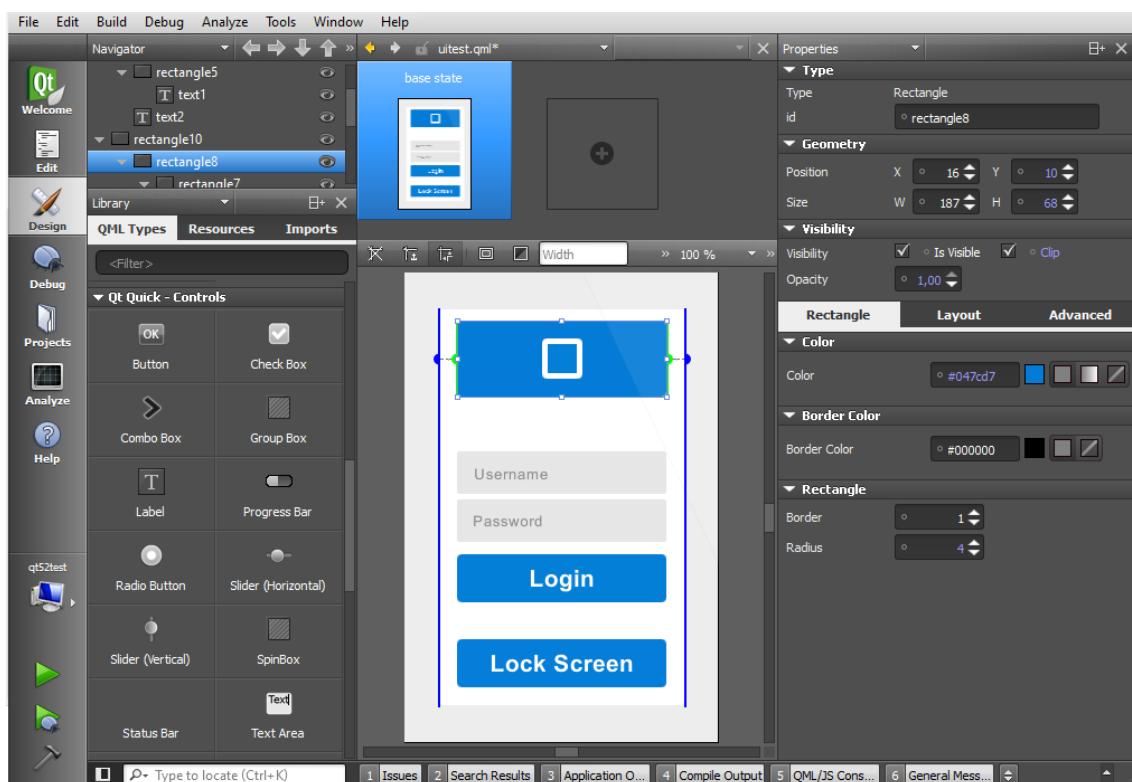
Qt5 è l’ultima versione della libreria multiplattaforma QT (pronuncia “cute” ossia carino in inglese) orientata allo sviluppo di applicativi dotata di interfacce grafiche.

Qt usa il linguaggio C++ standard, ma esistono interfacce per Java, Python, C, Perl e PHP.

Qt si basa sul concetto di widget, ossia un singolo elemento dell’interfaccia, che uniti fra loro compongono l’interfaccia grafica del software per intero.

Qt è molto potente, infatti oltre che occuparsi solamente dell’interfaccia grafica include altre utili funzionalità come la gestione dei processi e l’accesso ai database.

Qt inoltre è molto supportato ed è compatibile con tutti i sistemi e architetture, proprio come lo standard C++.



Qt è corredato di un Toolkit completo che facilita notevolmente lo sviluppo. Tale Toolkit è infatti provvisto di un'IDE chiamata Qt Creator che è provvista di designer grafico, strumento essenziale durante la progettazione dell'interfaccia grafica.

3.2.2 Com'è composto un widget

Ogni widget, inclusa la finestra principale è composto da tre file:

- **Un file .ui:** Il file .ui viene manipolato attraverso il tool grafico incluso nelle Qt. Si tratta di un file xml che definisce gli elementi grafici del widget;
- **Un file .h:** si tratta dell'header che specifica la classe che definirà il comportamento del widget;
- **Un file .cpp:** è l'implementazione della classe che definisce il comportamento del widget.

Esempio di file ui:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>dialogAddNet</class>
<widget class="QDialog" name="dialogAddNet">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>245</width>
```

```

        <height>70</height>
    </rect>
</property>
<property name="minimumSize">
    <size>
        <width>245</width>
        <height>70</height>
    </size>
</property>
<property name="maximumSize">
    <size>
        <width>245</width>
        <height>70</height>
    </size>
</property>
<property name="windowTitle">
    <string>Dialog</string>
</property>
<layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
        <widget class="QLabel" name="label_2">
            <property name="sizePolicy">
                <sizepolicy hsizeType="Minimum" vsizeType="Preferred">
                    <horzStretch>0</horzStretch>
                    <vertStretch>0</vertStretch>
                </sizepolicy>
            </property>
            <property name="text">
                <string>Network</string>
            </property>
        </widget>
    </item>
    <item row="0" column="1">
        <widget class="QComboBox" name="comboBox">
            <property name="sizePolicy">
                <sizepolicy hsizeType="MinimumExpanding" vsizeType="Fixed">
                    <horzStretch>0</horzStretch>
                    <vertStretch>0</vertStretch>
                </sizepolicy>
            </property>
        </widget>
    </item>
    <item row="1" column="0" colspan="2">
        <widget class="QDialogButtonBox" name="buttonBox">
            <property name="orientation">
                <enum>Qt::Horizontal</enum>
            </property>
            <property name="standardButtons">
                <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
            </property>
        </widget>
    </item>
</layout>
</widget>
<resources/>
<connections>
    <connection>
        <sender>buttonBox</sender>
        <signal>accepted()</signal>
        <receiver>dialogAddNet</receiver>
    </connection>
</connections>

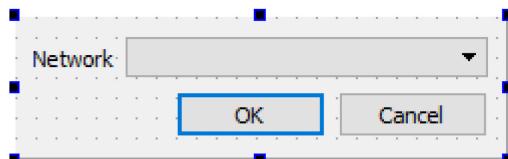
```

```

<slot>accept()</slot>
<hints>
  <hint type="sourcelabel">
    <x>248</x>
    <y>254</y>
  </hint>
  <hint type="destinationlabel">
    <x>157</x>
    <y>274</y>
  </hint>
</hints>
</connection>
<connection>
  <sender>buttonBox</sender>
  <signal>rejected()</signal>
  <receiver>dialogAddNet</receiver>
  <slot>reject()</slot>
  <hints>
    <hint type="sourcelabel">
      <x>316</x>
      <y>260</y>
    </hint>
    <hint type="destinationlabel">
      <x>286</x>
      <y>274</y>
    </hint>
  </hints>
</connection>
</connections>
</ui>

```

Che definisce il seguente widget...



Esempio di header:

```

#ifndef DIALOGADDNET_H
#define DIALOGADDNET_H

#include <QDialog>
#include "manager.h"

namespace Ui {
class dialogAddNet;
}

class dialogAddNet : public QDialog
{
    Q_OBJECT

public:

```

```

    explicit dialogAddNet(QWidget *parent = 0, QString contId =
"contId");
~dialogAddNet();

public slots:
    void apply();
    void abort();

signals:
    void refresh();

private:
    Ui::dialogAddNet *ui;
    manager* uiMan;
    void popComboBox();

    void setupBtns();

    QString contId;
};

#endif // DIALOGADDNET_H

```

Esempio di file .cpp:

```

#include <iostream>
#include "dialogaddnet.h"
#include "ui_dialogaddnet.h"

dialogAddNet::dialogAddNet(QWidget *parent, QString contId) :
    QDialog(parent),
    ui(new Ui::dialogAddNet)
{
    ui->setupUi(this);
    this->setWindowFlags(this->windowFlags() &
~Qt::WindowContextHelpButtonHint);
    setWindowTitle(tr("Connect Network"));

    uiMan = new manager();
    this->contId = contId;

    setupBtns();
    popComboBox();
}

dialogAddNet::~dialogAddNet()
{
    delete ui;
}

```

```

}

void dialogAddNet::popComboBox() {
    vector<string> nets = uiMan->readNetworksNames();

    for (vector <string>::iterator it=nets.begin(); it != nets.end();
++it) {
        QString name = QString::fromStdString(*it);
        if (name != "" && name != "" && name != "bridge" && name !=
"host" && name != "none") {
            ui->comboBox->addItem(name);
            //cout << *it << endl;
        }
    }
}

void dialogAddNet::setupBtns() {
    connect(ui->buttonBox, SIGNAL (accepted()), this, SLOT (apply()));
    connect(ui->buttonBox, SIGNAL (rejected()), this, SLOT (abort()));
}

void dialogAddNet::apply() {
    QString net = ui->comboBox->currentText();
    uiMan->getDocker()->connect(contId.toStdString(), net.toStdString());
    string contIdS=contId.toStdString();
    vector <string> iFaces = uiMan->readContIfs(contIdS);

    ...
    ...
    emit refresh();
    delete this;
}

void dialogAddNet::abort() {
    delete this;
}

```

3.3 CLION



Per lo sviluppo del software è stato scelto l'IDE C++ CrossPlatform JetBrains Lion. CLion è stato sviluppato dalla stessa compagnia che ha sviluppato IntelliJ Idea, IDE alla base di Android Studio.

CLion fa uso del compilatore GCC e del tool CMAKE presenti nel sistema, assiste il programmatore durante lo sviluppo grazie al sistema di auto completamento, supporta git e aiuta nella navigazione dei numerosi file del progetto.

CLion, per la compilazione, usa un singolo file CMakeLists.txt, liberamente modificabile dallo sviluppatore.

Inoltre JetBrains offre una licenza gratuita a tutti gli studenti.

Per tali motivi CLion è stato scelto come IDE principale.

A screenshot of the CLion IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar shows the project name "quantlib" and files "array.hpp", "blackcalculator.cpp", and "CMakeLists.txt". The left sidebar displays the "Project" structure with folders like "quantlib", "build", "config", "Docs", and "Examples". The main editor area shows the code for "array.hpp":

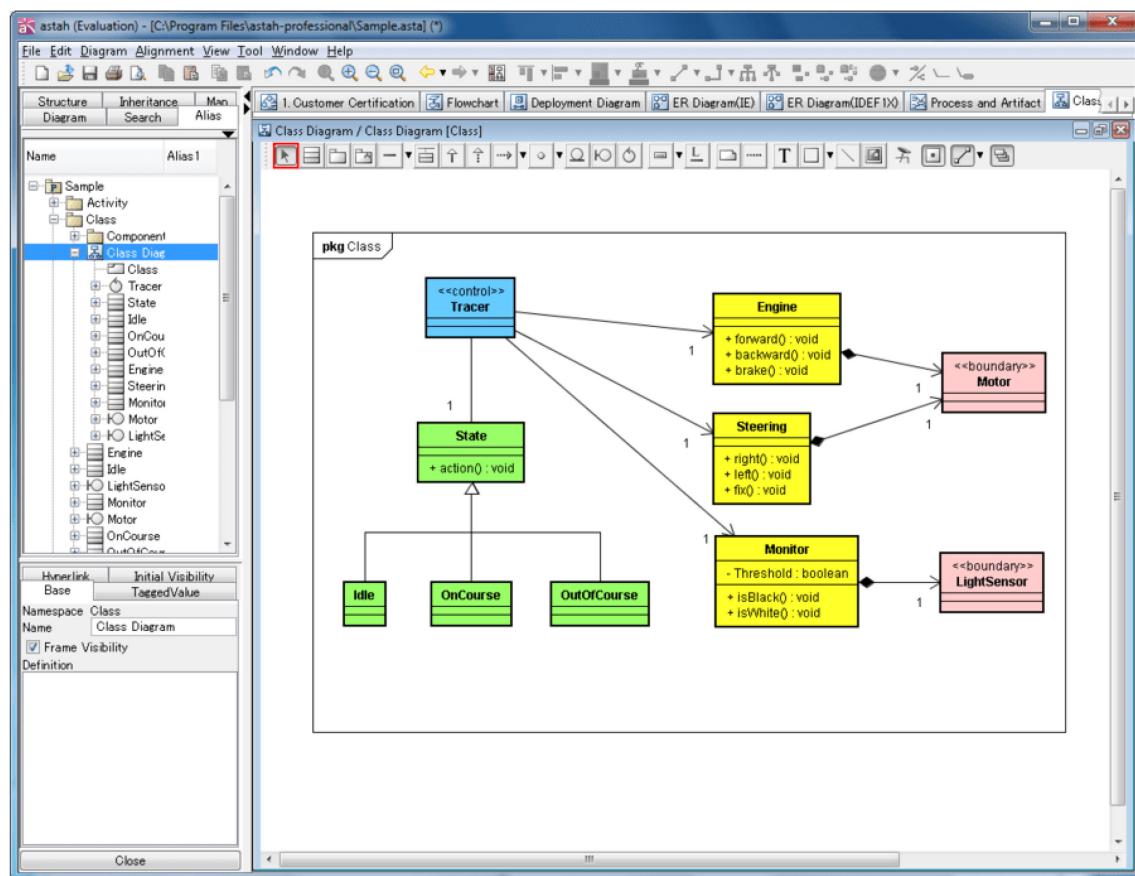
```
231
232     inline Array::Array(const Disposable<Array>& from)
233     : data_(new Real*[n_(0)]), n_(0) {
234         swap(const_cast<Disposable<Array>&>(from));
235     }
236 }
```

3.4 ASTAH UML



Durante la fase di progettazione è stato necessario creare alcuni schemi UML al fine di comprendere al meglio la struttura del software.

È stato scelto Astah poiché è un editor completo e leggero ed inoltre la licenza per la versione professional è gratuita per tutti gli studenti.



4 IL BACKEND: LA COMUNICAZIONE CON DOCKER

4.1 PREMESSE

Al fine di costruire un software in grado di comunicare con Docker sono state vagliate varie ipotesi.

Esistono infatti delle API ufficiali per Python e Go, ma non nessuno dei due linguaggi si prestava alle esigenze dello sviluppatore (vedi capitolo 3). Inoltre tali API sono pensate per essere utilizzate all'interno di script o piccole utility piuttosto che nel contesto di un software completo.

Delle API (Advanced Programming Interface) sono un set di tool e librerie che permettono allo sviluppatore di interfacciarsi programmaticamente ad un altro sistema software, come un sistema operativo o una piattaforma web. Nel nostro caso dovrebbero permetterci di interfacciarcici con Docker.

Oltre alle suddette API ufficiali ne esistono altre mantenute dalla community e che supportano numerosi linguaggi. Fra i quali il C++. Ciononostante tali API sono state scartate poiché sono poco aggiornate e troppo complesse per le esigenze di sviluppo.

Si è dunque scelto di sfruttare le caratteristiche del C++ che permettono di interfacciarsi direttamente con il sistema operativo e di utilizzare di conseguenza la Docker CLI (Command Line Interface).

La Docker CLI, come visto nel capitolo 2, permette di controllare in maniera completa le componenti di Docker e fornisce informazioni esaustive e facilmente manipolabili.

Pertanto è stata scritta una sorta di mini API che comunicasse con la Docker CLI.

4.2 LA CLASSE DOCKINT

4.2.1 Panoramica

Nella classe dockInt viene definita l'interfaccia con Docker, tale classe si occupa di comunicare con il sistema operativo e dunque con la CLI di Docker.

Questa classe è stata progettata per essere portatile ed indipendente, dunque utilizzabile nell'ambito di qualunque programma che utilizzi le classi di C++14 e non si lega dunque a particolari librerie.

Possono essere istanziati molteplici oggetti di classe container, poiché grazie all'uso della Docker CLI non si corre il rischio di incorrere in conflitti di alcun tipo.

4.2.2 Header

```
#ifndef CONTNET.DockInt_H
#define CONTNET.DockInt_H

#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <memory>

using namespace std;

class dockInt {
private:
    string execCmd(string command);
    bool isValid(string str);
    bool isValidU(string str);
    bool isConflict(string str);
    bool isAlready(string str);
    bool isOverlapping(string str);
public:
    dockInt() {}
    void startDocker();
    void addCont(string name, string netId, bool isRouter);
    string addCont(string name, string netId);
    void delCont(string id);
    string addNet(string name, string subnet, string gw);
    string delNet(string id);
    void connect(string contId, string netId);
    void disconnect(string contId, string netId);
    void deleteAllConts();
    void prune();
    string ifconfig(string contId);
    string routeAdd(string contId, string ip, string via);
    string routeDel(string contId, string ip);
    string route(string contId);
    string inspect(string id);
    string inspect(string id, string format);
    string ps(string mode, bool started);
    string getContImg(string id);
```

```

        string getContStatus(string id);
        string lsNet(string mode);
        string start(string id);
        string stop(string id);
        string ifconfig(string contId, string param);
    };

#endif //CONTNET.DockInt_H

```

Tramite l'header possiamo apprezzare la varietà di funzioni, disponibili allo sviluppatore, che permettono di utilizzare facilmente tutti i comandi utili alla gestione delle reti e container su Docker incluso routing e comunicazione diretta con il container tramite comuni comandi di Linux.

4.2.3 La definizione delle funzioni

Di seguito un estratto della definizione delle funzioni più importanti.

```

string dockInt::execCmd(string command)
{
    command = command + " 2>&1";

    const char *cmd = command.c_str();

    std::array<char, 128> buffer;
    std::string result;

    std::shared_ptr<FILE> pipe(popen(cmd, "r"), pclose);
    if (!pipe)
        throw std::runtime_error("popen() failed!");
    while (!feof(pipe.get()))
    {
        if (fgets(buffer.data(), 128, pipe.get()) != NULL)
            result += buffer.data();
    }

    return result;
}

```

La funzione `string dockInt::execCmd(string)` si occupa di comunicare con il sistema inviando un comando e prelevandone l'output.

A tale scopo viene aperta una pipe che tramite la funzione `popen()` invia il comando al sistema e ne preleva lo standard output.

Allo scopo di semplificare la procedura di rilevazione degli errori lo stderr viene reindirizzato nello stdout tramite l'aggiunta di `2>&1` al comando.

Le funzioni

```
bool isValid(string str);
bool isValidU(string str);
bool isConflict(string str);
bool isAlready(string str);
bool isOverlapping(string str);
```

servono a rilevare alcuni errori che possono accadere in seguito all'immissione di un input sbagliato da parte dell'utente.

Tutte le altre funzioni pubbliche fanno uso della funzione appena descritte per utilizzare la Docker CLI. Vediamone alcune

```
string dockInt::addCont(string name, string netId)
{
    string str = execCmd("docker run --name " + name + " --cap-
add=NET_ADMIN --network=" + netId + " -d -it debian:network /bin/bash");
    if (isAlready(str))
        return "already";
    if (isValid(str) || isValidU(str))
        return "invalid";
    return str;
}

void dockInt::delCont(string id)
{
    execCmd("docker rm " + id);
}

string dockInt::addNet(string name, string subnet, string gw)
{
    string str = execCmd("docker network create -d bridge --gateway " +
gw + " --subnet " + subnet + " " + name);
    if (isAlready(str))
        return "already";
    if (isValid(str))
        return "invalid";
    if (isOverlapping(str))
        return "overlapping";
    return str;
}

string dockInt::delNet(string id)
{
    string ret = execCmd("docker network rm " + id);
    ret.erase(std::remove(ret.begin(), ret.end(), '\n'), ret.end());
```

```

        return ret;
    }

void dockInt::connect(string contId, string netId)
{
    execCmd("docker network connect " + netId + " " + contId);
}

void dockInt::disconnect(string contId, string netId)
{
    execCmd("docker network disconnect " + netId + " " + contId);
}

string dockInt::ifconfig(string contId)
{
    return execCmd("docker exec " + contId + " ifconfig");
}

string dockInt::ifconfig(string contId, string param)
{
    return execCmd("docker exec " + contId + " ifconfig " + param);
}

string dockInt::routeAdd(string contId, string ip, string via)
{
    return execCmd("docker exec " + contId + " ip route add " + ip + " "
via " + via);
}

string dockInt::route(string contId)
{
    return execCmd("docker exec " + contId + " ip route");
}

string dockInt::inspect(string id)
{
    return execCmd("docker inspect " + id);
}

string dockInt::inspect(string id, string format)
{
    return execCmd("docker inspect " + id + " -f " + format);
}

string dockInt::lsNet(string mode)
{

```

```

if (mode == "all")
{
    return execCmd("docker network ls");
}

if (mode == "id")
{
    return execCmd("docker network ls -q");
}

if (mode == "name")
{
    return execCmd("docker network ls --format '{{.Name}}'");
}
}

string dockInt::start(string id)
{
    return execCmd("docker start " + id);
}

string dockInt::stop(string id)
{
    return execCmd("docker stop " + id);
}

void dockInt::startDocker()
{
    execCmd("systemctl start docker");
}

string dockInt::routeDel(string contId, string ip)
{
    return execCmd("docker exec " + contId + " ip route delete " + ip);
}

void dockInt::prune()
{
    execCmd("docker network prune -f");
}

void dockInt::deleteAllConts()
{
    execCmd("docker stop $(docker ps -a -q)");
    execCmd("docker rm $(docker ps -a -q)");
}

```

Tali funzioni fanno tutte uso dei comandi descritti nel capitolo 2, sono semplici e richiedono soltanto pochi parametri per essere chiamate.

Non tutte le funzioni presenti nella classe vengono utilizzate dal tool, poiché esso si limita soltanto ad un utilizzo didattico, pertanto molte operazioni devono essere lasciate allo studente.

È bene notare (vedi capitolo 2 per maggiori dettagli):

- **Il comando `docker exec`:** che ci permette di inviare un comando direttamente al container e prelevarne l'output.
- **Il tag `--format`:** che ci permette di passare in input un template Go al fine di formattare l'output.
- **Il comando “`docker run --name " + name + " --cap-add=NET_ADMIN --network=" + netId + " -d -it debian:network /bin/bash`”:** si tratta di un comando `docker run` a cui sono stati aggiunti vari tag. Il tag `--cap-add=NET_ADMIN` abilita alcune funzionalità di rete essenziali all'interno del container che permettono il routing, ip forwarding e gestione delle interfacce di rete. Il tag `-d` fa sì che il container venga eseguito in background. Il tag `-it` alloca una pseudo-TTY al container per permettere l'avvio di una shell interattiva. `debian:network` è l'immagine customizzata creata appositamente per questo tool.
- **Il comando “`docker network create -d bridge --gateway " + gw + " --subnet " + subnet + " " + name`”:** Le nuove reti vengono create tramite il suddetto comando, viene utilizzato il driver bridge.

4.3 LA CLASSE MANAGER

4.3.1 Panoramica

La classe manager fornisce una façade alle operazioni nella classe dockInt.

Tale classe si occupa di coniugare le esigenze del tool e di Docker. Tale classe infatti si occupa di parserizzare eventuali liste di output, usare comandi di Docker passando come argomento la formattazione necessaria e ad adattare lo scenario creato ad un uso didattico.

Ogni manager include al suo interno un oggetto dockInt e vi si può accedere soltanto tramite il manager.

4.3.2 L'header

```
#ifndef CONTNET_MANAGER_H  
#define CONTNET_MANAGER_H  
  
#include <sstream>
```

```

#include <iostream>
#include <vector>
#include <random>

class manager {
private:
    dockInt* docker;
    vector<string> splitString(const std::string& str, const std::string& delim);
    string createSubNetForExam();
    string createGwForExam(string sub);

public:
    manager();
    dockInt* getDocker();
    vector <string> readContainersId();
    vector <string> readContainerNetworks(string id);
    vector <string> readNetworksId();
    vector <string> readNetworkConts(string id);
    vector <string> readContainersStatus();
    vector <string> readContainersNames();
    vector <string> readNetworksNames();
    vector <string> readContIfs(string id);

    string getNetSubnet(string id);
    string getNetName(string id);
    string addNetExam(string name);
};

#endif //CONTNET_MANAGER_H

```

4.3.3 Definizione delle funzioni

Vediamo quali sono le funzioni più importanti

```

vector <string> manager::readContainersId() {
    string tmp;
    string out;
    out = (docker->ps("id", 0));
    vector<string> ret;

    ret = splitString(out, "\n");

    return ret;
}

vector <string> manager::readContainersStatus() {
    string out;
    out = (docker->ps("status", 0));

```

```

//cout << "call1 : "+ out << endl;
vector<string> ret;

ret = splitString(out, "\n");

return ret;
}

vector <string> manager::readContainersNames() {
    string out;
    out = (docker->ps("names", 0));
    //cout << "call1 : "+ out << endl;
    vector<string> ret;

    ret = splitString(out, "\n");

    return ret;
}

vector <string> manager::readContainerNetworks(string id) {
    string tmp;
    string out;
    out = docker->inspect(id, "'{{range $key, $value := .NetworkSettings.Networks}}{{printf \"%s\\n\" $key}}{{end}}'");
    vector<string> ret;

    ret = splitString(out, "\n");

    return ret;
}

vector <string> manager::readNetworksId() {
    string out;
    out = (docker->lsNet("id"));
    vector<string> ret;
    ret = splitString(out, "\n");
    return ret;
}

vector <string> manager::readNetworksNames() {
    string out;
    out = (docker->lsNet("name"));
    vector<string> ret;
    ret = splitString(out, "\n");
    return ret;
}

```

Queste funzioni di lettura sono state create al fine di ottenere vettori di informazioni da Docker. Per farlo viene chiamata una funzione dell'interfaccia descritta nel paragrafo precedente e spesso viene passato in argomento un template di GO al fine di parserizzare il risultato.

Ad esempio nella funzione `manager::readContainerNetworks()` viene chiamata una funzione `inspect()` formattata con il seguente template '`{{range $key, $value := .NetworkSettings.Networks}}{{printf \"%s\\n\" $key}}{{end}}`'.

Successivamente grazie alla funzione `splitString()` le stringhe ottenute in output da Docker vengono inserite in un vettore.

```
vector<string> manager::splitString(const std::string& str,
                                      const std::string& delim)
{
    try {
        std::vector<std::string> strings;

        std::string::size_type pos = 0;
        std::string::size_type prev = 0;
        while ((pos = str.find(delim, prev)) != std::string::npos)
        {
            strings.push_back(str.substr(prev, pos - prev));
            prev = pos + 1;
        }

        // To get the last substring (or only, if delimiter is not found)
        strings.push_back(str.substr(prev));

        return strings;
    } catch(int e) {
        vector<string> err;
        err[0] = "unsplittable " + e;
        return err;
    }
}
```

La funzione sopra è stata scritta al fine di suddividere la stringa in un vettore tramite l'uso di un delimitatore arbitrario.

```
string manager::createGwForExam(string sub) {
    vector<string> ip = splitString(sub, ".");
    string gw;
    if(ip.size() == 4){
        ip[3] = "60";
        gw = ip[0] + "." + ip[1] + "." + ip[2] + "." + ip[3];
```

```

    }

    else gw = "error";
    return gw;
}

string manager::createSubNetForExam() {
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_int_distribution<int> dist(0, 254);
    int i3 = dist(mt);
    string s3 = to_string(i3);
    string ip = "10.0." + s3 + ".0/24";
    cout << ip;
    return ip;
}

string manager::addNetExam(string name) {
    bool ok = false;
    string res;
    while (!ok){
        string sub = createSubNetForExam();
        string gw = createGwForExam(sub);
        res = docker->addNet(name, sub, gw);
        if (res != "overlapping") ok = true;
    }
    return res;
}

```

Le funzioni definite sopra sono utili a creare un ambiente adatto alla didattica. Inoltre eliminano eventuali impostazioni automatiche effettuate nei network da parte di Docker al fine di lasciare allo studente il compito di configurare delle topologie di rete funzionanti.

Le funzioni `manager::createGwForExam()` e `manager::createSubNetForExam()` si occupano di generare subnet e gateway predefiniti utili durante la chiamata della funzione `addNet()`. Tali subnet e gateway verranno ignorati durante l'uso vero e proprio del tool in quanto gli automatismi di Docker vengono disabilitati, ma sono utili ad identificare facilmente le reti create dal tool che presenteranno tutte subnet della classe 10.0.x.0/24 e gateway 10.0.x.60.

La funzione `manager::addNetExam()` è pubblica e usa le funzioni precedenti al fine di creare una rete adatta agli scopi didattici del tool.

```
vector<string> manager::readContIfs(string id) {
    string out;
    out = (docker->ifconfig(id, "| grep \"eth\" | awk -F'[: ]+' '{ print
$1 }'"));
    cout << out;
    vector<string> ret;
    ret = splitString(out, "\n");
    return ret;
}
```

Questo è un altro esempio della flessibilità della classe dockInt, grazie a questa funzione viene lanciato un comando `ifconfig | grep "eth" | awk -F'[:]+' '{ print $1 }` all'interno del container specificato. Tale comando è in grado di elencare i nomi di tutte le interfacce presenti all'interno del container, che sono tutte nella forma ethX.

5 IL FRONTEND: COME FUNZIONA LA GUI

5.1 INTRODUZIONE

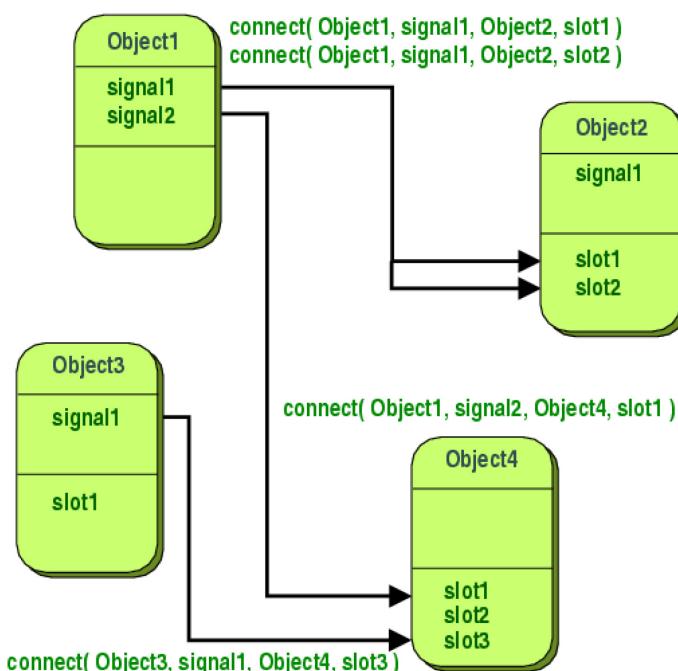
L'interfaccia grafica del tool è formata da più classi derivate da `QWidget` che insieme formano la GUI nel complesso. I vari widget sono in grado di comunicare fra loro al fine di aggiornarsi e con l'interfaccia di Docker descritta in precedenza.

5.2 INTEGRAZIONE CON DOCKER

Al fine di permettere una comunicazione efficace con la miniAPI inclusa nel software, ogni widget include un puntatore ad un oggetto manager che garantisce l'accesso a tutte le funzioni necessarie e, eventualmente, anche all'interfaccia stessa.

5.3 LA COMUNICAZIONE FRA I WIDGET

Per comunicare fra loro i widget utilizzano il sistema incluso in Qt dei `slot()` e dei `signal()`.



Uno slot è una funzione appartenente a un widget e risponde al segnale a cui viene collegato tramite la funzione `connect()` che prende in argomento l'oggetto che può lanciare il segnale, il segnale, l'oggetto che lo riceve e lo slot assegnato.

Grazie a questi potenti strumenti è possibile far comunicare i widget fra loro senza l'utilizzo di puntatori, semplificando notevolmente lo sviluppo.

Vediamo un esempio, sia la classe:

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

Dotata di uno slot `setValue()` e un segnale `valueChanged()`.

La funzione `setValue()` è così definita

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

Si nota che alla fine delle operazioni viene emesso il segnale tramite la parola chiave `emit`.

Di seguito un possibile uso della classe

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);      // a.value() == 12, b.value() == 12
b.setValue(48);      // a.value() == 12, b.value() == 48
```

Si può notare l'uso della funzione `QObject::connect()` tramite la quale vengono connessi oggetti, slot e segnali fra loro. Di conseguenza i due oggetti riescono a mantenere sincronizzato il valore della loro variabile.

Questo sistema è stato usato in maniera ricorrente nello sviluppo del tool.

5.4 LA MAINWINDOW

Al fine di chiarire meglio la struttura seguita durante lo sviluppo dei widget vedremo insieme la struttura della mainWindow che è rappresentativa di quella degli altri widget.

Tale widget fornisce bottoni per aggiungere container e reti, permette inoltre di rimuovere tutti i container e tutte le reti inutilizzate.

Presenta due tab, nella prima vengono inseriti i widget che rappresentano ogni container nella seconda quelli che rappresentano le reti.

Può istanziare i widget: QContainerWidgetExam, QNetworkWidget

5.4.1 File .ui

In questo file sono stati definiti gli elementi grafici di default tramite l'editor grafico messo a disposizione dal ToolKit QT.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>1077</width>
<height>671</height>
</rect>
</property>
<property name="windowTitle">
<string>MainWindow</string>
</property>
<widget class="QWidget" name="centralWidget">
<layout class="QGridLayout" name="gridLayout">
<item row="0" column="1">
<widget class="QTabWidget" name="tabWidget">
<property name="currentIndex">
<number>0</number>
</property>
<widget class="QWidget" name="tabCont">
<attribute name="title">
<string>Containers</string>
```

```

</attribute>
<layout class="QGridLayout" name="gridLayout_4">
    <item row="0" column="0" colspan="2">
        <widget class="QScrollArea" name="scrollArea">
            <property name="widgetResizable">
                <bool>true</bool>
            </property>
            <widget class="QWidget" name="scrollAreaWidgetContentsCont">
                <property name="geometry">
                    <rect>
                        <x>0</x>
                        <y>0</y>
                        <width>1033</width>
                        <height>560</height>
                    </rect>
                </property>
                <layout class="QGridLayout" name="gridLayoutCont"/>
            </widget>
        </item>
        <item row="1" column="0">
            <widget class="QCommandLinkButton" name="addBtn">
                <property name="sizePolicy">
                    <sizepolicy hsizeType="MinimumExpanding"
vsizetype="Preferred">
                        <horzstretch>0</horzstretch>
                        <verzstretch>0</verzstretch>
                    </sizepolicy>
                </property>
                <property name="text">
                    <string>ADD</string>
                </property>
            </widget>
        </item>
        <item row="1" column="1">
            <widget class="QCommandLinkButton" name="delAllContButton">
                <property name="text">
                    <string>Delete All</string>
                </property>
            </widget>
        </item>
    </layout>
</widget>
<widget class="QWidget" name="tabNet">
    <attribute name="title">
        <string>Networks</string>
    </attribute>
    <layout class="QGridLayout" name="gridLayout_2">
        <item row="0" column="0" colspan="3">
            <widget class="QScrollArea" name="scrollAreaNet">
                <property name="widgetResizable">
                    <bool>true</bool>
                </property>
                <widget class="QWidget" name="scrollAreaWidgetContentsNet">
                    <property name="geometry">
                        <rect>
                            <x>0</x>
                            <y>0</y>
                            <width>98</width>

```

```

        <height>28</height>
    </rect>
</property>
<layout class="QGridLayout" name="gridLayoutNet"/>
</widget>
</widget>
</item>
<item row="1" column="2">
<widget class="QCommandLinkButton" name="commandLinkButton">
<property name="text">
<string>Prune</string>
</property>
</widget>
</item>
<item row="1" column="0" colspan="2">
<widget class="QCommandLinkButton" name="addBtnNet">
<property name="sizePolicy">
<sizepolicy hsizetype="MinimumExpanding"
vsizetype="Preferred">
<horstretch>0</horstretch>
<verstretch>0</verstretch>
</sizepolicy>
</property>
<property name="text">
<string>ADD</string>
</property>
</widget>
</item>
</layout>
</widget>
</widget>
</item>
</layout>
</widget>
</widget>
</ui>
```

5.4.2 Header

Nel seguente header possiamo vedere l'inclusione del puntatore alla classe manager che permette la comunicazione con Docker e l'allocazione dei tre slot `addCont()`, `addNet()` e `refresh()` che permettono l'aggiunta di nuovi container, reti e l'aggiornamento dell'interfaccia.

Nelle funzioni `popContList()` e `popNetList()` viene realizzata la creazione dei widget dedicati ai container e network già presenti nella macchina.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "manager.h"
#include "qcontainerwidget.h"
```

```

#include "qcontainerwidgetexam.h"
#include "dialognewcont.h"
#include "ui_mainwindow.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    Ui::MainWindow* getUi();
    QWidget* getP();
    ~MainWindow();

public slots:
    void addCont();
    void addNet();
    void refresh();

protected:
    manager* uiMan;

private:
    Ui::MainWindow *ui;
    void popContList();
    void popNetList();
    QWidget* p;

    void clearWidgets(QLayout * layout);
};

#endif // MAINWINDOW_H

```

5.4.3 Implementazione

Nella seguente implementazione è possibile evincere l'uso della funzione `connect()` e l'implementazione delle funzioni più importanti.

```

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle(tr("Vlad II"));
}

```

```

uiMan = new manager();
p = parent;

popContList();
popNetList();

connect(ui->addBtn, SIGNAL(released()), this, SLOT(addCont()));
connect(ui->addBtnNet, SIGNAL(released()), this, SLOT(addNet()));
}

void MainWindow::popContList() {
    vector <string> conts = uiMan->readContainersId();
    vector <string> status = uiMan->readContainersStatus();
    vector <string> names = uiMan->readContainersNames();
    vector <string>::iterator it2=status.begin();
    vector <string>::iterator it3=names.begin();

    for (vector <string>::iterator it=conts.begin(); it != conts.end();
++it){
        QString id = QString::fromStdString(*it);
        QString st = QString::fromStdString(*it2);
        QString name = QString::fromStdString(*it3);
        if(id != "") {
            QContainerWidgetExam* elem = new QContainerWidgetExam(this,
name, id, st);
            ui->gridLayoutCont->addWidget(elem);
            //cout << *it << endl;
        }
        ++it2;
        ++it3;
    }
}

void MainWindow::popNetList() {
    vector <string> nets = uiMan->readNetworksId();
    vector <string> names = uiMan->readNetworksNames();
    vector <string>::iterator it2=names.begin();

    for (vector <string>::iterator it=nets.begin(); it != nets.end();
++it){
        QString id = QString::fromStdString(*it);
        QString name = QString::fromStdString(*it2);
        if(id != "" && name != "" && name != "none" && name != "bridge"
&& name != "host") {
            QNetworkWidget* elem = new QNetworkWidget(this, id, name);
            connect(elem, SIGNAL(refresh()), this, SLOT(refresh()));
            ui->gridLayoutNet->addWidget(elem);
        }
    }
}

```

```

        ++it2;
    }
    ui->gridLayoutNet->setAlignment(Qt::AlignTop);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::addCont() {
    dialogNewCont* win = new dialogNewCont(this);
    connect(win, SIGNAL(refresh()), this, SLOT(refresh()));
    win->show();
    //clearWidgets(ui->gridLayoutCont);
}

void MainWindow::addNet() {
    dialogNewNet* win = new dialogNewNet(this);
    connect(win, SIGNAL(refresh()), this, SLOT(refresh()));
    win->show();
}

Ui::MainWindow* MainWindow::getUi() {
    return ui;
}

QWidget *MainWindow::getP() {
    return p;
}

void MainWindow::refresh() {
    clearWidgets(ui->gridLayoutNet);
    clearWidgets(ui->gridLayoutCont);
    popContList();
    popNetList();
}

void MainWindow::clearWidgets(FlowLayout* layout) {
    if (! layout)
        return;
    while (auto item = layout->takeAt(0)) {
        delete item->widget();
        clearWidgets(item->layout());
    }
}

```

5.5 PANORAMICA DEI WIDGET

QContainerWidgetExam

Questo è il widget che fornisce all'utente tutti gli strumenti e le informazioni relative al singolo container. Permette di cambiare lo stato del container, aprire un terminale, eliminare il container e connetterlo a più reti. Fornisce informazioni sulla tabella di routing.

Viene agganciato alla prima tab della *mainWindow*. Ne vengono creati tanti quanti sono i container presenti nella macchina. Può istanziare dei widget *DialogAddNet* e *QContConnectedNetWidget*.

QNetworkWidget

Questo è il widget che fornisce all'utente le informazioni relative al singolo network e permette di eliminarlo.

Viene agganciato alla seconda tab della *mainWindow*. Ne vengono creati tanti quanti sono i network presenti nella macchina.

QContConnectedNetWidget

Molto simile al *QNetworkWidget*, fornisce le informazioni relative al singolo network connesso ad un container e permette di effettuare la disconnessione.

Viene agganciato al *QContainerWidgetExam*. Ne vengono creati tanti quante sono le connessioni del container in esame.

DialogNewCont e DialogNewNet

Sono i widget che permettono di creare nuovi container e network. Vengono istanziati dalla *mainWindow*.

DialogAddNet

È il widget che serve a connettere un network ad un container. Viene istanziato da *QContainerWidgetExam*.

6 REQUISITI, INSTALLAZIONE E DISTRIBUZIONE

I requisiti del tool, come da obiettivo, sono molto modesti.

- Sistema Linux dotato di kernel avente versione superiore o uguale alla 3.10. Sono supportati sia i sistemi a 32 che a 64 bit;
- CPU che raggiunga almeno 1GHz;
- 1 Gb di RAM.

Il tool dovrebbe dunque supportare molte configurazioni, anche quelle dotate di hardware molto datato.

Le dipendenze essenziali sono:

- Docker CE v17;
- La presenza di un Desktop Environment;
- L'emulatore terminale Konsole.

Per effettuare l'installazione verrà fornito uno script che installerà il tool e le eventuali dipendenze mancanti.

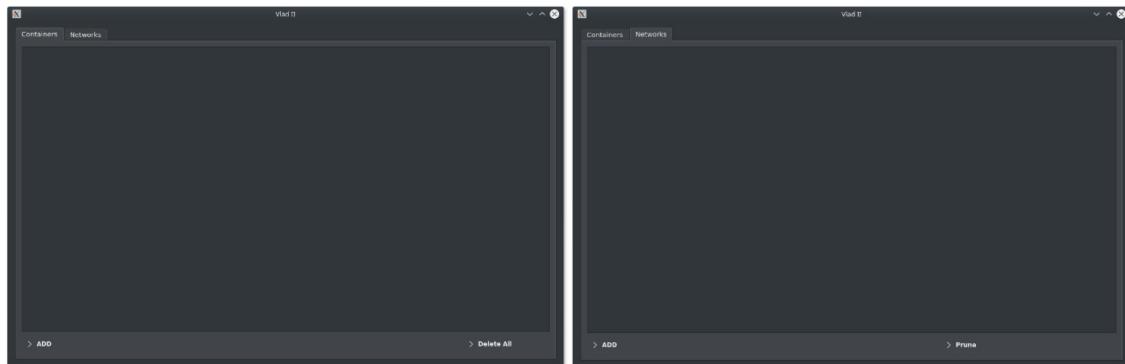
Inoltre il tool verrà distribuito all'interno di una macchina virtuale e un sistema live basati su Debian sviluppati ad hoc per gli scopi didattici del tool.

7 GUIDA ALL'USO

7.1 AVVIO

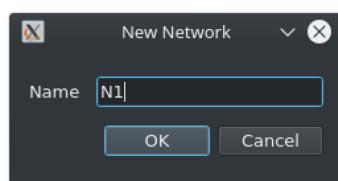
Una volta avviato il tool troveremo di fronte a noi due tab, la prima sarà popolata dai container presenti della macchina, se non ve ne sarà alcuno sarà vuota.

La seconda tab sarà popolata dai network presenti nella macchina, anche in questa se non ve ne sarà alcuno sarà vuota.

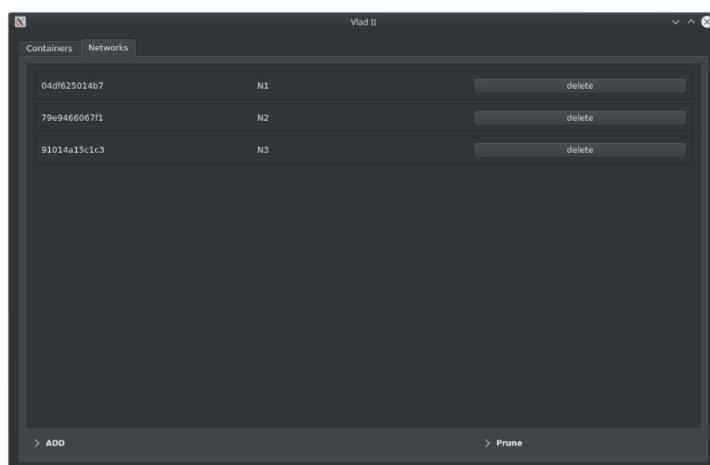


7.2 AGGIUNGERE UN NETWORK

Recarsi nella tab network e cliccare su “ADD”, si aprirà la seguente finestra:

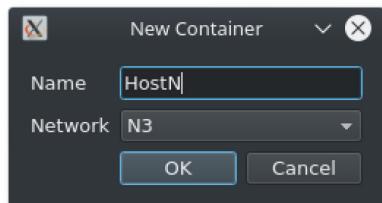


Inserire il nome della rete e premere ok, il tool creerà un nuovo network compatibile con Docker e lo aggiungerà a quelli presenti nella rispettiva tab.



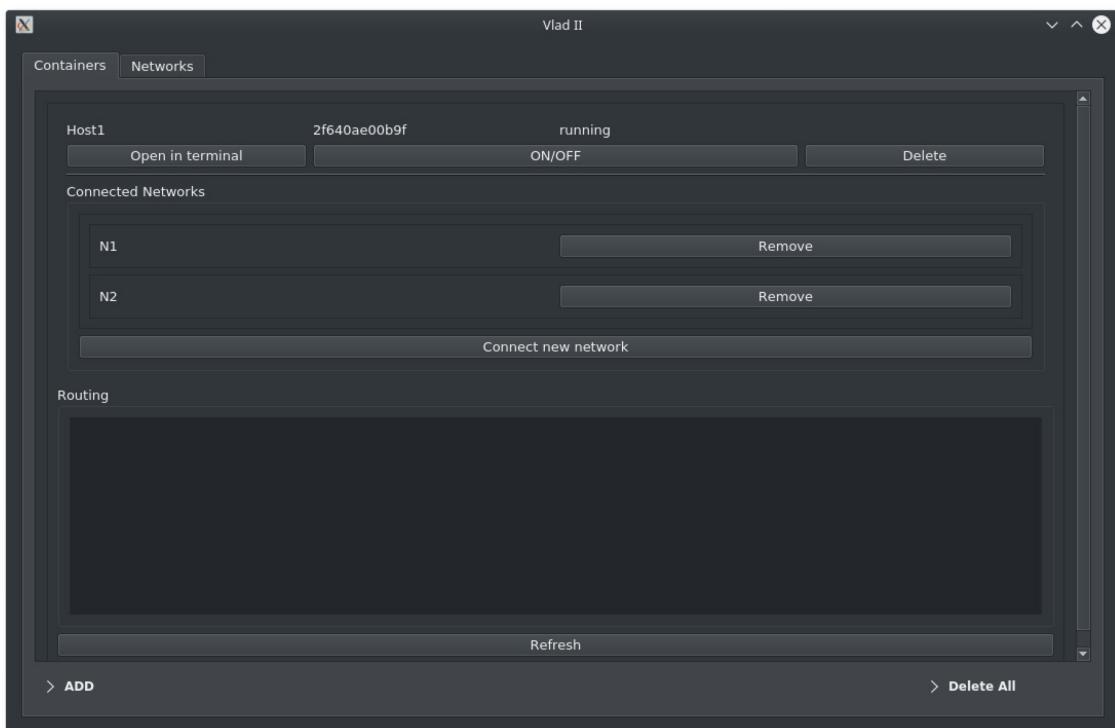
7.3 AGGIUNGERE UN NUOVO CONTAINER (NODO)

Recarsi nella tab container e cliccare su “ADD”, si aprirà la seguente finestra:



Scegliere dunque un nome per il container e la prima rete dall’elenco, scegliendo l’opzione none il container non verrà connesso ad alcuna rete.

Il tool creerà il nuovo nodo e lo aggiungerà all’elenco della tab.



7.4 CONNETTERE E RIMUOVERE NETWORK DAI CONTAINER

Nella sezione “Connected network” è possibile vedere a quali reti è connesso il container, ad ogni rete corrisponde un’interfaccia di rete.

È possibile disconnettere l’interfaccia tramite il tasto remove.

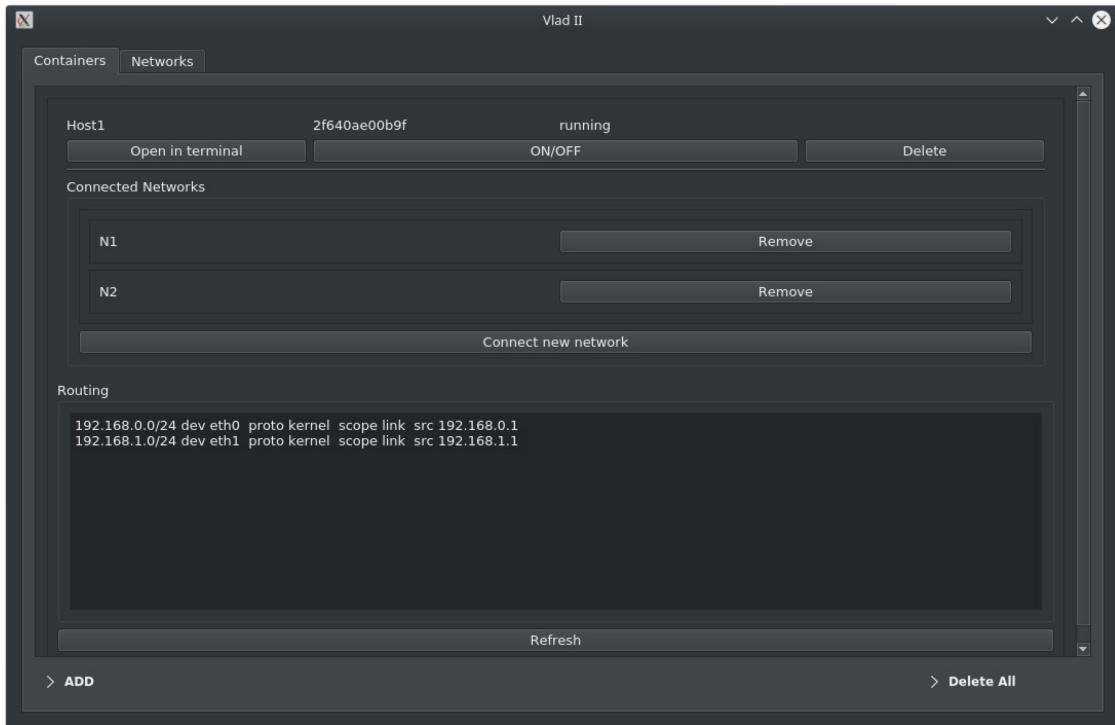
Per collegare un nuovo network cliccare su “Connect new network”, si aprirà la seguente finestra:



Scegliere la rete dall’elenco e premere ok, la rete verrà aggiunta nell’apposita sezione.

Tali operazioni possono essere svolte a caldo, ossia a container avviato, ma sarà comunque necessario eseguire nuovamente la configurazione del nodo.

7.5 L’INTERFACCIA DEDICATA AL CONTAINER



La GUI fornisce informazioni riguardanti lo stato del container, il nome e l’ID.

Tramite il tasto “ON/OFF” è possibile spegnere e riaccendere il container.

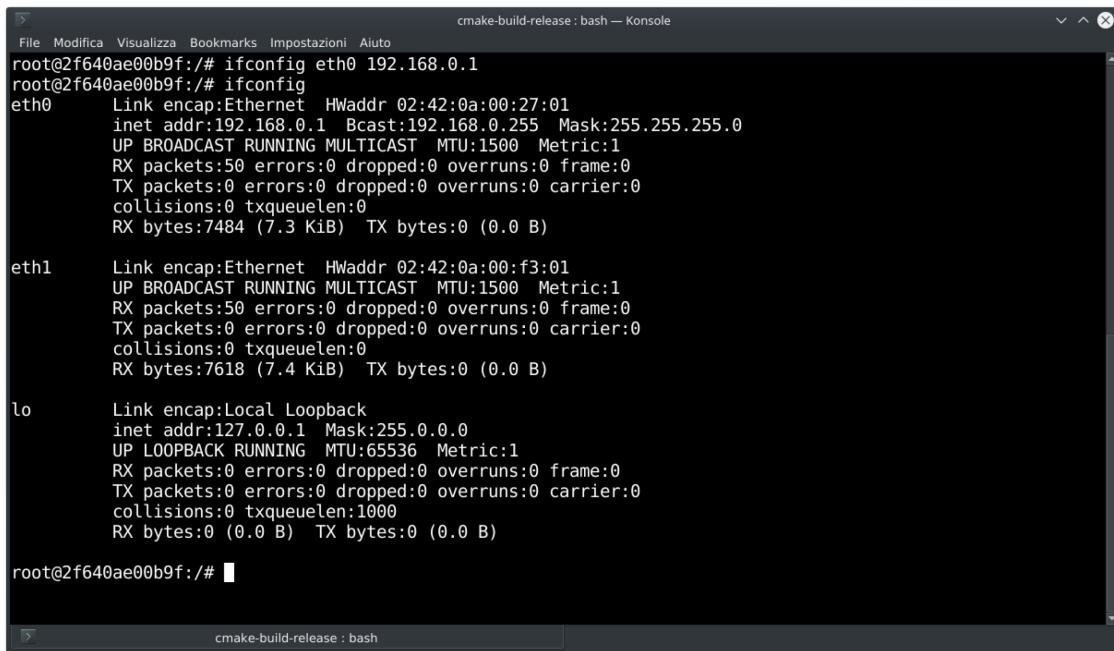
Il tasto “delete” lo eliminerà.

Nella sezione di routing verranno riportate le informazioni sul routing estrapolate tramite *ip route*, se presenti.

Tramite il bottone refresh è possibile aggiornare le informazioni relative alla macchina, questa operazione viene fatta in automatico ogni 30 secondi.

7.6 IL TERMINALE

Cliccando su “Open in terminal” verrà aperta una finestra di Konsole collegata al container. Tramite tale terminale sarà possibile effettuare tutte le operazioni necessarie a configurare correttamente il nodo.



```
cmake-build-release : bash — Konsole
File Modifica Visualizza Bookmarks Impostazioni Aiuto
root@2f640ae00b9f:/# ifconfig eth0 192.168.0.1
root@2f640ae00b9f:/# ifconfig
eth0      Link encap:Ethernet HWaddr 02:42:0a:00:27:01
          inet addr:192.168.0.1 Bcast:192.168.0.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:50 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7484 (7.3 KiB) TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet HWaddr 02:42:0a:00:f3:01
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:50 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7618 (7.4 KiB) TX bytes:0 (0.0 B)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

root@2f640ae00b9f:/# █
```

Per uscire dal container usare la sequenza **ctrl + p + ctrl + q**. Usare il comando exit o il tasto X provocherà lo spegnimento del container, dunque la perdita delle configurazioni effettuate.

8 CONCLUSIONI

8.1 EFFICACIA

Grazie alle scelte effettuate in fase di progettazione il tool è in grado di effettuare tutti i task richiesti, è in grado di essere eseguito anche su macchine dotate di hardware limitato, può essere usato in fase di esame, è facilmente installabile, leggero ed è integrabile in un ambiente completo come una macchina virtuale o una distribuzione live.

8.2 SVILUPPI FUTURI

Prendendo in considerazione le scoperte effettuate durante lo sviluppo ed il codebase sviluppato è evidente che questo tool si può evolvere in uno strumento professionale che non si limiti solo al mondo didattico. Si sta infatti già pianificando e realizzando un software più completo che permetta di gestire in maniera rapida e quanto più automatizzata possibile delle topologie di rete complesse formate da Docker containers.

Inoltre il tool, come il software che ne deriverà, potrà essere reso compatibile anche con i sistemi Windows e MacOs. Anche se Docker rende al meglio nei sistemi dotati di kernel Linux.

8.3 RINGRAZIAMENTI

Innanzitutto vorrei ringraziare il mio relatore, Prof. Salvatore Antonio Riccobene per avermi permesso di realizzare questo splendido progetto. Vorrei inoltre ringraziarlo per avermi fatto appassionare al mondo del networking, influenzando quindi in maniera positiva le mie scelte accademica, e per il suo inimitabile carisma che ha reso l'ambiente universitario più umano e vicino allo studente.

Ringrazio il mio correlatore, Dott. Federico Fausto Santoro, che si è rivelato preziosissimo durante lo sviluppo del progetto. Le sue idee e la conoscenza approfondita di Docker si sono rivelate determinanti al fine della buona riuscita del progetto, mi è sempre stato accanto e la sua disponibilità è stata, oserei dire, infinita.

Ringrazio il Dott. Dario Messina che mi ha illustrato il software da lui sviluppato, VB-ANT, e il funzionamento delle API di VirtualBox.

Ringrazio il Prof. Giampaolo Bella che grazie alla sua professionalità ed abilità come docente mi ha permesso di avvicinarmi al mondo della sicurezza informatica. Argomento di mio grande interesse.

Ringrazio inoltre il resto del corpo docenti che mi ha permesso di crescere ed imparare molto più di quanto io potessi sperare.

Vorrei ringraziare i colleghi:

- Nino Pulvirenti per l'impareggiabile aiuto fornito durante lo studio di alcune materie più ostiche e per essermi stato accanto come prezioso amico;
- Dott. Pietro Maugeri per essere stato un eccellente compagno di sviluppo durante la realizzazione del progetto di Programmazione I e per i saggi consigli che ha elargito al sottoscritto durante gli studi;
- Dott. Orazio Contarino per il prezioso apporto dato durante lo sviluppo dei progetti di Basi di Dati e Computer Forensics;
- Marco Aloisi, Aureliano Consoli e Carlo Zerma per il lavoro svolto insieme durante e dopo il corso di Internet Security;
- Mirko Cantone i cui consigli, informazioni e disponibilità si sono sempre rivelati indispensabili durante questi anni;
- Paolo Ganesio, Emanuele Viglianisi, Dario Tizzoni, Alberto Ottimo, Simone Marano, Davide de Pasquale, Anna Lo Trovato, Alfio Faro, Vincenzo Aliperti, Salvatore Pizzimento, Ludovico Trupia, Giulia Barchitta, Stefano Borzì, Dario Flaccavento, Filippo Randazzo e Amira Bouali che oltre ad essere colleghi sono stati amici che hanno reso questa esperienza accademica indimenticabile;
- Tutti i colleghi che posso aver dimenticato di scrivere e che mi perdoneranno per la dimenticanza.

Si ringrazia inoltre gli amici Mariagrazia Maugeri, Victoria Abbattista, Martina Giumarra e Dario Porto; altre splendide persone, conosciute grazie ai colleghi, con i quali si è formato un saldo legame di amicizia.

Un ringraziamento speciale va alla mia grande amica, la Dott.ssa. Angela Ficarelli, che mi è stata vicina durante questi difficili mesi e il cui aiuto, supporto morale e i saggi consigli durante lo studio e nella correzione delle varie relazioni e di questa tesi sono stati indispensabili al conseguimento di questa mia Laurea di Primo Livello.

Ringrazio tutti i miei parenti, che mi sopportano e sostengono da una vita. In particolare mia nonna Gaetana Scandura, che non ha mai smesso di credere in me.

Il più grande GRAZIE va a mia madre, Giuseppa Cutuli. Senza i suoi sacrifici e la sua dedizione nel crescermi non sarei mai riuscito a fare tutta questa strada e non sarei qui a scrivere questa tesi e a raggiungere questo importante risultato.

Mamma, ti devo tantissimo, grazie, spero che tu possa essere fiera di me e di non deluderti mai.

GRAZIE a mio padre, Alfredo Steccanella. È stato lui a spingere e far nascere il mio amore nei confronti dell'informatica. È stato lui che ha fatto sì, ad ogni costo, che io potessi studiare e prendere finalmente questa laurea, coronamento di un sogno avuto sin da bambino.

Papà, nessun ringraziamento sarà sufficiente, i tuoi primi insegnamenti mi sono sempre stati cari e utili, il tuo sostegno: essenziale.

*Infine grazie a voi, che avete avuto la voglia, l'interesse e la pazienza di
leggere questa tesi.*

Luca Steccanella

8.4 SITOGRAFIA

<https://www.docker.com/>

<https://docs.docker.com/>

<https://goto.docker.com/rs/929-FJL-178/images/Docker-for-Virtualization-Admin-eBook.pdf>

<https://azure.microsoft.com/it-it/services/container-service/>

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html>

<http://www.cplusplus.com/>

<http://en.cppreference.com/w/>

<https://www.qt.io/>

<https://doc.qt.io/>

<http://man7.org/linux/man-pages/>