Virtualizzazione

Cos’è la virtualizzazione?

Perché utilizzare la tecnica della virtualizzazione?

Esistono tipi diversi di virtualizzazione?

La virtualizzazione è una tecnica per condividere le risorse di un computer tra più ambienti di esecuzione.

La virtualizzazione classica, quella che storicamente si è sviluppata per prima, permette di eseguire contemporaneamente più istanze di sistemi operativi “**guest**” in un’unica macchina fisica “**host**”. I sistemi operativi “guest” colloquiano con le risorse messe a disposizione dalla macchina fisica “host” attraverso un componente software di livello intermedio generalmente denominata “**hypervisor**” o “virtual machine monitor” (VMM).

La possibilità di avere più ambienti virtuali all’interno di una sola macchina fisica presente diversi vantaggi. Per esempio:

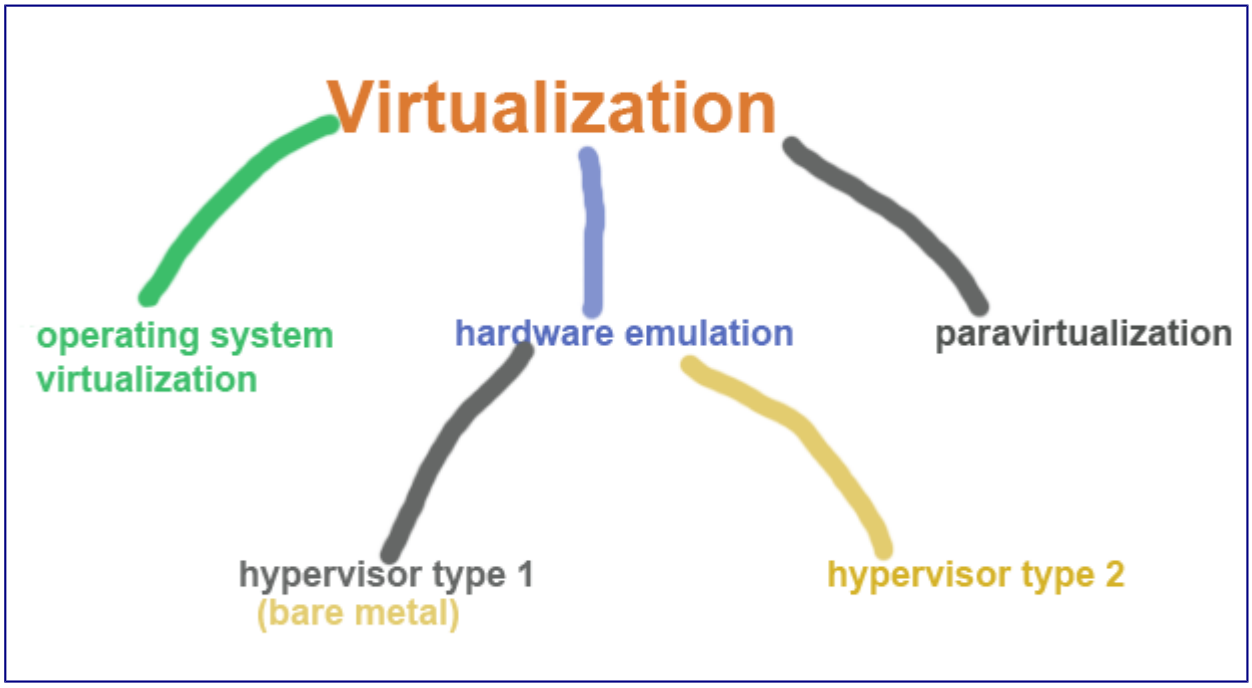
- Uno sviluppatore può verificare il funzionamento dei suoi prodotti su piattaforme diverse senza dover disporre di più macchine fisiche.
- Un amministratore può verificare il funzionamento dei servizi che deve gestire su diversi sistemi operativi associati a diverse macchine virtuali ed anche in un ambiente di rete virtuale.
- Facilita la predisporre di ambienti di testing.
- Aumenta l'affidabilità e la sicurezza del sistema dato che è possibile associare ad ogni sistema guest uno (o comunque pochi) servizi. L’hypervisor isola le macchine guest garantendo che un malfunzionamento di una non influenzi la stabilità delle altre.
- Società che offrono servizi possono ridurre notevolmente gli investimenti in hardware offrendo questi servizi in modo virtualizzato. Del resto i notevoli progressi tecnologici (si pensi all’evoluzione dei processori, delle RAM, dei supporti di memorizzazione di massa) permette ad un solo host di virtualizzare molti sistemi guest.
- Se si riducono gli investimenti nell’acquisto di server fisici di conseguenza si avranno dei risparmi relativamente alle spese di tipo TCO (Total Cost of Ownership), come per esempio consumo di energia, canoni di manutenzione per assistenza, utilizzo di ambienti che rispettano le normative in termini di sicurezza .
- Ancora sul piano dell’affidabilità è più facile garantire il ripristino di un server a causa di un guasto. Semplici procedure permettono il salvataggio di una macchina

guest ed il suo ripristino (eventualmente su un altro host con caratteristiche hardware compatibili con quello originale).

In definitiva i vantaggi ruotano tutti intorno ai seguenti punti:

1. **Riduce i costi**
2. **Semplifica la gestione di una infrastruttura IT**
3. **Consente una migliore affidabilità**
4. **Consente una maggiore flessibilità**

La virtualizzazione può essere di tre tipi fondamentali: la virtualizzazione a livello di Sistema Operativo (soluzione di virtualizzazione più recente, per esempio i container come "docker"), quella che si basa sull'emulazione dell'hardware e la paravirtualizzazione.



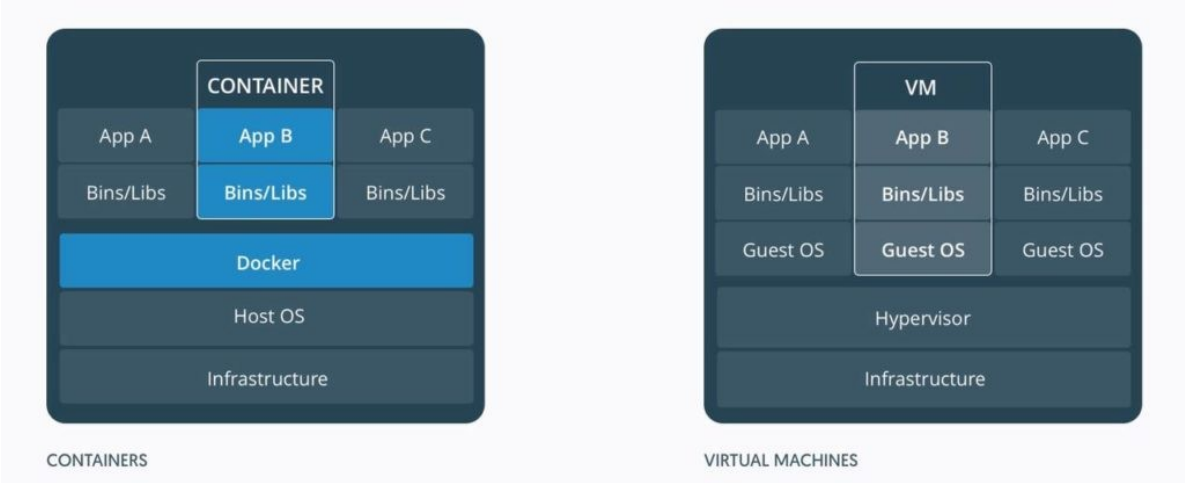
Virtualizzazione a livello di sistema operativo

Invece di creare una istanza virtuale di tutto un server fisico (processore, storage, connessioni di rete, sistema operativo…) come accade per le macchine virtuali, si attiva una istanza virtuale **solo dello spazio utente**, quindi essenzialmente dell'ambiente di esecuzione delle applicazioni. Queste istanze si chiamano **container**. I container condividono lo stesso sistema operativo e questo li rende molto più "leggeri" delle macchine virtuali (richiedono poche risorse di CPU) e attivabili in pochi istanti.

I container offrono la possibilità di sviluppare soluzioni (per esempio un'applicazione web) senza preoccuparsi della versione del sistema operativo, del linguaggio, del runtime installato sul sistema, della versione del database delle versioni delle librerie. Tutti i file e tutte le librerie di cui l'applicazione ha bisogno sono incluse in un container. Lo sviluppatore deve semplicemente creare una immagine basata sulla sua distribuzione preferita, includere tutte le dipendenze e distribuirla.

Un container permette sostanzialmente di creare “un pacchetto” con tutto ciò che serve ad un’applicazione (applicazione, librerie, database, …..)

L’immagine seguente riassume le differenze tra container e virtualizzazione classica (Docker è una soluzione di virtualizzazione che in questi periodo sta avendo molto successo)



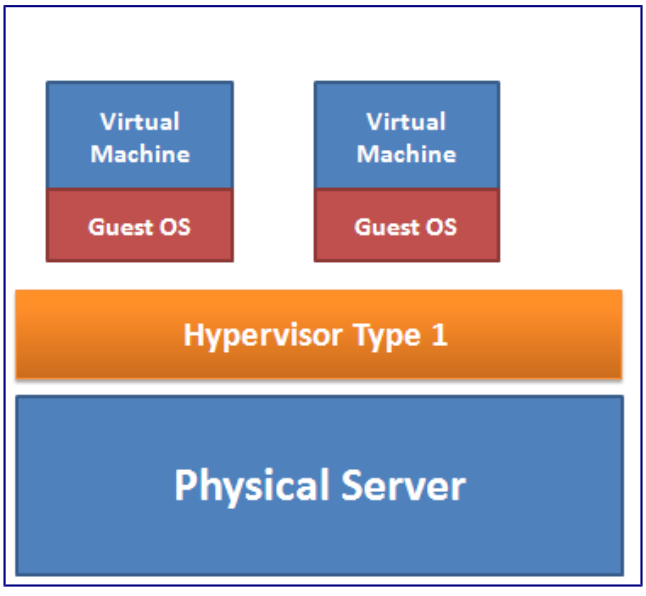
Emulazione Hardware

Nell’emulazione hardware è presente un software specifico: **Hypervisor** o VMM (Virtual Machine Monitor). L’hypervisor gestisce l’emulazione hardware per i diversi S.O. presenti in un singolo server fisico.

In emulazione, il sistema di virtualizzazione simula tutto l'hardware su cui viene installato il sistema operativo ospite, detto **guest.**

Tipi di Hypervisor

Esistono due tipi di Hypervisor: Type 1 e Type 2. Type 1 Hypervisor sono anche chiamati implementazioni “bare-metal” (metallo nudo) perché si pongono immediatamente sopra l’hardware, senza nessun sistema operativo. Comunicando direttamente con l’hardware sono più veloci dei type 2



Gli Hypervisor Type 2 si appoggiano sul sistema operativo della macchina Host.

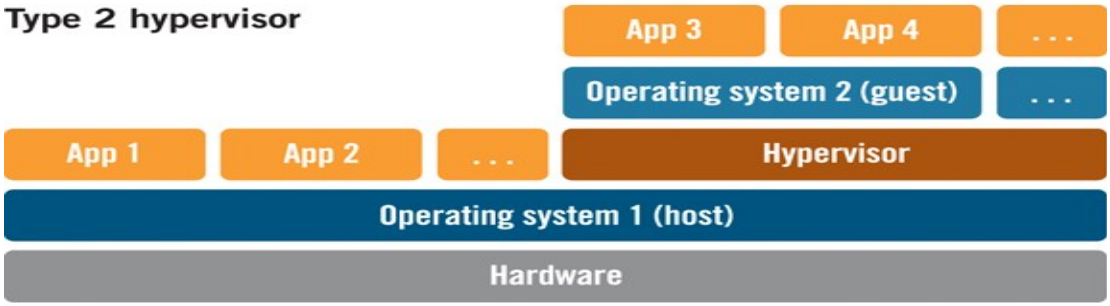


Figure 1. A Type 2 hypervisor runs as an application on a host operating system.

VirtualBox, per esempio, è un esempio di hypervisor di tipo 2, infatti viene installato su un sistema operativo già presente.

KVM (Kernel-Based Virtual Machine) di Linux è considerato da alcuni un hypervisor di tipo 1 , da altri di tipo 2. Effettivamente KVM è un modulo integrato nel kernel di Linux per cui in quanto tale può essere considerato come uno strato posto immediatamente sopra l'hardware.

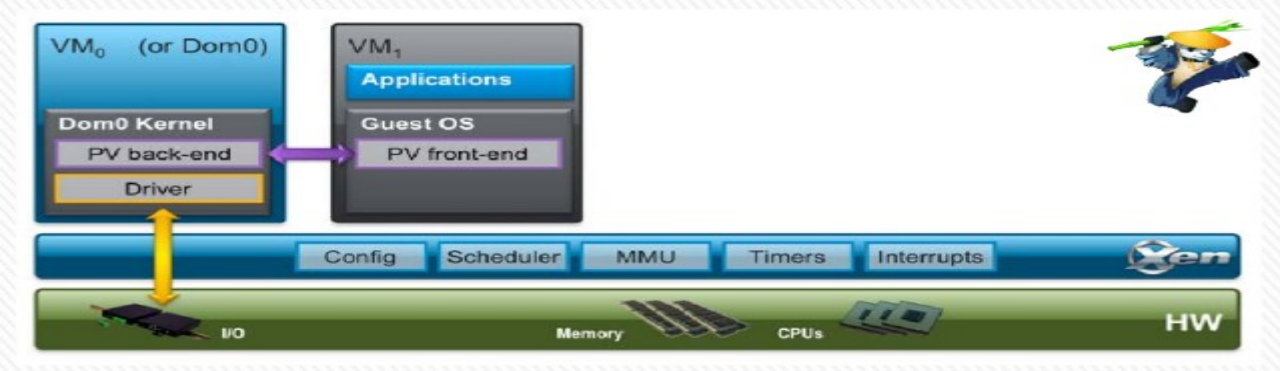
Paravirtualizzazione

La paravirtualizzazione non mira a creare un'emulazione dell'hardware di un generico computer, ma piuttosto a regolare e controllare l'accesso alle risorse fisiche della macchina da parte delle varie istanze delle macchine virtuali.

In un ambiente paravirtualizzato un sistema operativo ospite (chiamato DomainU) è direttamente installato sull'hypervisor che non contiene nessun device driver per i dispositivi hardware. Invece un sistema operativo ospite speciale chiamato Domain 0 ha accesso diretto all'hardware. Tutti i sistemi ospite accedono all'hardware tramite il Domain 0.

DomainO è un sistema operativo standard (per es. Linux) che è stato modificato per comunicare con l'hypervisor per gestire l'accesso all'hardware. I sistemi operativi ospiti non dialogano direttamente con l'hardware, ma con il domini speciale Domain0.

Il maggior problema alla paravirtualizzazione è la necessità di modificare il sistema operativo ospite rimpiazzando le chiamate all'hardware a chiamate alle API messe a disposizione dall'Hypervisor. Il kernel deve essere modificato prima dell'installazione. Siccome non è possibile modificare il kernel dei sistemi operativi proprietari gli utenti della paravirtualizzazione devono utilizzare sistemi operativi open source.



La seguente tabella riassume i principali tipi di virtualizzazione, le loro caratteristiche ed alcune soluzioni implementative

Virtualization type	Remarks	Products
Operating system virtualization	It creates an abstracted view of the OS including root file system, process table etc. They are suitable for homogeneous OS environment where consistent OS version is necessary.	OpenVZ, Virtuozzo, Sun Solaris, docker
Hardware emulation	It Gives abstracted view of underlying hardware, offering support for heterogeneous OS.This is a market leading virtualization technology.	VMware VMserver, VMware ESX, Microsoft virtual server, Virtualbox, KVM
paravirtualization	It multiplexes the access to hardware resources, offers high performance, requires guest OS modification before deployment.More complex than hardware emulation	Xen(open source)

Virtualizzazione KVM

KVM è una soluzione di virtualizzazione Open Source per sistemi x86 semplice e performante perché integrata nel kernel Linux. Per attivarla bisogna avere un kernel >= 2.6.20 e un processore con le estensioni **Intel VT** oppure **AMD-V**.

Per vedere se il processore ha tali estensioni controllare che in `/proc/cpuinfo` sia presente il flag **vmx** (per i processori Intel) **svm** (per i processori AMD).

Per utilizzare KVM in Debian deve essere installato il pacchetto kvm (qemu-kvm in stretch) e devono essere caricati nel kernel i relativi moduli:

 # lsmod | grep kvm

```
root@dalu:~# lsmod | grep kvm
kvm_intel          192512  0
kvm                589824  1 kvm_intel
irqbypass           16384  1 kvm
```

Se il test precedente ha avuto successo è necessario verificare che le estensioni di virtualizzazione siano abilitate nel BIOS.

KVM può essere considerato un Hypervisor di tipo 1. In realtà QEMU (Hypervisor di tipo 2) utilizza il modulo del kernel Linux KVM e da questa unione nasce un Hypervisor di tipo 1.

Libvirt

Libvirt è uno strumento di gestione per piattaforme di virtualizzazione e tra le molte piattaforme supportate c'è anche KVM. I programmi che utilizzano libvirt sono molti, tra questi citiamo **VMM** (Virtual Machine Manager) che mette a disposizione un tool grafico di gestione e l'interfaccia a linea di comando **virsh**.

Il seguente comando installa, oltre al sistema kvm, anche libvirt:

apt-get install qemu-kvm libvirt-clients libvirt-daemon-system virtinst virt-manager virt-viewer

Per gestire le macchine virtuali da utente non privilegiato:

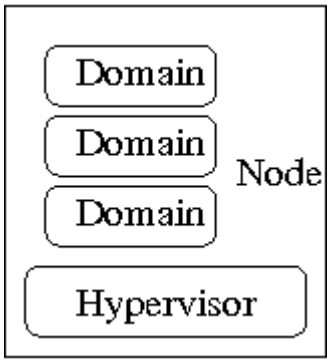
 # adduser <youruser> libvirt
 # adduser <youruser> libvirt-qemu

libvirt: terminologia (https://libvirt.org/goals.html)

E' bene avere chiari alcuni termini e concetti utilizzati da libvirt e/o relativi in generale alla virtualizzazione:

- un **nodo** è una singola macchina fisica

- un **Hypervisor** è uno strato di software che permette la virtualizzazione all'interno di un nodo di un insieme di macchine virtuali con configurazioni anche diverse rispetto al nodo stesso
- un **dominio** è un'istanza di un sistema operativo (o di un "sottosistema" nel caso della virtualizzazione a livello di sistema operativo (container)) che è eseguito su una macchina virtuale messa a disposizione dal container.



Obiettivo di libvirt è quello di mettere a disposizione uno strumento per gestire i domini presenti su un nodo. Quindi libvirt metterà a disposizione strumenti per: creare, modificare, monitorare, controllare migrare i domini.

Libvirt usa un file xml per descrivere la configurazione di un dominio; questo permette agli utenti di modificare il funzionamento del dominio utilizzando un editor o altri strumenti. Per esempio il seguente è un frammento di file XML relativo ad un dominio KVM:

```
<domain type='kvm'>
  <name>testkvm</name>
  <uuid>5169786e-168b-40c1-81d9-7d255fb03174</uuid>
  <memory unit='KiB'>1048576</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-i440fx-2.8'>hvm</type>
    <boot dev='hd'/>
  </os>
          ……...
 <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/var/lib/libvirt/images/testkvm.qcow'/>
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </disk>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hda' bus='ide'/>
      <readonly/>
      <address type='drive' controller='0' bus='0' target='0' unit='0'/>
    </disk>
          ……
<interface type='network'>
      <mac address='52:54:00:0d:11:bf'/>
      <source network='default'/>
      <model type='virtio'/>
```

```
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

libvirt: task (https://wiki.libvirt.org/page/VM_lifecycle)

Creare un dominio

Innanzitutto è necessario creare il dispositivo virtuale di rete a cui si connettono le macchine virtuali e che è necessario per accedere alla rete esterna. La rete è definita utilizzando un file XML:

virsh net-create /etc/libvirt/qemu/networks/default.xml

Per attivare la rete eseguire i comandi:

virsh net-start default	attiva la rete di default
virsh net-autostart default	la rete di default viene attivata al boot

Per visualizzare l'elenco delle reti:

virsh net-list --all

```
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
```

Ora dovrebbe essere possibile creare un domino. Questo può essere fatto in diversi modi. Al seguente link, per esempio, è descritta la procedura utilizzando la GUI di Virtual Machine Manager: https://wiki.libvirt.org/page/CreatingNewVM_in_VirtualMachineManager .

Un altro modo è quello di utilizzare il comando virt-install.

```
#virt-install --connect qemu:///system --virt-type kvm --name testkvm --ram 1024  \
        --disk /var/lib/libvirt/images/testkvm.qcow,format=qcow2,size=10 \
        --cdrom /home/roberto/Scaricati/debian-9.1.0-i386-DVD-1.iso \
        --network bridge=virbr0 --vnc --os-type linux --os-variant debian9
```

N.B. fare attenzione al nome del bridge (virbr0) che deve coincidere con quello specificato nel file xml di definizione della rete.

Un altro modo ancora è quello di creare il file XML di definizione del dominio e del volume e poi eseguire virsh con gli appropriati comandi: vol-create and define.

I "Volumi" sono associati ad un pool, di default quello chiamato "default". "Default" è un pool "directory type" , cioè tutti i volumi sono memorizzati come file in una directory.

Per vedere i volumi presenti in un pool:

root@dalu:~# virsh vol-list default

Esempio di definizone XML di un volume (new_volume.xml):

```
<volume>
 <name>sparse.img</name>
 <capacity unit="G">10</capacity>
</volume>
```

Questo definisce un volume della capacità di 10 GB. Per creare il volume nel pool "*default*":

```
# virsh vol-create default new_volume.xml
```

Esempio di definizione XML di un dominio (MyNewVM.xml):

```
<domain type='kvm'>
  <name>MyNewVM</name>
  <currentMemory>524288</currentMemory>
  <memory>524288</memory>
  <uuid>30d18a08-d6d8-d5d4-f675-8c42c11d6c62</uuid>
  <os>
    <type arch='x86_64'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/><apic/><pae/>
  </features>
  <clock offset="utc"/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <vcpu>1</vcpu>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/MyNewVM.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='block' device='cdrom'>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <mac address='52:54:00:9c:94:3b'/>
      <model type='virtio'/>
    </interface>
    <input type='tablet' bus='usb'/>
    <graphics type='vnc' port='-1'/>
    <console type='pty'/>
    <sound model='ac97'/>
    <video>
      <model type='cirrus'/>
    </video>
  </devices>
</domain>
```

Per definire il dominio:

```
# virsh define MyNewVM.xml
```

N.B. libvirt distingue tra domini transienti e persistenti. Se il dominio viene creato (ma non definito) non sarà possibile riattivarlo successivamente alla sua terminazione. Tuttavia da un dominio transiente è possibile ottenere il suo dump XML:

virsh dumpxml `GuestID` > `guest.xml`

e poi creare in modo persistente il sistema guest:

virsh create guest.xml

(***DA CAPIRE COME DOMINIO E VOLUME SONO LEGATI)***

Visualizzare i domini presenti su un host

#virsh list –all

Modificare un dominio

Any domain can be edited in a user's favourite editor. What is needed is to set the $VISUAL or $EDITOR environment variable and run:

 # virsh edit <domain> (la variabile $VISUAL permette di specificare l'editor preferito)

E' anche possibile utilizzare la GUI Virtual Machine Manager.

Attivare un dominio

 # virsh start MyNewVM

Stop o reboot di un dominio

 # virsh shutdown <domain>

 # virsh reboot <domain>

Esiste anche quello che si chiama hard-stop:

 # virsh destroy <domain>

(è equivalente a togliere il cavo dell'alimentazione)

Mettere in pausa un dominio

 # virsh suspend <domain>

In questo stato il sistema guest consuma RAM, ma non CPU

Risvegliare un dominio

 # virsh resume <domain>

Snapshot di un dominio

 # virsh snapshot-create <domain>

Listing degli snapshots di un dominio

`# virsh snapshot-list <domain>`

```
 Name                 Creation Time             State
----------------------------------------------------
 1295973577           2011-01-25 17:39:37 +0100 running
 1295978837           2011-01-25 19:07:17 +0100 running
```

Ripristinare un dominio da uno snapshot

 # virsh snapshot-restore <domain> <snapshotname>

Rimuovere uno snapshot da un dominio

```
# virsh snapshot-delete <domain> <snapshotname>
```

Clonare una macchina virtuale:

#virt-clone - -original nome_vm_originale - -auto-clone

Migrazione

Libvirt mette a disposizione il supporto per la migrazione di un dominio. Significa che è possibile migrare un dominio da un host ad un altro attraverso la rete. La migrazione può avvenire in 2 modi:

- Plain migration: The source host VM opens a direct unencrypted TCP connection to the destination host for sending the migration data. Unless a port is manually specified, libvirt will choose a migration port in the range 49152-49215, which will need to be open in the firewall on the remote host.
- Tunneled migration: The source host libvirtd opens a direct connection to the destination host libvirtd for sending migration data. This allows the option of encrypting the data stream. This mode doesn't require any extra firewall configuration, but is only supported with qemu 0.12.0 or later, and libvirt 0.7.2 or later.

For a successful migration there are couple of things needed to be done. For instance, storage settings have to match. All volumes that migrated domain use have to be stored under the same paths. For full checklist follow this page. We also recommend you use secure migration.

Once pre-migration checks are done, you can migrate machine using virsh:

```
# virsh migrate <domain> <remote host URI> --migrateuri tcp://<remote
host>:<port>
```

Cancellazione di un dominio

 # virsh undefine <domain>

Rinominare un dominio (https://www.cyberciti.biz/faq/how-to-rename-kvm-virtual-machine-vm-domain-with-virsh-command/)

- Primo metodo

  # virsh domrename {domain} {new-name}

- Secondo metodo. Si crea una nuova definizione XML del dominio e la si reimporta

  # virsh dumpxml vecchio_nome > domain.xml

  # vi domain.xml

<name>nuovo_nome</name>

 # virsh shutdown vecchio_nome

 # virsh undefine vecchio_nome
 # virsh define domain.xml

virsh start nuovo_nome

Liberare la memoria utilizzata da un dominio

Un volume utilizzato da un dominio. Un volume può contenere dati confidenziali per cui è necessario effettuare una cancellazione completa dei dati prima di rimuoverlo

 # virsh vol-wipe <volume>

(rende inaccessibili i dati sovrascrivendoli)

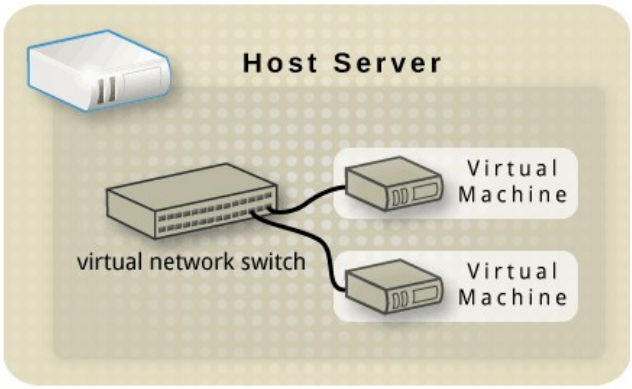
 # virsh vol-delete <volume>

Configurazione del networking

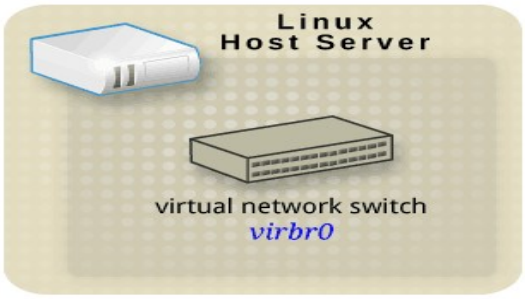
(https://wiki.libvirt.org/page/VirtualNetworking)

Libvirt introduce il concetto di “Virtual Network Switch”:

Si tratta di un costrutto software presente sull’host a cui i sistemi ospiti si connettono.



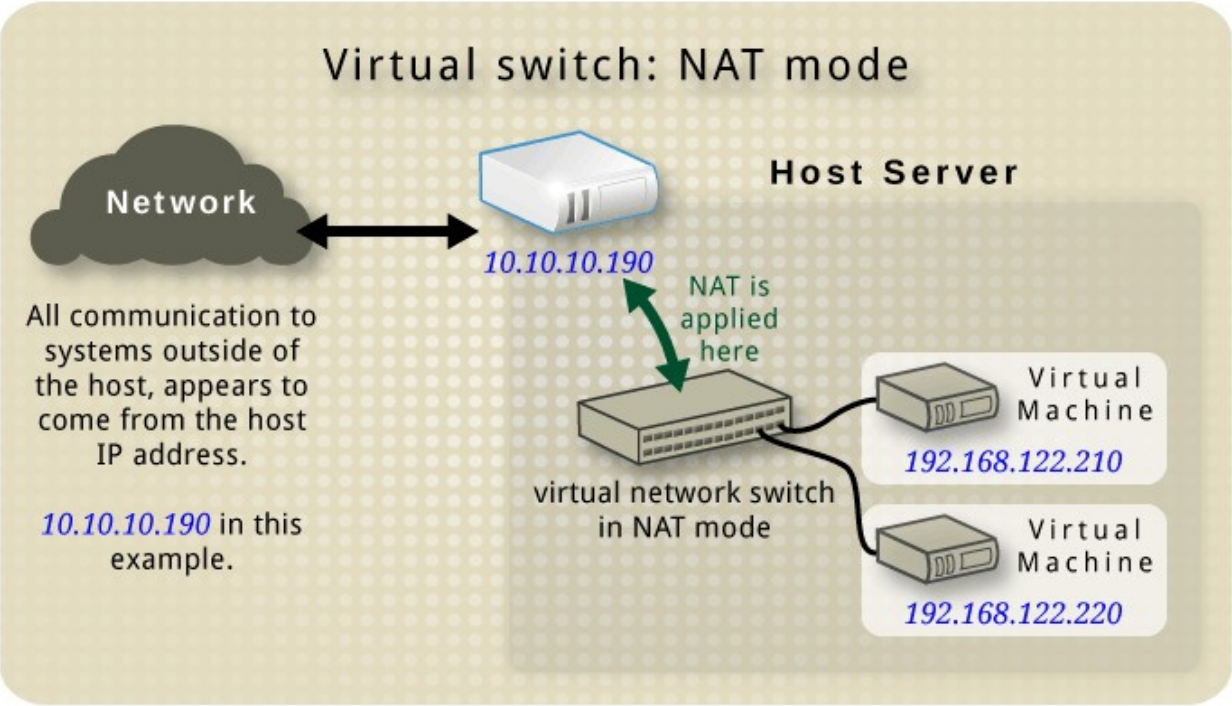
Sulla macchina host Linux lo switch virtuale viene visto come un’interfaccia di rete:



Questo è il risultato del comando ifconfig su un host in cui è presente lo switch virtuale vibr0:

```
root@dalu:~# ifconfig virbr0
virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:11:09:8d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Lo switch virtuale esegue un’operazione di NAT, quindi ogni host virtuale comunica con l’esterno utilizzando l’indirizzo IP:



Ovviamente il natting è gestito via iptables, quindi bisogna prestare attenzione a non modificare le regole definite da libvirt per non incorrere in malfunzionamenti dei domini guests.
Ogni switch virtuale fornisce via dhcp un intervallo di indirizzi IP ai domini guests (da 192.168.122.2 fino a 192.168.122.254). Libvirt ussa per questo scopo il programma **dnsmasq** che libvirt configura ed attiva automaticamente per ogni switch virtuale.

Se si ha la necessità di associare indirizzi IP statici ai domini guest (per esempio se devono fornire servizi all'esterno) si può operare nel seguente modo:

- si ricavano i mac address dei domini con il comando:

  root@dalu:~# virsh dumpxml <nome_del_dominio>

- si modifica la configurazione della rete virtuale associando i mac address agli indirizzi IP

```
root@dalu:~# virsh net-edit default
<network>
  <name>default</name>
  <uuid>f8e7da51-c6c6-47b5-a0d9-3a1f58a0f1b9</uuid>
  <forward mode='nat'/>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:11:09:8d'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
      <host mac='52:54:00:0d:11:bf' name='test1' ip='192.168.122.11'/>
      <host mac='52:54:00:73:88:dd' name='test2' ip='192.168.122.12'/>
    </dhcp>
  </ip>
</network>
```

- si esegue lo stop e lo start della rete

  virsh net-destroy default
  virsh net-start default

Per le altre modalità di gestione della rete vedere la guida del networking di libvirt:
https://wiki.libvirt.org/page/VirtualNetworking
La configurazione della rete può essere effettuata graficamente attraverso la GUI virt-manager, oppure con i comandi virsh appositamente definiti:

virsh net-list [--all] [--inactive]	permette di vedere tutte le reti
virsh net-start default	attiva la rete di default
virsh net-autostart default	la rete di default viene attivata al boot

Collegarsi ai sistemi guest

Per collegarsi da console del sistema host:

1. Utilizzando un tunnel SSH:
   - individuare l'indirizzo IP della macchina guest con i comandi "arp -n" (o "ip neighbour") o "nmap -sn"
   - connettersi via ssh utilizzando l'account non privilegiato utilizzato in fase di installazione del sistema guest
2. Utilizzando un terminale di console:
   - *# systemctl start serial-getty@ttyS0*
   - *# systemctl enable serial-getty@ttyS0*

   Successivamente sul sistema guest:

   - si abilita la console editando il file /etc/default/grub in modo che sia presente la riga:

     GRUB_CMDLINE_LINUX_DEFAULT="console=ttyS0"

   - si aggiorna GRUB*: sudo update-grub2*

   ora è possibile connettersi da terminale al dominio:

   # virsh console testkvm

   per chiudere la connessione:
   # Ctrl+]

3. Utilizzando l'interfaccia grafica

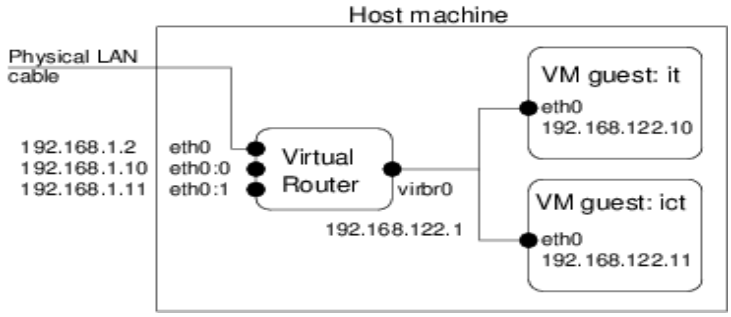
   # virt-viewer testkvm

Oppure quando la macchina virtuale è attivata si può utilizzare la GUI grafica (VMM) per aprire la sua console (grafica)

Eseguire più server Linux come KVM Virtual Machines

(https://sandilands.info/sgordon/linux-servers-as-kvm-virtual-machines)

La topologia della rete per l'host e le macchine guest è la seguente:



La macchina host ha un indirizzo IP statico (o dinamico via DHCP) per l'interfaccia eth0 (o enp…..) e uno switch virtuale (`virbr0`) è presente con IP 192.168.122.1. Avendo modificato la rete virtuale di default (vedi sopra) le interfacce virtuali sulle macchine guest hanno un ip fisso. Il meccanismo degli alias (interfacce virtuali) permette di accedere dall'esterno ai servizi sulle macchine virtuali.

Bisogna impostare le interfacce virtuali in modo permanente

```
auto lo
iface lo inet loopback

auto enp2s0
iface enp2s0 inet static
        address 192.168.1.37
        network 192.168.1.0
        netmask 255.255.255.0
        broadcast 192.168.1.255
        gateway 192.168.1.1
        nameserver 192.168.1.1

#per raggiungere guest kvm in DNAT
auto enp2s0:0
iface enp2s0:0 inet static
        address 192.168.1.10
        network 192.168.1.0
        netmask 255.255.255.0
        broadcast 192.168.1.255
        gateway 192.168.1.1

auto enp2s0:1
iface enp2s0:1 inet static
        address 192.168.1.11
        network 192.168.1.0
        netmask 255.255.255.0
        broadcast 192.168.1.255
        gateway 192.168.1.1
```

o temporaneo (per fare un test)

```
root@dalu:~# ip addr add 192.168.1.11/24 dev enp2s0
root@dalu:~# ip addr add 192.168.1.12/24 dev enp2s0
```

enp2s0:0 è l'interfaccia virtuale utilizzata per rendere accessibile l'interfaccia della macchina virtuale 192.168.122.11 (in modo similare per eth0:1).

Configurazione del NAT (dnat) sull'host

Il DNAT deve essere configurato per permettere al traffico in arrivo su una determinata interfaccia dell'host (quelle virtuali) di proseguire verso i domini guests. Inoltre dovranno essere predisposte le regole relative ai servizi che i sistemi guests offriranno. Esempio:

```
$ sudo iptables -t nat -I PREROUTING -d 192.168.1.10 -j DNAT --to-destination 192.168.122.10

$ sudo iptables -t nat -I POSTROUTING -s 192.168.122.10 -j SNAT --to-source 192.168.1.10   (nella documentazione – vedi link poco sopra – la mette, a me pare non serva)

$ sudo iptables -t nat -I PREROUTING -d 192.168.1.11 -j DNAT --to-destination 192.168.122.11

$ sudo iptables -t nat -I POSTROUTING -s 192.168.122.11 -j SNAT --to-source 192.168.1.11  (nella documentazione – vedi link poco sopra – la mette, a me pare non serva)
```

Vanno opportunamente configurate le politiche di forward per permettere che il traffico da/per la macchina virtuale attraversi la chain di forward sull'host. Agendo in modo drastico (ma da fare solo per un primo test!!) si potrebbe dare il comando:

iptables -I FORWARD -j ACCEPT (passa tutto!!)

A questo link è spiegato il comportamento di libvirt nei confronti del firewall: https://libvirt.org/firewall.html

Starting VM Guests al boot

$ virsh -c qemu:///system
Welcome to virsh, the virtualization interactive terminal.

Type: 'help' for help with commands
'quit' to quit

virsh # autostart <nome_virtual_machine>

Riferimenti:

https://wiki.libvirt.org/page

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/virtualization/