Creación de maquinas virtuales utilizando kvm-qemu

{{>TOC}}

Gestión de máquinas virtuales

Por consola

- [[gestion-maq-virt| Gestión de máquinas virtuales por consola]]

Gráficamente

Para gestionar las Máquinas virtuales gráficamente desde un Ubuntu con X, podemos instalar el meta-paquete ‘ubuntu-virt-mgmt’, que instala el paquete ‘virt-manager’ y ‘virt-viewer’ en nuestro desktop para gestionar y crear las máquinas y acceder (por VNC) a los guests.

1. Instalar

   sudo apt-get install ubuntu-virt-mgmt

2. Configurar conexión
   - Una vez instalado hay que ir a Archivo -> Añadir conexión: KVM/Qemu / ssh / ... etc etc

¿Por qué utilizar kvm-quemu?

Básicamente podemos considerar 3 tipos de virtualización: emulación, virtualización completa (Full Virtualization) y paravirtualización:

Emulación:

La emulación se basa en crear máquinas virtuales que emulan el hardware de una o varias plataformas hardware distintas. Este tipo de virtualización es la más costosa y la menos eficiente, ya que obliga a simular completamente el comportamiento de la plataforma hardware a emular e implica también que cada instrucción que se ejecute en estas plataformas sea traducida al hardware real.
Sin embargo la emulación tiene características interesantes, como poder ejecutar un sistema operativo diseñado para una plataforma concreta sobre otra plataforma, sin tener que modificarlo, o en el desarrollo de firmware para dispositivos hardware, donde se pueden comenzar estos desarrollos sin tener que esperar a tener disponible el hardware real.

Uno de los ejemplos más destacados de la actualidad es QEMU. QEMU, entre otras cosas, permite emular diferentes plataformas Hardware como x86, x86-64, PowerPC, SPARC o MIPS. Así pues, podríamos tener dentro de un servidor linux varios equipos x86 o PowerPC, corriendo diferentes versiones de Linux.

Virtualización completa

Con este término se denominan aquellas soluciones que permiten ejecutar sistemas operativos huesped (Guest), sin tener que modificarlos, sobre un sistema anfitrión (Host), utilizando en medio un Hypervisor o Virtual Machine Monitor que permite compartir el hardware real. Esta capa intermedia es la encargada de monitorizar los sistemas huesped con el fin de capturar determinadas instrucciones protegidas de acceso al hardware, que no pueden realizar de forma nativa al no tener acceso directo a él.
Su principal ventaja es que los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma, aunque como inconveniente frente a la emulación, el sistema operativo debe estar soportado en la arquitectura virtualizada.

En lo que respecta al rendimiento, éste es significativamente mayor que en la emulación, pero menor que en una plataforma nativa, debido a la monitorización y la mediación del hypervisor. Sin embargo, recientes incorporaciones técnicas en las plataformas x86 hechas por Intel y AMD, como son Intel VT y AMD-V, han permitido que soluciones basadas en la virtualización completa se acerquen prácticamente al rendimiento nativo.

Un par de ejemplos significativos son VMware y KVM.

Hay que tener en cuenta también que la virtualización completa no se refiere a todo el conjunto de hardware disponible en un equipo, sino a sus componentes principales, básicamente el procesador y memoria. De esta forma, otros periféricos como tarjetas gráficas, de red o de sonido, no se virtualizan. Las máquinas huesped no disponen de los mismos dispositivos que el anfitrión, sino de otros

virtuales genéricos. Por ejemplo, si se dispone de una tarjeta nVidia GeForce en el anfitrión, los equipos huesped no verán esta tarjeta sino una genérica Cirrus.

Paravirtualización

La paravirtualización surgió como una forma de mejorar la eficiencia de las máquinas virtuales y acercarlo al rendimiento nativo. Para ello se basa en que los sistemas virtualizados (huesped) deben estar basados en sistemas operativos especialmente modificados para ejecutarse sobre un Hypervisor. De esta forma no es necesario que éste monitorice todas las instrucciones, sino que los sistemas operativos huesped y anfitrión colaboran en la tarea.
Uno de los componentes más destacados de esta familia es XEN, el cual fue mi principal candidato durante bastante tiempo. Permite paravirtualización utilizando sistemas operativos modificados, y virtualización completa sobre procesadores con tecnología Intel-VT o AMD-V. Para la gestión de las máquinas virtuales existen aplicaciones propietarias e incluso alguna open-source como ConVirt, que permite gestionar también desde un único sitio las máquinas virtuales de diferentes servidores, realizar tareas sobre ellas, o modificar sus configuraciones.

Cabría destacar otro tipo de productos para virtualizar que son aplicaciones de escritorio para virtualizar (OS level Virtualization), como Vmware-player y VirtualBox. Estos tipos de virtualización están muy bien para virtualizar Sistemas Operativos en nuestro escritorio y hacer pruebas puntuales pero no para estar siempre ejecutándose. Ya que son programas que requieren tener las X (Desktop), comparten los recursos del Host y no están indicados para ser ejecutados con un servicio o demonio del sistema en un servidor y su rendimiento es menor.

Características de KVM

Como he explicado en uno de los puntos anteriores, KVM es uno de los productos que en estos momentos ofrece virtualización completa, permitiendo a priori ejecutar, como huesped, cualquier sistema operativo para plataformas x86 sin tener que haberlo modificado previamente. Para ello se basa en las nuevas instrucciones Intel-VT y AMD-V, por lo que será necesario disponer de un equipo con un procesador que las soporte para poder utilizarlo.

Sin embargo, esto es algo que también permite XEN, entonces, ¿qué ventaje tiene KVM?. Pues la principal ventaja es que viene incluido como un módulo del kernel desde la versión 2.6.20 de Linux, por lo que no es necesario instalar ningún software específico. Si se dispone de una máquina con el procesador adecuado y se instala un linux basado en este kernel, se dispondrán automáticamente de los servicios de virtualización. Es de esperar por tanto, que esta tecnología evolucione bastante rápido y que aparezcan herramientas que permitan sacarle más provecho.

El funcionamiento de KVM se basa en que el módulo introduce un nuevo modo de ejecución aparte de los habituales kernel y user mode, con el nombre “guest mode”. Este modo se utiliza para ejecutar todo el código de los huesped que no sea de entrada/salida (I/O), mientras que el modo “user mode” se utiliza para la I/O de los huesped.

Como nota final, KVM utiliza también QEMU para realizar la virtualización, pero no como emulación, sino únicamente como herramienta para realizar la carga de las máquinas virtuales.

Instalación

Pasos preliminares

Primero para comprobar que nuestra máquina soporta virtualización ejecutamos el siguiente comando:

```
kvm-ok
 INFO: Your CPU supports KVM extensions
 INFO: /dev/kvm exists
 KVM acceleration can be used
```

Si no soporta virtualización saldría:

```
INFO: Your CPU does not support KVM extensions
KVM acceleration can NOT be used
```

También lo podemos comprobar ejecutando:

```
egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

Hay que usar un Host 64 bits siempre que sea posible, ya que en uno de 32 bits estaremos limitados a poner un máximo de 2Gb de

RAM por máquina virtual (limite en 64 bits?) y un host 64 bits puede albergar guests 64 y 32 bits, mientras que un un host 32 bits, solo puede albergar máquinas virtuales de 32 bits.

Para comprobar que tenemos un procesador 64 bits, ejecutamos:

```
grep ' lm ' /proc/cpuinfo
```

Si no sale nada es que no tenemos un host 64 bits.

También hemos de comprobar que tenemos una distribución de 64 bits. Para ello ejecutamos:

```
uname -m
```

y nos ha de salir 'x86_64?.
Si ves que te sale 386, i486, i586 o i686, es que estas ejecutando una distro con un kernel 32 bits.

Instalación

Se utiliza Ubuntu Server 10.04

Para instalar KVM en Ubuntu lo podemos hacer seleccionándolo durante la instalación (Virt Host o algo así) o si ya tenemos instalado un servidor instalamos este meta-paquete:

```
sudo aptitude install ubuntu-virt-server
```

Que instala entre otros los paquetes 'kvm libvirt-bin ubuntu-vm-builder bridge-utils'.
- kvm – es la solución de virtualización que se compone de una serie de módulos para el núcleo linux.Usa QEMU
- libvirt-bin – son unas herramientas para administrar kvm y libvirt.
- bridge-utils – sirve usar la red local con la red virtual

Podemos comprobar que todo ha ido bien ejecutando:

```
virsh -c qemu:///system list
```

Y nos debe salir:

```
 Id Name State
----------------------------------
```

NOTA:
Nos aseguramos que el usuario que va a crear y gestionar las VM's sea incluido en el grupo libvirtd.
En Lucid por defecto lo hace si no, lo haríamos a mano:

```
adduser $USER libvirtd
```

Configurar la red

Hay varias maneras de permitir acceder al interfaz de red físico del host desde las diferentes VM's.
La forma predefinida es en modo 'usermode', el cual usa el protocolo SLIRP para acceder por NAT al la red (de esta forma las VM's no pueden tener el mismo rango de red que el host).
La mejor forma es hacer un bridge. De esta forma la red de las diferentes VM's pueden estar en el mismo rango de red que el Host.
Para ello instalamos las bridge-utils:

```
aptitude install bridge-utils
```

Y configuramos la red:

```
#The loopback network interface
auto lo
iface lo inet loopback
```

```
auto br0
iface br0 inet static
address 192.168.1.3 # pon los datos que tengas tu
network 192.168.1.0
netmask 255.255.255.0
broadcast 192.168.1.255
gateway 192.168.1.1
bridge_ports eth0
bridge_fd 9
bridge_hello 2
bridge_maxage 12
bridge_stp off
```

Finalmente reiniciamos la red con un:

```
/etc/init.d/networking restart
```

Y ya deberíamos ver un interfaz de red br0

Instalar y manejar VM's

Para crear y manejar las máquinas virtuales (guests) podemos hacerlo gráficamente o por consola.

Por línea de comandos (virt-install y virsh):
Tenemos las siguientes herramientas:

ubuntu-vm-builder – es una herramienta desarrollada por Canonical para crear maquinas virtuales Ubuntu. Realmente lo que instala es el paquete # # python-vm-builder
python-vm-builder – Script que automatiza la creacción de una VM basada en Linux (por línea de comandos)

Las instalamos con:

```
aptitude install ubuntu-vm-builder
```

Creación

Creación de VM's por línea de comando – virt-install:
Permite crear máquinas de todo tipo.
Virt-install es un Script desarrollado por Red Hat que automatiza la creacción de una VM de cualquier tipo (Linux, Windows, BSD…).
Virt-install utiliza la librería 'libvirt'.

Lo instalamos con:

```
aptitude install virtinst
```

Para ver todas las opciones hacemos un 'man virt-install'

Para crear una VM de Lucid Server 64bits haremos:

```
virt-install --connect=qemu:///system --name=lucidVirtual --ram=512 --vcpus=1 --check-cpu --os-type=linux --hvm --vnc
--accelerate --disk=/var/lib/libvirt/images/nombredelaimagen.iso,size=10,sparse=true
--cdrom=/var/lib/libvirt/images/ubuntuserver1004.iso --network=bridge:br0
```

Lo cual significa:
–name=lucid -> El nombre que le vamos a dar a la VM
–ram=512 -> La RAM que le vamos a asignar (se puede modificar posteriormente)
–vcpus=1 -> CPU's virtuales que le vamos a asignar
–check-cpu -> Comprueba que el nº de CPU's asignadas no exceda el del Host y si es así nos avisa
–os-type=linux -> Tipo de Sistema Operativo que vamos a instalar (linux o windows)
–os-variant=OS_VARIANT -> Variante para ajustar mas la instalación. Si se pone esta opción no hace falta la anterior

(–os-type). Hay varios tipos ‘winxp’, ‘ubuntukarmic’ (consultar man).
–hvm -> Que utilice full-virtualization. Si quisiésemos usar paravirtualizacion usaríamos la opción –paravirt
–acelerate -> Esta opción se usa ya por defecto y está deprecated (se puede omitir)
–disk=/var/lib/libvirt/images/lucid.img,size=4,sparse=true -> El disco que vamos a crear, size=4 el tamaño en GB, sparse=true que no reserve todo el espacio y que lo vaya asignando a medida que crece el disco (.img)
–cdrom=/var/lib/libvirt/images/ubuntu-10.04-server-amd64.iso -> La iso o el cd si está puesto y montado en la bandeja (
–network=bridge:br0 -> Que utilice como red el dispositivo bridge ‘br0? para poder tener la red en el mismo rango que el host

Mas opciones:
–sound -> Attach a virtual audio device to the guest.
–vnclisten=VNCLISTEN -> Por defecto solo escucha en localhost (127.0.0.1), pero podemos hacer que se puedan conectar desde una ip en concreto, varias separadas por comas o rango entero.

Saldría algo como:

```
Empezando la instalación...
Creando archivo de almace 100% |=========================| 4.0 GB 00:00
Creando dominio... 0 B 00:00
No se ha podido conectar a una consola gráfica: no está instalado el virt-viewer. Por favor, instale el paquete "virt-viewer"-
La instalación del dominio continúa en progreso. Puede reconectarse a
la consola para completar el proceso de instalación.
```

Tal y como indica la instalación empieza pero como no tenemos X, no puede mostrarnos el proceso. Si tuviesemos X en el mismo servidor, sería tan fácil como:

```
sudo virt-viewer vm10
```

Desde otro equipo podemos instalar el paquete virt-viewer y conectarnos haciendo:

```
sudo virt-viewer --connect qemu+ssh://usuario@ip /system lucid
```

Ahora si se abre la ventana y podemos continuar con la instalación.
Nota: Una vez instalado no reinicia la VM. Esta queda apagada. Habría que iniciarla (lo explico mas abajo).

También podemos usar virt-install en modo prompt y nos va solicitando las opciones:

```
virt-install --prompt
```

Gestión de las VM’s por línea de comando – virsh:
La creación de la máquina virtual ha generado un archivo xml con el nombre de la VM en /etc/libvirt/qemu/

Para arrancar y trabajar con mv, ver: [[gestion-maq-virt| Gestión de máquinas virtuales]]

ACPI

Para que las VM’s Linux apaguen correctamente hay que instalar el demonio ‘acpid’ en la máquina cliente:

```
aptitude install acpid
```

Destrucción

Si una VM no responde podemos destruir el dominio:

```
virsh -c qemu:///system destroy 6
```

Si hemos definido una VM y la queremos borrar ‘completamente’ hemos de ejecutar el siguiente comando:

```
#virsh destroy guindous
  #virsh undefine guindous
```

Esto significa que ya podemos volver a crear un guest que se llame ‘guindous’.

Para poner y quitar un CD o una iso:
Vamos a los xml de configuración de cada guest en /etc/libvirt/qemu y lo editamos.

Donde pone:

```
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Ponemos:

```
<disk type='file' device='cdrom'>
<driver name='qemu'/>
<source file='/var/lib/libvirt/images/winxp.iso'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Si en vez de una iso queremos poner el cd o dvd que haya en la bandeja del host, ponemos:

```
<disk type='file' device='cdrom'>
<driver name='qemu'/>
<source file='/media/cdrom'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Aumentar tamaño del disco duro

Primero que nada debemos crear un disco virtual con el nuevo tamaño, dentro de la carpeta donde se encuentra el disco de nuestra maquina virtual.

```
qemu-img create -f qcow2 destino.img 20G
```

Ahora nos vamos a la carpeta /etc/libvirt/qemu, allí se encuentran los xml con la configuración de cada maquina.
Editamos el de nuestra pc (ej. facundo.xml)

```
cd /etc/libvirt/qemu
vim facundo.xml
```

En la configración debemos hacer dos cambios importantes:

Agregar el disco duro nuevo

Obs: Para que el manejador gráfico funcione correctamente, hay que instalar las imágenes de disco en: /var/lib/libvirt/images/

Luego de la linea donde se encuentra el disco hda, agregamos otro de la siguiente manera:

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/pc.img'/>
<target dev='hda' bus='ide'/>
</disk>

<disk type='file' device='disk'>
```

```
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/facundo.img'/>
<target dev='hdb' bus='ide'/>
</disk>
```

También cargamos la iso del clonzilla, para clonar de un disco al otro:

```
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/clonezilla.iso'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Reiniciamos el servicio

```
/etc/init.d/libvirt-bin restart
```

Luego nos conectamos

```
virt-viewer --connect qemu+ssh://usuario@ip /system lucid
```

Entramos en el clonzilla y clonamos la partición original a la nueva
Luego de clonada la partición, debemos redimenzionarla para que tome todo el tamaño disponible del disco, para ello utilizaremos la iso de gpartedlive.

Procedemos a editar el xml

```
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/gparted.iso'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Reiniciamos el servicio

```
/etc/init.d/libvirt-bin restart
```

Y nos volvemos a conectar

```
virt-viewer --connect qemu+ssh://usuario@ip /system lucid
```

Entramos al live de gparted y agrandamos la partición a el tamaño total del disco.
Apagamos la maquina y volvemos a editar el xml, para dejarlo en condiciones óptimas para su funcionamiento (Dejamos solo el disco nuevo y desmontamos la iso)

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw'/>
<source file='/var/lib/libvirt/images/facundo.img'/>
<target dev='hda' bus='ide'/>
</disk>
<disk type='file' device='cdrom'>
<driver name='qemu' type='raw'/>
<target dev='hdc' bus='ide'/>
<readonly/>
</disk>
```

Reiniciamos y listo!

Aumentar el tamaño de disco raw

Creamos un nuevo disco del tamaño que necesitemos:

```
sudo dd if=/dev/zero of=zeros.raw bs=1024k count=4096
```

Fusionamos los dos discos:

```
cat alpha-clone-2.img zeros.raw > clon.raw
```

Luego iniciamos el gparted en la nueva partición y agrandamos la partición.

Agregar usuarios al grupo kvm

```
adduser $USER libvirtd
```

Enlaces de interés

- Documentación oficial KVM:
  - http://www.linux-kvm.org/page/Main_Page
- Otra
  - http://doc.ubuntu.com/ubuntu/serverguide/C/libvirt.html#libvirt-virt-viewer
  - http://www.gonzalomarcote.com/blog/?p=57
  - http://josecely.tecsua.com/?p=40
  - https://help.ubuntu.com/community/KVM/CreateGuests