INSTITUTO LINUX

05:06

VIRTUALIZACIÓN EN REDES LINUX

KVM

EXPERTO EN ADMINISTRACIÓN DE REDES LINUX

-. MÓDULO 05 · CAPÍTULO 06 .-

LINUX FOUNDATION AUTHORIZED TRAINING PARTNER
Linux Professional Institute
UTN FACULTAD REGIONAL DELTA
Universidad Nacional de San Luis
debian

Capítulo 06 - **KVM**

ÍNDICE

Capítulo 06 - **KVM**

KVM como ejemplo de Full Virtualization

KVM hacen referencia a las de **Kernel Virtual Machine**. Esta es una solución para full virtualization sobre sistemas GNU/Linux para x86 que contengan las extensiones de Virtualización de Intel VT o AMD-V.

Está formado por un módulo del núcleo que tiene el nombre de **kvm.ko** y un conjunto de herramientas que se encuentran en el espacio de usuario siendo en su totalidad software libre.

KVM me va a **permitir ejecutar máquinas virtuales utilizando imágenes de disco que van a contener los sistemas operativos sin ninguna modificación,** ya que estamos trabajando con este tipo de modo de Virtualización.

Recordarán que en el tipo anterior de virtualización aprendido, OS virtualization, ya de por si iniciábamos la máquina con un Kernel modificado invalidando la posibilidad de que en algún container le instaláramos algún módulo del kernel. Mucho menos pensar en instalar algún otro sistema operativo que no sea GNU/GNU/Linux

Al ser full virtualization, **cada VM va a tener su propio hardware virtualizado**, o dicho de otra manera, la máquina virtual que se encuentre corriendo no va a notar la diferencia entre un hardware real y el virtualizado ya que contará con: Una tarjeta de red, disco, CPU, memoria, etc.

La “librería” libvirt

Libvirt es una **colección de software** (por eso el título entre comillas) que provee una manera conveniente de **administrar máquinas virtuales**, junto con otro tipo de funcionalidades como ser almacenamiento o la administración de interfaces de red.

Un punto interesante a tener en cuenta es que incluye una API junto con un demonio y una CLI que nos permitirá administrar las máquinas virtuales. Pero no solo lo podremos hacer mediante esta interfaz de comandos, también lo vamos a poder hacer mediante una interfaz gráfica gracias al uso de la API y el daemon.

Uno de los objetivos de libvirt es proveer una manera simple de administración de Hypervisors.

Por ejemplo, el comando **virsh list –all** que ejecutamos con el CLI de libvirt puede utilizarse para listar máquinas de varios Hypervisors como ser KVM o Xen. O sea, si el hypervisor utiliza libvirt, entonces vamos a poder utilizar la APÏ de libvirt para controlar ese hypervisor, por lo que nos desligamos de la relación hypervisor-set de comandos.

Aparte, la API no es solo local. Esto significa que **podremos acceder a cualquier máquina que se encuentra corriendo el demonio libvirt**.

Qemu

Qemu es un emulador de procesadores con capacidades de virtualizar y básicamente emula un sistema operativo dentro de otro sin tener que reparticionar el disco duro.

Esto significa que va a poder utilizar cualquier ubicación del mismo y lo vamos a poder administrar mediante el **Qemu manager**.

Este puede operar de dos maneras. La primera es modo de emulación del modo usuario; pero nosotros utilizaremos el modo de emulación completo, ya que necesitaremos de emular un sistema completo, incluyendo procesador y varios periféricos.

Qemu, libvirt y KVM

Pero, ¿en qué punto se relacionan Qemu con KVM y libvirt? **Hypervisor es el agente**, como sabemos, **que administra las máquinas virtuales**. Particularmente nos ayudará a crear las mismas. **Qemu y KVM**, ambos, son capaces de actuar como hypervisor; pero **se utilizan en combinación**, ya que Qemu es menos efectivo en lo que es el aprovechamiento del hardware virtualizado. Es por ese motivo que verán muchas veces que Qemu se utiliza por si solo.

KVM ayuda a Qemu a acceder a los features del hardware en diferentes arquitecturas, y aparte agrega la aceleración al proceso Qemu.

Bien, ahora llegó el momento de incluir a libvirt en todo esto. **Libvirt** es, entre otras cosas, una **librería para administrar lo que es virtualización**; pero ¿qué rol cumple en esto?, básicamente administra tanto KVM como Qemu. Consiste, como mencionamos antes, en tres cosas fundamentales: **API**, **libvirtd** y **virsh**.

También como mencionamos antes, cualquier hypervisor que implemente libvirt lo vamos a poder administrar con cualquier cosa que interactúe con su API.

Entonces, en resumidas cuentas Qemu utiliza KVM para funcionar mejor, ambos funcionan como un hypervisor, administrado mediante libvirt.

Revisando extensiones

Algo que nos incrementará la performance del uso de las máquinas virtuales en full virtualization es si nuestro procesador tiene las extensiones para virtualización. Por lo que antes que nada **vamos a ver si tenemos la extensión para virtualizacion dentro del microprocesador**:

```
egrep '^flags.*(vmx|svm)' /proc/cpuinfo --color
```

Aquí estamos observando la información cargada en el archivo **cpuinfo**, que como se deduce, contiene información respecto de lo que es el microprocesador. VMX se refiere a la extensión para los microprocesadores Intel y SVM para los microprocesadores AMD.

Si no llegamos a tenerlas no existe inconveniente, ya que vamos a poder virtualizar de la misma manera; pero nuestro rendimiento se verá impactado.

También cabe aclarar que **estas extensiones deben ser habilitadas en la BIOS del sistema**.

Instalación de paquetes

Vamos a instalar los paquetes necesarios para poder iniciar KVM. Aclaración, vamos a utilizar KVM; pero recuerden que estamos utilizando el trinomio KVM+Kemu+Libvirt:

```
# apt-get -y install kvm libvirt0 python-libvirt virt-manager
virt-viewer virtinst
```

Para ahondar un poco más en el comando anterior, los paquetes virtinst y virtviewer son dos aplicaciones que nos permitirán instalar por línea de comandos o administrar mediante una interfaz gráfica que hará uso del demonio y la API libvirtd.

Recuerden, como libvirt controla al hypervisor, cada vez que una aplicación haga uso de su API, estará administrando el hypervisor.

Durante la instalación, va a chequear que la extensión del microprocesador se encuentre activado.

Luego **debemos instalar Qemu para crear nuestros discos virtuales**. Como comentamos anteriormente, Quemu es básicamente un emulador de procesadores y también tiene capacidades para virtualización dentro de un sistema operativo, por eso es que Qemu utiliza ciertas cosas de KVM.

Como venimos diciendo, el objetivo es emular un sistema operativo dentro de otro sin tener que reparticionar el disco duro, empleando para su ubicación cualquier directorio dentro de este.

Vamos a utilizar **Qemu en modo de emulación completa**, esto nos permite emular un sistema completo:

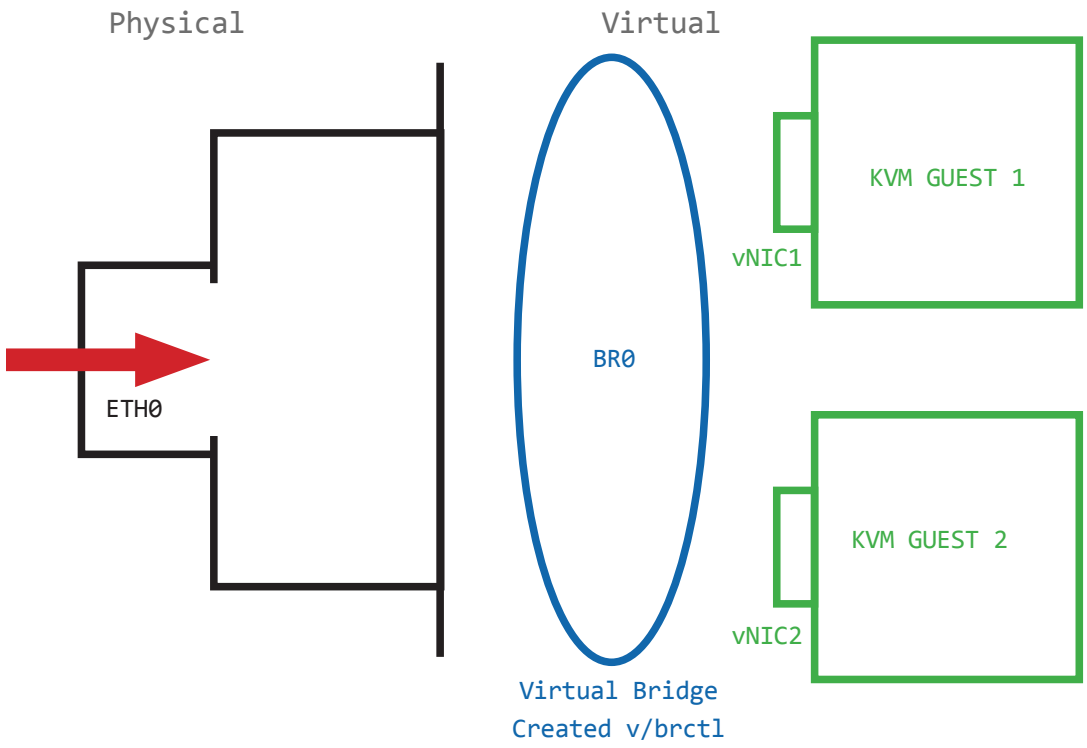
```
# apt-get -y install qemu-kvm
```

Bridging

Si pretendemos que nuestras Vms sean a nivel de red como un dispositivo físico más, debemos **instalar el paquete bridge-utils**, que nos **permitirá crear las interfaces de bridging**:

```
# aptitude install bridge-utils
```

Por ahora simplemente entendamos que las interfaces de bridging son interfaces virtuales que se encuentran conectadas de manera directa con una interfaz física.



Más adelante veremos cómo funciona internamente bridging. Primero vamos a detener networking.

Pero ojo, hay que tener conexión desde consola y no remoto porque nos podemos llegar a quedar afuera.

Abrimos el archivo para editarlo:

```
# vi /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

# Bridged
auto br0
iface br0 inet static

        address 192.168.1.128
        netmask 255.255.255.0
        gateway 192.168.1.1
        bridge_ports eth0
```

Las líneas que contienen los parámetros a resaltar son dos:

- **auto br0**: aquí pueden ver que el nombre de esta interfaz virtual es br0. Podemos colocarle el nombre que queramos
- **bridge_ports eth0**: aquí le estamos indicando que la interfaz virtual br0 va a estar asociada a la física eth0.

Para quienes hayan utilizado VirtualBox, esto es lo mismo que utilizar una interfaz Bridge. Observen que en Virtualbox (hypervisor) le decimos que la interfaz de la máquina virtual esté asociada a la eth0 en modo Bridge. Aquí estamos haciendo eso.

Una vez hecho esto, debemos de **reiniciar la máquina**:

```
# reboot
```

Y si prestamos atención durante el inicio podemos ver que en un momento hace referencia al módulo **kvm.ko**.

Una vez finalizado el reinicio, lo que va a suceder es que va a **bridgear la eth0 a br0**. Esto lo podremos ver fácilmente al ejecutar el comando **ifconfig**:

```
# ifconfig
```

También podemos observar que asignamos una IP a br0 y eth0 sin IP. Por lo que eth0 pasaría a funcionar a nivel de capa 2, y la IP estaría asignada a br0.

Luego, vamos a descargar una iso de GNU/Linux para instalar. Primero accederemos a la carpeta por defecto donde se encuentran todas las imágenes:

```
# cd /var/lib/libvirt/images/
```

Y vamos a descargar la imagen ISO de la misma. Esta URL la pueden obtener de la página de Debian.

Digo Debian en este caso como ejemplo; pero es la imagen ISO de cualquier sistema operativo ya que es la misma imagen que quemamos en un dispositivo óptico o USB. Recuerden, esto es full virtualization y estamos emulando todo:

```
# wget http://cdimage.debian.org/cdimage/archive/<versión>/amd64/
iso-cd/debian-<version>-amd64-netinst.iso
```

Algo muy interesante en esta técnica de virtualización es que el disco es una imagen del mismo (como comentamos) que se encuentra en un archivo. Dicho en pocas palabras el archivo representa al disco. Por lo que lo que hagamos con ese archivo es como si lo hiciéramos con el disco físico. **Tener un backup de ese archivo es lo mismo que tener un backup del disco físico.**

Estas imágenes tienen distinto formato, en particular vamos a utilizar **qcow2**:

```
# qemu-img create disk.jessie.qcow2 -f qcow2 30G
```

Donde:

- disk.jessie.qcow2: es el nombre de la imagen del disco
- -f: define el formato de la imagen
- 30G: el tamaño de la imagen. Cuidado, porque puede ser que reserve los 30G directamente o tenga 30G; pero el espacio real en el disco físico lo vaya tomando a medida que lo ocupa.

La extensión **qcow2** hace a la forma en la cual va a almacenar información en los discos de las máquinas virtuales, o imágenes de disco:

- **QCOW**: significa “qemu copy on write” y es una estrategia en la cual se va alocando datos a medida que va necesitando.
- **VMDK**: es el formato que utiliza vmware
- **VHD**: es un formato que representa Virtual Hard Disks.
- **RAW**: es un dd prácticamente del disco, o dicho de otra manera, son los bits en 1s y 0s crudos.
- **VDI**: es el formato que utiliza Virtualbox.

Entonces, muchas veces la pregunta es “¿puedo pasar una máquina de vmware a GNU/Linux?” y la respuesta es sí, si el hypervisor puede leer ese formato de disco. Esto funciona ya que como les comenté antes, en full virtualization la imagen del disco representa al disco. Es lo mismo que tomar un disco en un equipo y conectarlo en otra máquina.

Por último; pero no menos importante, vamos a **activar el forward de paquetes**:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Instalación de la primera máquina virtual

Esta técnica nos va a acercar más a lo que ya vinimos viendo con OpenVZ, ya que crea una máquina virtual nueva utilizando una herramienta denominada **virt-install** que nos va a servir para **instalar Vms nuevas**; pero es requisito tener redireccionado el X-Windows a nuestra máquina de escritorio con gestor de ventanas. Esto se logra conectándose mediante **ssh -X**. También más adelante vamos a ver una GUI para administrar las máquinas virtuales de una manera más simple, junto con una shell que nos permitirá administrar las mismas.

```
# virt-install --name kvm101 \
--vcpus=2 \
--ram=2048 \
--os-type=linux \
--os-variant=debiansqueeze \
--hvm \
--connect=qemu:///system  \
--network bridge:br0 \
--cdrom=/var/lib/libvirt/images/<versión>-amd64-netinst.iso \
--disk path=/var/lib/libvirt/images/disk.jessie.qcow2,size=30 \
--vnc \
--keymap=es \
--force
```

Donde:

- **name:** nombre del Guest
- **arch:** arquitectura del Guest.
- **vcpus:** cantidad de CPUs asignado al Guest.
- **ram:** MB de RAM asignado al Guest.
- **os-type** y **os-variant:** distro y versión. Esto es nada más que representativo. man virt-install.
- **hvm:** Hosted Virtual Machine o full virtualization.
- **connect:** a que Hypervisor nos conectaremos. Como pueden ver, aquí será mediante qemu al systema local. Recuerden que podríamos conectarnos a un hypervisor remoto.
- **network bridge:** que interfaz de bridging utilizaremos en el Guest.
- **cdrom:** la ISO del sistema operativo a instalar.
- **disk path=x,size=y:** ruta y tamaño de la imagen para el Guest.
- **accelerate:** si vamos a utilizar las capacidades de aceleración.
- **vnc:** a donde exportaremos la consola del VM.
- **keymap:** el layout del teclado.

Esto nos abrirá una nueva ventana donde instalaremos Debian. Una vez creado, de manera similar a OpenVZ, KVM guarda la configuración de la máquina virtual dentro de un archivo de configuración en la siguiente carpeta:

```
/etc/libvirt/qemu/
```

Como pueden ver, aquí dentro se encuentran los parámetros de configuración que toma la máquina virtual al momento de iniciar:

```
vi kvm101.xml
```

Si vemos el contenido observaremos que se encuentra tageado y hay elementos como la macaddress de la placa, memoria y hasta el formato del disco.

Hay que tener cuidado ya que todo lo que modifiquemos aquí, será persistente la próxima vez que reiniciemos la máquina virtual por lo que modifiquemos con responsabilidad. Igualmente más adelante veremos una herramienta gráfica para poder administrar más simple lo que son las máquinas virtuales.

Operando desde la consola

Vamos a tener una consola mediante la cual vamos a poder operar con las máquinas virtuales. Accedemos a la misma ejecutando “virsh”, o sino accionando virsh comando. Por ejemplo, para listar las máquinas virtuales que tenemos encendidas haremos:

```
# virsh list
```

Pero si quisiéramos **observar la totalidad de las máquinas virtuales** podremos ejecutar:

```
# virsh list --all
```

- El estado running se refiere a los guests que se encuentran activos dentro del CPU.
- El estado blocked se encuentran bloqueadas por lo que no se encuentran corriendo, ni tampoco se pueden iniciar. Esto es causado por algún guest que espera algún I/O o en modo sleep.
- El estado paused lo vamos a encontrar si un administrador coloca en pausa al VM. Esto se logra mediante el comando vrish suspend. Cuando esta pausado consume memoria y otros recursos; pero no schedulea recursos de CPU a través del hypervisor.
- El estado shutdown es para guests en proceso de apagado. El guest envía una señal de shutdown y se encuentra en el proceso de apagado.
- Dying es cuando está en proceso de finalización. A diferencia del anterior puede ser que se deba a un crash o un intento no exitoso de apagado.
- Crashed se refiere a un guest que no se encuentra corriendo, o fallo en algún momento. Esto solo pude ocurrir si el gust está configurado para que no reinicie ente un crash.

Así como el CT0 se refiere al hnode, el Domain-0 hace referencia a la misma maquina.

Si quisiéramos **ingresar a la consola** ejecutamos:

```
#virt-viewer kvm101
```

Para **apagar, prender o reiniciar un dominio**:

```
# virsh start kvm101
# virsh shutdown kvm101
# virsh reboot kvm101
```

Y si lo quisiéramos **suspender**:

```
# virsh suspend kvm101
# virsh resume kvm101
```

Podemos **ver la info del dominio** con:

```
# virsh dominfo kvm101
```

Vimos donde se encuentra el archivo de **conf** de una máquina virtual en cuestión; pero también podemos **acceder y modificar** dicho contenido:

```
# virsh edit kvm101
```

Si quisiéramos **información del nodo en el cual se encuentra corriendo**:

```
# virsh nodeinfo
```

Podemos **ver específicamente la información del CPU**:

```
# virsh vcpuinfo kvm101
```

Si queremos **ver el output referente a información de networking**:

```
# virsh net-list
```

Para **guardar el estado de un Guest**:

```
# virsh save kvm101 /tmp/bkp_kvm101
```

Y si quisiéramos **recuperar ese estado**:

```
# virsh restore /tmp/bkp_kvm101
```

Para **eliminar una máquina virtual de manera graceful**, primero debemos de pasarle el undefine. Esto hace un "undefine" de las máquinas cuyo xml fue definido previamente. Esto se realizó automáticamente en la instalación:

```
# virsh stop kvm101
# virsh undefine kvm101
# virsh destroy kvm101
```

Si quisiéramos **eliminar el Guest** de manera brusca:

```
# virsh destroy kvm101
```

Este comando realiza un apagado brusco y detiene la sesión de este guest domain por lo que existe una potencial posibilidad de que algún archivo quede corrupto. Esta opción solo debe de utilizarse si deseamos finalizarlo cuando la el guest no responde.