Linux con KVM

Sumario

Introducción

KVM (Kernel-based Virtual Machine) es una tecnología de hipervisor integrada en el kernel de Linux. Con esta tecnología podemos crear y gestionar

máquinas virtuales bajo el paradigma de la ?full virtualization?. Aunque hablemos de KVM realmente deberíamos hablar de Qemu/KVM. **Qemu** es una plataforma de simulación de hardware virtual, funcionando de modo que, intercepta las instrucciones para la CPU virtual y utiliza el sistema operativo anfitrión para ejecutar esas instrucciones en la CPU real. Qemu es el encargado de proporcionar recursos, del nodo o host, a los domains, MVs o guests. KVM cumple una función similar, sin embargo ambos operan juntos por una cuestión de eficiencia y compatibilidad. Qemu es compatible con muchas arquitecturas y permite acceder a múltiple hardware; mientras que KVM ofrece el rendimiento necesario para ello, mejorando las características de velocidad del primero.

Más específicamente **KVM es el módulo del kernel** de Linux que permite acelerar la ejecución de una máquina virtual. Sin esta aceleración la ejecución de las instrucciones de la MV en la CPU virtual tienen que ser traducidas por el hipervisor a la CPU física. KVM permite utilizar las características de algunas CPU modernas de mapear partes de la CPU físicas directamente a la máquina virtual. De este modo las instrucciones ejecutadas por la CPU virtual pasan sin la sobrecarga de traducción del hipervisor a la CPU física. Esta característica tiene como consecuencia que, para poder obtener todo el rendimiento de aceleración de KVM, **la CPU virtual y la CPU física deben de tener la misma arquitectura**. De no ser el caso, solo podríamos utilizar Qemu sin KVM, con la consiguiente pérdida de rendimiento asociada. La combinación de Qemu y KVM permite implementar un hipervisor de tipo 1, debido a las características de aceleración y mapeo directo de las instrucciones virtuales/físicas que este último proporciona.

También conviene hablar de **libvirt**. Se trata de un API o librería de acceso a características de virtualización compatible con múltiples tecnologías de hipervisor, incluyendo por supuesto Qemu/KVM. A través de esta librería, y de su interfaz cliente virsh, podremos gestionar los recursos virtualizados y la creación y gestión de MVs. Podemos interpretar **Qemu/KVM** como una tecnología de base para implementar un **hipervisor de tipo 1** sobre el kernel de Linux. De este modo dispondremos de características que proporcionarán un rendimiento elevado, próximo al del entorno de ejecución ?bare metal?.

Con Qemu/KVM vamos a poder ejecutar cualquier tipo de sistema operativo, Windows, Linux, compatible con la arquitectura de la CPU.

Para ejecutar Qemu/KVM necesitamos que **el procesador soporte extensiones de virtualización** en su juego de instrucciones, podemos comprobarlo ejecutando el comando

```
egrep 'svm|vmx' /proc/cpuinfo --color
```

El resultado debería mostrar varias líneas con el texto buscado resaltado en color. Si este es el caso entonces nuestro procesador soporta KVM. Los procesadores Intel mostrarán el texto **vmx** resaltado y los procesadores AMD mostrarán el texto **svm** resaltado.

NOTA: Es importante comprobar que en la BIOS están activados las instrucciones virtuales VT/x, de lo contrario KVM no funcionará

También deberíamos de comprobar si los módulos correspondientes están cargados

```
lsmod | grep kvm
```

Debería mostrar una salida no vacía

Por último el comando

```
virsh capabilities
```

Mostraría un listado de las capacidades de virtualización de nuestro sistema

Plataformas

Usaremos como base para la realización de las pruebas y los comandos explicados

- **Debian 9 (Stretch) versión desktop**: Podremos utilizar todos los comandos por consola, por tanto no es estrictamente necesario que sea una versión desktop, sin embargo, si queremos utilizar la interfaz gráfica de gestión virt-manager, sí sería necesario este requisito

Ejecución KVM nested

Con esta característica se permite la ejecución de instrucciones KVM dentro de máquinas virtuales KVM, lo cual abre un abanico de posibilidades interesantes a la ejecución de máquinas virtuales dentro de máquinas virtuales, pudiendo resultar útil para pruebas y formación. Un ejemplo típico de lo anterior sería la posibilidad de ejecutar la plataforma Proxmox en una máquina virtual, de modo que podremos crear máquinas virtuales con KVM desde esa plataforma, no siendo necesario hacer una instalación ?bare metal? de Proxmox.

Veamos el procedimiento

Activar KVM nested en el anfitrión

Comprobamos la salida del comando

```
cat /sys/module/kvm_intel/parameters/nested
```

Si es ?N?, entonces activamos el soporte para que el procesador pueda ejecutar las instrucciones necesarias para KVM en la máquina virtual KVM. Ejecutamos como root:

```
echo 'options kvm_intel nested=1' >> /etc/modprobe.d/qemu-system-x86.conf
```

Reiniciamos la máquina

Para poder utilizar en una máquina virtual KVM la anterior característica debemos elegir la opción Copy host CPU Configuration a la hora de definir la CPU virtual de la máquina.

Virt-manager

Virt-manager es una herramienta gráfica de gestión de máquinas virtuales. Es compatible con tecnología de tipo full virtualization KVM y Xen, pero también permite la gestión de containers basados en tecnología LXC y OpenVZ.

El siguiente comando instala virt-manager

```
apt update
apt install virt-manager
```

Tras su instalación podemos lanzarla, como root

```
virt-manager &
```

NOTA

Si queremos ejecutar virt-manager con un usuario no root debemos incluir al usuario en los grupos: libvirt, libvirt-qemu y kvm. Podemos usar el siguiente comando

```
usermod -a -G libvirt,libvirt-qemu,kvm usuario
```

Muchos de los procedimientos ilustrados a continuación podrán ser realizados desde esta herramienta en modo gráfico

Creación de MV

Existen varios modos para crear máquinas virtuales KVM, desde hacerlo con la interfaz gráfica proporcionada por la herramienta virt-manager, hasta, como veremos a continuación, utilizar algún comando específico

Creación de MV con virt-install

```
virt-install --name guest1-deb9 --memory 2048 --vcpus 2 --disk size=20 \
--cdrom /path/to/debian9.iso --network network=default --os-variant debian9
```

Las opciones son autoexplicativas

- name: nombre de la máquina virtual
- memory: cantidad de memoria asignada
- vcpus: número de CPUs virtuales
- disk: establece opciones de disco duro virtual
- cdrom: indica los medios de instalación
- network: indica a que red virtual se va a conectar la máquina. Por defecto usará la network default de tipo NAT. También se puede indicar que se cree el guest sin red, indicando --network none
- os-variant: identifica el tipo de sistema operativo de la MV

Existen otras muchas opciones y posibilidades a la hora de crear una máquina virtual con el comando virt-install. Desde importación de otra maquina, instalación a través de la red, uso de PXE, instalación desatendida, etc.

Creación de MV a partir de archivo xml existente

A veces nos interesa crear MV similares, o con características muy parecidas, las cuales pueden ser tomadas a partir de una MV ya existente. De este modo, duplicamos el xml original, hacemos las modificiones pertinentes en el nuevo archivo y ya podemos crear una nueva MV a partir de él. Veamos como hacerlo. En primer lugar, una aclaración. Cada MV en KVM tiene asociado un número único, UUID, de tipo hexadecimal. Este número debe de ser generado para poder ser utilizado en nuevas MV. Para ello utilizamos la herramienta uuid, sino la tenemos instalada procedemos a su instalación con

```
apt install uuid
```

Tras lo cual podremos generar uuid nuevos con el comando

```
uuid
```

Ahora vamos a volcar el contenido del archivo de configuración xml de alguna de las MV existentes en un nuevo archivo que será modificado apropiadamente, para luego servir como base para la creación de una nueva MV. Usaremos la herramienta virsh dumpxl

```
virsh dumpxml W10-CLIENTE > W10-CLIENTE-2.xml
```

Ahora podremos editar el archivo W10-CLIENTE-2.xml con algún editor de texto. Es importante modificar al menos los 3 primeros campos del archivo

```
        <name>W10-CLIENTE-2</name>
        <uuid>d3212b52-bff8-11e7-84d9-63cfc2c59a3b</uuid>
        <title>W10-CLIENTE-2</title>
```

y también el campo, dentro de la sección disk, el cual hace referencia al volumen asociado

```
        <source file='/var/lib/libvirt/images/win10_2.qcow2'/>
```

El valor uuid del campo <uuid> deberá ser generado previamente con la herramienta uuid y copiado en ese campo.

NOTA: Veremos más adelante como crear volúmenes nuevos a partir de otros

Una vez modificado los elementos del archivo necesarios creamos la nueva MV

```
virsh create W10-CLIENTE-2.xml
```

El comando anterior creará la MV a partir del archivo modificado

virsh

virsh es una potente herramienta de línea de comandos que nos permite gestionar múltiples aspectos de la gestión de máquinas virtuales KVM. Dentro de la terminología de KVM hacemos referencia al anfitrión o host mediante el término node (nodo). Para la gestión con virsh se utiliza ampliamente el concepto de domain (dominio), un domain no es más que una máquina virtual. Veamos un ejemplo

```
virsh start guest1-win10
```

El comando anterior inicia el domain (dominio o máquina virtual guest1-win10) Existen multitud de comandos virsh, los cuales pueden listarse con la opción --help

```
virsh --help
```

Otra opción es entrar en el modo consola de virsh, desde donde pueden invocarse todos los comandos con posibilidad de autocomplección. Para entrar en la consola virsh:

```
virsh
```

Con virsh podemos gestionar:

- domains
- host e hypervisor
- interfaces
- redes
- dispositivos
- almacenamiento (pools y volúmenes)
- snapshots
- secretos (passwords)

Comandos virsh básicos de gestión de MV

Listado de MV

```
virsh list --all
```

Mostrará todas las MV, estén activas o no, sino se incluye la opción --all entonces solo mostrará las iniciadas

Ver el estado de una MV

Como se ha mencionado anteriormente el concepto domain hace referencia a las MV. Para ver el estado de un domain ejecutamos el comando dominfo de virsh

```
virsh dominfo guest01-win10
```

Arranque y parada de MV

```
virsh start guest01-w10
```

Arrancaría el domain, MV, de nombre guest01-w10

```
virsh shutdown guest01-w10
```

Detendría, de forma ordenada, el domain indicado

Para ejecutar un apagado forzado

```
virsh destroy guest01-win10
```

Para ver más comandos en relación a la gestión de domains con virsh

```
virsh help domain
```

Algo de información sobre los dispositivos virtuales

Existen múltiples comandos virsh para esto, veamos algunos ejemplos

```
virsh domiflist guest01-w10
```

el comando anterior mostraría las interfaces de red usadas por el domain

```
virsh domblklist guest01-win10
```

mostraría los dispositivos de bloques asociados al domain

```
virsh domblkinfo guest01-win10 hdb
```

muestra la información del dispositivo hdb asociado al domain guest01-w10

NOTA: La denominación de un dispositivo depende del tipo de interfaz de conexión, por ejemplo los volúmenes conectados a través del emulador IDE tienen nombres de tipo hdX. Los volúmenes conectados por SATA tienen nombres tipos sdX

Para ver más información relativa a monitorización de un domain podemos ejecutar `virsh help monitor`

Editar la configuración de una MV

La definición de los domains KVM se realiza mediante archivos XML en donde se declara los recursos de hardware virtual, en este caso simulados mediante qemu (por ese motivo se habla por lo general de virtualización KVM/qemu), usados por la MV.

Para poder editar el archivo correspondiente usamos también el comando virsh

```
virsh edit guest01-w10
```

Abriría, con nuestro editor de texto favorito, el archivo de configuración correspondiente sobre el que podremos realizar los cambios oportunos. Una vez realizados los cambios podemos reiniciar la MV con

```
virsh reboot guest01-w10
```

Establecer autoarranque de MV

Es posible configurar un domain (MV) para que se inicie cuando arranque el sistema. Esto resulta muy útil en entornos de producción cuando se inicia un servidor. Para ello es suficiente con ejecutar el comando

```
virsh autostart guest01-w10
```

Al reiniciar la máquina veremos como inicia también el domain indicado. Para deshabilitarlo

```
virsh autostart --disable guest01-w10
```

Eliminación de una MV

En el lenguaje de virsh una máquina virtual es un domain. Para eliminar un domain podemos utilizar el comando

```
virsh undefine guest01-win7
```

Este comando elimina el domain guest01-win7

Sin embargo, antes de eliminar la MV es recomendable eliminar el disco virtual, para ello usaremos el comando

```
virsh vol-delete /var/lib/libvirt/images/guest01-w7.qcow2 --pool default
```

Acceso Remoto a MV

Podemos acceder a las MV a través de varios medios y protocolos

Acceso en modo gráfico con VNC

Para poder acceder a través del protocolo VNC a una MV podemos, o bien hacerlo desde virt-manager, en las propiedades de la MV, Monitor VNC, ahí podremos configurar el puerto y las interfaces de acceso al servidor, o bien, editando el archivo de configuración xml de la MV

```
virsh edit guest01-win10
```

Buscamos las líneas del elemento graphics y definimos sus parámetros, como por ejemplo

```
        <graphics type='vnc' port='5900' autoport='no' listen='0.0.0.0' keymap='es'>
              <listen type='address' address='0.0.0.0'/>
        </graphics>
```

En este caso podremos acceder desde cualquier interfaz de red, incluso desde la IP del anfitrión, a través del puerto 5900/tcp, mediante el protocolo VNC. El puerto puede modificarse e incluso ser asignado de forma automática, para poder acceder a distintas MV utilizando puertos diferentes. Podemos restringir el acceso a ciertas interfaces mediante el uso de la directiva listen anterior. Cuando esté habilitado el acceso mediante VNC en el mismo puerto 5900, podemos obtener el número de pantalla a partir del comando vncdisplay de virsh

```
virsh vncdisplay guest01-win10
```

Mostraría el número de pantalla: :0, :1, etc. que deberá incorporarse después de la dirección o hostname del servidor VNC en la URL de acceso VNC

Acceso en modo gráfico con Spice

Spice es un protocolo de acceso remotoen modo gráfico a las MV, consta de 3 elementos, un servidor Spice, corriendo en el hipervisor, un agente Spice, corriendo en cada domain o MV y un cliente Spice, el programa que usaremos para acceder en remoto.

Al igual que con VNC puede configurarse el modo de acceso desde virt-manager, en el apartado Monitor Spice en el que puede configurarse el acceso mediante Spice, o bien editando el archivo de configuración xml del domain, veamos un ejemplo

```
virsh edit guest01-win10
```

Buscamos de nuevo el elemento graphics y definimos una configuración como la siguiente

```
        <graphics type='spice' port='5900' autoport='no' listen='0.0.0.0' keymap='es'>
              <listen type='address' address='0.0.0.0'/>
        </graphics>
```

Este caso es análogo al anterior, con la salvedad de que ahora accedemos a través del protocolo Spice

Acceso en modo consola

Una opción utilizada para acceder a sistemas GNU/Linux en modo consola

```
virsh console guest01-debian1
```

Para poder acceder de este modo a una máquina GNU/Linux tenemos que añadir la consola a través de puerto serie en las opciones de grub. Para ello editamos el archivo /etc/default/grub y añadimos a la línea GRUB_CMDLINE_LINUX el texto console=ttyS0, de modo que la línea quedará

```
        GRUB_CMDLINE_LINUX=?console ttyS0?
```

Ahora actualizamos grub

```
update-grub
```

Y reiniciamos

```
reboot
```

Tras el reinicio podemos utilizar el comando anterior para acceder por consola a la MV

```
virsh console guest01-debian1
```

Clonación de MV

Una funcionalidad interesante es poder clonar MV existentes, de modo que obtengamos una MV igual a la original. Para ello usamos la herramienta virt-clone

```
virt-clone --original=guest01-win10 --name=guest01-win10-CLONE --file=/var/lib/libvirt/images/guest01-win10-CLONE.qcow2
```

El comando anterior clona el domain (MV) de nombre guest01-win10 en un nuevo domain de nombre guest01-win10-clone, además especifica que la ubicación de su volumen de almacenamiento tendrá la ruta indicada.

Gestión de volúmenes

Listado de volúmenes

```
virsh vol-list --details default
```

El comando anterior listará los volúmenes disponibles en el pool default. Un pool es una ubicación de almacenamiento de volúmenes. Es necesario indicar el pool para que el comando anterior funcione. Por defecto el volumen default, predefinido, está asociado al directorio /var/lib/libvirt/images. La opción --details muestra información detallada de los volúmenes. Mostraría una salida del tipo

```
 Name                Path                                      Type   Capacity  Allocation
 w10_cliente.qcow2   /var/lib/libvirt/images/w10_cliente.qcow2  file  20,00 GiB  234,95 MiB
 w2016.qcow2         /var/lib/libvirt/images/w2016.qcow2        file  30,00 GiB   10,93 GiB
 w2016_server.qcow2  /var/lib/libvirt/images/w2016_server.qcow2 file  30,00 GiB    1,11 GiB
 win10.qcow2         /var/lib/libvirt/images/win10.qcow2        file  20,00 GiB    9,10 GiB
```

Notar como se especifica por un lado la Capacidad del volumen, como se define a la hora de crear el volumen, y el espacio que está siendo realmente utilizado por los datos del volumen. Obtener información de un volumen

```
virsh vol-info /var/lib/libvirt/images/w2016_server_clone.qcow2
```

Mostraría la información del volumen indicado en el formato siguiente

```
Name:           w2016_server.qcow2
Type:           file
Capacity:       30,00 GiB
Allocation:     1,11 GiB
```

Para obtener más información sobre comandos relacionados con volúmenes

```
virsh help volume
```

Gestión de imágenes con qemu-img

A continuación vamos a ver un comando de qemu específicamente creado para gestionar las imágenes de almacenamiento asociadas a los volúmenes de datos de las MV

Comando qemu-img

Clonación enlazada

En el apartado anterior vimos como crear un clon de un domain. Este comando resulta útil cuando queremos realizar un clon ?completo?, es decir, una copia exacta y totalmente independiente del domain original.

Sin embargo, en ocasiones, es muy útil utilizar las ventajas del formato de volumen qcow2 (qemu copy on write), el cual permite, entre otras cosas, trabajar con volúmenes ?enlazados?; en el sentido de que un volumen enlazado a otro, que le sirve de base, solo almacenará las modificaciones efectuadas respecto al volumen de base. De este modo conseguimos ahorrar mucho espacio de almacenamiento al trabajar con domains que utilizan volúmenes que se van construyendo ?unos sobre otros?.

Por ejemplo, supongamos que tengo un volumen, que sirve de disco virtual al domain guest01-w10. Ese volumen se almacena en la ruta /var/lib/libvirt/images/guest01-w10.qcow2. Vamos a crear un clon de este domain, de nombre guest01-w10-lclone, que además, hará uso de un volumen que opera de modo enlazado al original, el volumen se almacenará en /var/lib/libvirt/images/guest01-w10-lclone.qcow2. Lo primero es crear un volumen ?enlazado? al original, es decir, no vamos a duplicar el volumen original, sino que vamos a crear un volumen que actúe tomando el volumen original ?base?, de este modo solo se almacenarán las diferencias, y no el sistema de archivos completo, en el nuevo volumen, para ello usaremos la herramienta qemu-img

```
qemu-img create -f qcow2 -b /var/lib/libvirt/images/guest01-w10.qcow2 /var/lib/libvirt/images/guest01-w10-lclone.qcow2
```

Para ver la información asociada al volumen recién creado

```
qemu-img info /var/lib/libvirt/images/guest01-w10-lclone.qcow2
```

Mostraría la información siguiente

```
image: /var/lib/libvirt/images/guest01-w10-lclone.qcow2
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 196K
cluster_size: 65536
backing file: /var/lib/libvirt/images/guest01-w10.qcow2
```

```
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

En la salida anterior podemos apreciar como el campo image indica la ruta del volumen recién creado, pero el campo backing file indica la ruta al archivo de imagen del volumen original. De este modo estamos trabajando con un volumen ?enlazado?, que tendrá las características indicadas previamente.

A continuación vamos a efectuar el clon mediante la herramienta virt-clone, pero especificando que tome el volumen que acabamos de crear con el comando anterior, sin modificar sus datos, con la opción --preserve-data

```
virt-clone --original=guest01-w10 --name=guest01-w10-lclone --preserve-data --file=/var/lib/libvirt/images/guest01-w10-lclone.qcow2
```

Ahora vamos a ver la lista de volúmenes y sus tamaños

```
virsh vol-list --details default
```

La salida mostraría

```
Name                Path                                      Type   Capacity  Allocation
-------------------------------------------------------------------------------------------
 w2016.qcow2              /var/lib/libvirt/images/w2016.qcow2         file  30,00 GiB   10,93 GiB
 w2016_server.qcow2       /var/lib/libvirt/images/w2016_server.qcow2  file  30,00 GiB    1,11 GiB
 guest01-w10.qcow2        /var/lib/libvirt/images/win10.qcow2         file  20,00 GiB    9,10 GiB
 guest01-w10_lclone.qcow2 /var/lib/libvirt/images/win10_clone.qcow2   file  20,00 GiB   57,13 MiB
```

Podemos ver como el tamaño del volumen del nuevo clon ocupa solamente 57,13 MB, sin embargo el volumen que le sirve de base ocupa 9,10 GB.

Gestión de networks

Introducción

El tratamiento de las network por parte de qemu/KVM se basa en el concepto de Virtual Network. Este concepto se basa un Switch Virtual, es decir un dispositivo software que actúa como un conmutador o switch.

Al mismo switch virtual se conectarán todas las MV, guests, asociadas a la misma virtual network. De este modo todas son visibles entre sí, porque además comparten el direccionamiento IP establecido en la virtual network. También es posible configurar un servicio DHCP para una virtual network, de modo que todas las MV conectadas a él puedan tener IPs asignadas de modo automático. Otra opción es la asignación de IP manual dentro del rango de direccionamiento asociado a la virtual network.

Por defecto se crea una Virtual Network de nombre ?default?, que opera en modo NAT, y que tiene asociado un dispositivo de tipo Switch Virtual de nombre ?virbr0?. Por supuesto son posibles muchas más configuraciones. Respecto al modo en que las MV se conectan al exterior, a través del comportamiento de reenvío del tráfico en la virtual network, encontramos los siguientes modos de operación.

Archivo de configuración de una network

Vamos a ver a modo de ejemplo el archivo de configuración de la network default, para ello volcamos el contenido con el comando

```
virsh net-dumpxml default
```

Lo cual arroja la salida

```
<network>
  <name>default</name>
  <uuid>3b6d6a45-a302-4641-974a-8379dbca58da</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:60:56:f1'/>
```

```
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

Como podemos observar, el nombre de la network es ?default?, opera en modo forward ?nat?, utiliza como dispositivo bridge, o switch virtual, virbr0, teniendo asignada la MAC 52:54:00:60:56:f1, y la dirección IP del propio switch virtual es 192.168.122.1, es decir la IP más baja dentro del rango de direccionamiento asociado a la network. Por último, vemos que tiene asociado un rango de cesiones DHCP asignables las MVs conectadas a esa network.

NAT Mode

Esta es el modo de red por defecto para los domains (MVs). De hecho se dispone por defecto de una network de este tipo con nombre default Para ver las networks definidas

```
virsh net-list --all
```

Mostraría las networks creadas. Sino se especifica el modificador --all solo se visualizan las network activas.

Este tipo de network permite construir un entorno de red virtual en el cual las MVs conectadas a la network son visibles entre sí, además de poder comunicarse desde y hacia el anfitrión, a través de la primera IP del rango de direcciones de la subred asignada.

Para ello utiliza una técnica de IP masquerade con la IP del adaptador físico del host. La limitación que tiene este modo de operación es que las MVs no son accesibles directamente desde fuera del host, aunque sí pueden comunicarse entre sí, con el anfitrión y salir hacia el exterior.

Veamos la información de la red NAT network default

```
virsh net-dumpxml default
```

```
<network>
  <name>default</name>
  <uuid>3b6d6a45-a302-4641-974a-8379dbca58da</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:60:56:f1'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

Vemos como se indica el mode=?nat?, los parámetros del port forwarding, el nombre del dispositivo bridge utilizado por la network, la dirección MAC del dispositivo bridge y el rango de direccionamiento IP asignable por DHCP.

Por defecto la network default viene deshabilitada, es decir hay que iniciarla explícitamente para que pueda ser utilizada. Para iniciarla:

```
virsh net-start default
```

Si queremos que la network se inicie de modo automático en el arranque ejecutamos `virsh net-autostart default`

Bridged Mode

En este modo los MVs (guests) son visibles en la misma LAN que el anfitrión, de modo que se visualizan transparentemente como hosts dentro de la misma LAN. No hay por tanto mecanismos NAT asociados, de modo que la configuración de la red se establece en el dominio de la LAN sobre la que se despliega el propio anfitrión.

Routed Mode

Este modo de operación es similar a NAT, con la salvedad de que las reglas de reenvío de tráfico son específicamente definidas en el switch virtual, el cual es susceptible de examinar el tráfico en tránsito y efectuar decisiones de enrutamiento. En esta modalidad el switch virtual actua, por lo tanto, como un enrutador, siendo necesario especificar todas las reglas de enrutamiento explícitamente.

Isolated Mode

Por último, el modo aislado solo permite conectar las MVs entre sí, pero sin ningún tipo de posibilidad de acceso al exterior a través del switch virtual. Es un modo adecuado si queremos un escenario de pruebas aislado en el que no influir.

Creación de una network a partir de otra

Podemos volcar en un archivo xml la información de definición de una network, modificar el archivo correspondiente y por último crear una nueva network a partir del archivo modificado. Veamos un ejemplo

```
virsh net-dumpxml default > red192.xml
```

Editamos el archivo red192.xml resultante para que quede

```
        <network>
          <name>red192</name>
          <uuid>bba275f0-8866-4580-ad86-eb42e94c07fd</uuid>
          <forward mode='nat'>
            <nat>
              <port start='1024' end='65535'/>
            </nat>
          </forward>
          <bridge name='virbr1' stp='on' delay='0'/>
          <mac address='52:54:00:2f:a3:d9'/>
          <domain name='red192'/>
          <ip address='192.168.0.1' netmask='255.255.255.0'>
            <dhcp>
              <range start='192.168.0.128' end='192.168.0.254'/>
            </dhcp>
          </ip>
        </network>
```

Y ahora creamos la nueva network a partir el archivo anterior

```
virsh net-create red192.xml
```

Gestión de snapshots

Las snapshots (instantáneas) permiten almacenar estados de ejecución de un domain. De este modo es posible restaurar un domain al estado de ejecución almacenado en la snapshot

Creación de snapshot con virsh

Para crear una snapshot podemos usar virsh

```
virsh snapshot-create-as guest01-win10 --name snap1
```

crearía la snapshot snap1 para el domain guest01-win10

Lista de snapshots de un domain

Para ver la lista de snapshots de un domain `virsh snapshot-list guest01-win10`

Restaurar un domain al estado de una snapshot

Debemos indicar el domain y el nombre de la snapshot que queremos restaurar

```
virsh snapshot-revert guest01-win10 snap1
```

restauraría el domain al estado almacenado en la snapshot snap1

Eliminar snapshot de domain

```
virsh snapshot-delete guest01-win10 snap1
```

eliminaría la snapshot creada anteriormente

Más acciones con snapshots

Podemos ver más comandos virsh relacionados con snapshots ejecutando

```
virsh help snapshot
```

Uso de drivers KVM paravirtualizados (virtio)

Los drivers paravirtualizados son drivers de dispositivos modificados para utilizar las características de rendimiento avanzado del hipervisor KVM. Cuando utilizamos drivers emulados el rendimiento es bastante menor, pues el hipervisor tiene que simular esos dispositivos. Al utilizar drivers paravirtualizados, mediante el estándar virtio, el propio driver coopera con el hipervisor, aprovechando las características de rendimiento avanzado.

Se dispone de drivers paravirtualizados para los dispositivos de bloques y tarjetas de red. En el caso de guests Linux éstos vienen integrados en KVM y son integrables en los kernel del guest.

Para guest windows podemos descargar los correspondientes binarios, a través de una .iso que será utilizada durante el proceso de instalación de la máquina virtual.

Activar virtio para dispositivos de bloques en guest Linux

Vamos a activar el uso de virtio para dispositivos de almacenamiento de bloques. Para ello usaremos el comando virsh edit usado para modificar la configuración de una MV

```
virsh edit guets01-debian1
```

Sustituimos en el elemento disk correspondiente el atributo bus dentro del elemento target, tal que así

```
<disk type='...' device='disk'>
  ...
  <target dev='vda' bus='virtio'/>
</disk>
```

Si todo va bien al iniciar el guest con virsh start no deberíamos obtener ningún error

Activar virtio para dispositivo de red en guest Linux

De un modo similar al procedimiento anterior activamos el bus virtio en el dispositivo correspondiente

```
virsh edit guets01-debian1
```

Sustituimos en el elemento interface correspondiente el atributo type dentro del elemento model, tal que así

```
<interface type='network'>
  ...
  <model type='virtio' />
</interface>
```

LABORATORIO DE MVs

A continuación vamos a desarrollar una práctica de construcción de un pequeño laboratorio de máquinas virtuales utilizando Qemu/KVM

Referencias

virt-install man page

https://linux.die.net/man/1/virt-install

Guía de Administración de KVM en RHEL

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-guest_virtual_machin

linux-kvm.org

https://www.linux-kvm.org/page/Documents

virtio

https://wiki.libvirt.org/page/Virtio

Volver

JavierFP 20:17 7 nov 2017 (CET)