KVM

KVM, Kernel-based Virtual Machine, is a **hypervisor** built into the Linux kernel. It is similar to **Xen** in purpose but much simpler to get running. Unlike native **QEMU**, which uses emulation, KVM is a special operating mode of QEMU that uses CPU extensions (**HVM**) for virtualization via a kernel module.

Related articles

Category:Hypervisor

Libvirt

Using KVM, one can run multiple virtual machines running unmodified GNU/Linux, Windows, or any other operating system. (See **Guest Support Status (http://www.linux-kvm.org/page/Guest_Support_Status)** for more information.) Each virtual machine has private virtualized hardware: a network card, disk, graphics card, etc.

Differences between KVM and **Xen**, **VMware**, or QEMU can be found at the **KVM FAQ (http://www.linux-kvm.org/page/FAQ#General_KVM_information)**.

This article does not cover features common to multiple emulators using KVM as a backend. You should see related articles for such information.

Contents

Checking support for KVM

Hardware support

KVM requires that the virtual machine host's processor has virtualization support (named VT-x for Intel processors and AMD-V for AMD processors). You can check whether your processor supports hardware virtualization with the following command:

```
$ LC_ALL=C lscpu | grep Virtualization
```

Alternatively:

```
$ grep -E --color=auto 'vmx|svm|0xc0f' /proc/cpuinfo
```

If nothing is displayed after running either command, then your processor does **not** support hardware virtualization, and you will **not** be able to use KVM.

Note: You may need to enable virtualization support in your BIOS.

Kernel support

Arch Linux kernels provide the appropriate **kernel modules** to support KVM and VIRTIO.

KVM modules

You can check if necessary modules (`kvm` and one of `kvm_amd` , `kvm_intel`) are available in your kernel with the following command (assuming your kernel is built with `CONFIG_IKCONFIG_PROC`):

```
$ zgrep CONFIG_KVM /proc/config.gz
```

The module is **only** available if it is set to either `y` or `m` .

Para-virtualized devices

Para-virtualization provides a fast and efficient means of communication for guests to use devices on the host machine. KVM provides para-virtualized devices to virtual machines using the Virtio API as a layer between the hypervisor and guest.

All virtio devices have two parts: the host device and the guest driver.

VIRTIO modules

Use the following command to check if needed modules are available:

```
$ zgrep VIRTIO /proc/config.gz
```

Loading kernel modules

First, check if the kernel modules are automatically loaded. This should be the case with recent versions of **udev**.

```
$ lsmod | grep kvm
$ lsmod | grep virtio
```

In case the above commands return nothing, you need to **Kernel modules#Manual module handling** kernel modules.

Tip: If modprobing `kvm_intel` or `kvm_amd` fails but modprobing `kvm` succeeds, (and `lscpu` claims that hardware acceleration is supported), check your BIOS settings. Some vendors (especially laptop vendors) disable these processor extensions by default. To determine whether there's no hardware support or there is but the extensions are disabled in BIOS, the output from `dmesg` after having failed to modprobe will tell.

List of para-virtualized devices

- network device (virtio-net)
- block device (virtio-blk)
- controller device (virtio-scsi)
- serial device (virtio-serial)
- balloon device (virtio-balloon)

How to use KVM

See the main article: **QEMU**.

Tips and tricks

Note: See **QEMU#Tips and tricks** and **QEMU#Troubleshooting** for general tips and tricks.

Nested virtualization

Nested virtualization enables existing virtual machines to be run on third-party hypervisors and on other clouds without any modifications to the original virtual machines or their networking.

On host, enable nested feature for `kvm_intel`:

```
# modprobe -r kvm_intel
# modprobe kvm_intel nested=1
```

To make it permanent (see **Kernel modules#Setting module options**):

```
/etc/modprobe.d/kvm_intel.conf
```

```
options kvm_intel nested=1
```

Verify that feature is activated:

```
$ systool -m kvm_intel -v | grep nested
```

```
    nested              = "Y"
```

Enable the "host passthrough" mode to forward all CPU features to the guest system:

1. If using **QEMU**, run the guest virtual machine with the following command: `qemu-system-x86_64 -enable-kvm -cpu host`.
2. If using *virt-manager*, change the CPU model to `host-passthrough` (it will not be in the list, just write it in the box).
3. If using *virsh*, use `virsh edit vm-name` and change the CPU line to `<cpu mode='host-passthrough' check='partial'/>`

Boot VM and check if vmx flag is present:

```
$ grep -E --color=auto 'vmx|svm' /proc/cpuinfo
```

Enabling huge pages

You may also want to enable hugepages to improve the performance of your virtual machine. With an up to date Arch Linux and a running KVM you probably already have everything you need. Check if you have the directory `/dev/hugepages`. If not, create it. Now we need the right permissions to use this directory. The default permission is root's uid and gid with 0755, but we want anyone in the kmv group to have access to hugepages.

Add to your `/etc/fstab`:

```
hugetlbfs       /dev/hugepages  hugetlbfs       mode=1770,gid=78        0 0
```

Of course the gid must match that of the `kvm` group. The mode of `1770` allows anyone in the group to create files but not unlink or rename each other's files. Make sure `/dev/hugepages` is mounted properly:

```
# umount /dev/hugepages
# mount /dev/hugepages
$ mount | grep huge
```

```
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,mode=1770,gid=78)
```

Now you can calculate how many hugepages you need. Check how large your hugepages are:

```
$ grep Hugepagesize /proc/meminfo
```

Normally that should be 2048 kB ≙ 2 MB. Let's say you want to run your virtual machine with 1024 MB. 1024 / 2 = 512. Add a few extra so we can round this up to 550. Now tell your machine how many hugepages you want:

```
# echo 550 > /proc/sys/vm/nr_hugepages
```

If you had enough free memory you should see:

```
$ grep HugePages_Total /proc/meminfo
```

```
HugesPages_Total:  550
```

If the number is smaller, close some applications or start your virtual machine with less memory (number_of_pages x 2):

```
$ qemu-system-x86_64 -enable-kvm -m 1024 -mem-path /dev/hugepages -hda <disk_image> [...]
```

Note the `-mem-path` parameter. This will make use of the hugepages.

Now you can check, while your virtual machine is running, how many pages are used:

```
$ grep HugePages /proc/meminfo
```

```
HugePages_Total:     550
HugePages_Free:       48
HugePages_Rsvd:        6
HugePages_Surp:        0
```

Now that everything seems to work you can enable hugepages by default if you like. Add to your `/etc/sysctl.d/40-hugepage.conf`:

```
vm.nr_hugepages = 550
```

See also:

- **summary of hugetlbpage support in the Linux kernel (https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt)**
- **Debian Wiki - Hugepages**

See also

- **KVM Howto (http://www.linux-kvm.org/page/HOWTO)**
- **KVM FAQ (http://www.linux-kvm.org/page/FAQ#General_KVM_information)**

Retrieved from "https://wiki.archlinux.org/index.php?title=KVM&oldid=508327"