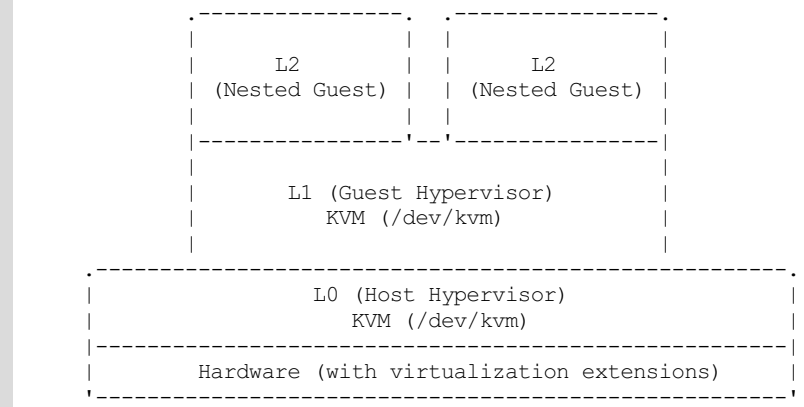
Running nested guests with KVM

A nested guest is the ability to run a guest inside another guest (it can be KVM-based or a different hypervisor). The straightforward example is a KVM guest that in turn runs on a KVM guest (the rest of this document is built on this example):



Terminology:

- L0 â€“ level-0; the bare metal host, running KVM
- L1 â€“ level-1 guest; a VM running on L0; also called the "guest hypervisor", as it itself is capable of running KVM.
- L2 â€“ level-2 guest; a VM running on L1, this is the "nested guest"

> **Note**
>
> The above diagram is modelled after the x86 architecture; s390x, ppc64 and other architectures are likely to have a different design for nesting.
>
> For example, s390x always has an LPAR (LogicalPARtition) hypervisor running on bare metal, adding another layer and resulting in at least four levels in a nested setup â€” L0 (bare metal, running the LPAR hypervisor), L1 (host hypervisor), L2 (guest hypervisor), L3 (nested guest).
>
> This document will stick with the three-level terminology (L0, L1, and L2) for all architectures; and will largely focus on x86.

Use Cases

There are several scenarios where nested KVM can be useful, to name a few:

- As a developer, you want to test your software on different operating systems (OSes). Instead of renting multiple VMs from a Cloud Provider, using nested KVM lets you rent a large enough "guest hypervisor" (level-1 guest). This in turn allows you to create multiple nested guests (level-2 guests), running different OSes, on which you can develop and test your software.
- Live migration of "guest hypervisors" and their nested guests, for load balancing, disaster recovery, etc.
- VM image creation tools (e.g. `virt-install`, etc) often run their own VM, and users expect these to work inside a VM.
- Some OSes use virtualization internally for security (e.g. to let applications run safely in isolation).

Enabling "nested" (x86)

From Linux kernel v4.20 onwards, the `nested` KVM parameter is enabled by default for Intel and AMD. (Though your Linux distribution might override this default.)

In case you are running a Linux kernel older than v4.19, to enable nesting, set the `nested` KVM module parameter to `Y` or `1`. To persist this setting across reboots, you can add it in a config file, as shown below:

1. On the bare metal host (L0), list the kernel modules and ensure that the KVM modules:

   ```
   $ lsmod | grep -i kvm
   kvm_intel             133627  0
   kvm                   435079  1 kvm_intel
   ```

2. Show information for `kvm_intel` module:

   ```
   $ modinfo kvm_intel | grep -i nested
   parm:           nested:bool
   ```

3. For the nested KVM configuration to persist across reboots, place the below in `/etc/modprobed/kvm_intel.conf` (create the file if it doesn't exist):

```
$ cat /etc/modprobe.d/kvm_intel.conf
options kvm-intel nested=y
```

4. Unload and re-load the KVM Intel module:

```
$ sudo rmmod kvm-intel
$ sudo modprobe kvm-intel
```

5. Verify if the `nested` parameter for KVM is enabled:

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

For AMD hosts, the process is the same as above, except that the module name is `kvm-amd`.

Additional nested-related kernel parameters (x86)

If your hardware is sufficiently advanced (Intel Haswell processor or higher, which has newer hardware virt extensions), the following additional features will also be enabled by default: "Shadow VMCS (Virtual Machine Control Structure)", APIC Virtualization on your bare metal host (L0). Parameters for Intel hosts:

```
$ cat /sys/module/kvm_intel/parameters/enable_shadow_vmcs
Y

$ cat /sys/module/kvm_intel/parameters/enable_apicv
Y

$ cat /sys/module/kvm_intel/parameters/ept
Y
```

> **Note**
>
> If you suspect your L2 (i.e. nested guest) is running slower, ensure the above are enabled (particularly `enable_shadow_vmcs` and `ept`).

Starting a nested guest (x86)

Once your bare metal host (L0) is configured for nesting, you should be able to start an L1 guest with:

```
$ qemu-kvm -cpu host [...]
```

The above will pass through the host CPU's capabilities as-is to the gues); or for better live migration compatibility, use a named CPU model supported by QEMU. e.g.:

```
$ qemu-kvm -cpu Haswell-noTSX-IBRS,vmx=on
```

then the guest hypervisor will subsequently be capable of running a nested guest with accelerated KVM.

Enabling "nested" (s390x)

1. On the host hypervisor (L0), enable the `nested` parameter on s390x:

```
$ rmmod kvm
$ modprobe kvm nested=1
```

> **Note**
>
> On s390x, the kernel parameter `hpage` is mutually exclusive with the `nested` paramter â€" i.e. to be able to enable `nested`, the `hpage` parameter *must* be disabled.

2. The guest hypervisor (L1) must be provided with the `sie` CPU feature â€" with QEMU, this can be done by using "host passthrough" (via the command-line `-cpu host`).
3. Now the KVM module can be loaded in the L1 (guest hypervisor):

```
$ modprobe kvm
```

Live migration with nested KVM

Migrating an L1 guest, with a *live* nested guest in it, to another bare metal host, works as of Linux kernel 5.3 and QEMU 4.2.0 for Intel x86 systems, and even on older versions for s390x.

On AMD systems, once an L1 guest has started an L2 guest, the L1 guest should no longer be migrated or saved (refer to QEMU documentation on "savevm"/"loadvm") until the L2 guest shuts down. Attempting to migrate or save-and-load an L1 guest while an

L2 guest is running will result in undefined behavior. You might see a `kernel BUG!` entry in `dmesg`, a kernel 'oops', or an outright kernel panic. Such a migrated or loaded L1 guest can no longer be considered stable or secure, and must be restarted. Migrating an L1 guest merely configured to support nesting, while not actually running L2 guests, is expected to function normally even on AMD systems but may fail once guests are started.

Migrating an L2 guest is always expected to succeed, so all the following scenarios should work even on AMD systems:

- Migrating a nested guest (L2) to another L1 guest on the *same* bare metal host.
- Migrating a nested guest (L2) to another L1 guest on a *different* bare metal host.
- Migrating a nested guest (L2) to a bare metal host.

Reporting bugs from nested setups

Debugging "nested" problems can involve sifting through log files across L0, L1 and L2; this can result in tedious back-n-forth between the bug reporter and the bug fixer.

- Mention that you are in a "nested" setup. If you are running any kind of "nesting" at all, say so. Unfortunately, this needs to be called out because when reporting bugs, people tend to forget to even *mention* that they're using nested virtualization.
- Ensure you are actually running KVM on KVM. Sometimes people do not have KVM enabled for their guest hypervisor (L1), which results in them running with pure emulation or what QEMU calls it as "TCG", but they think they're running nested KVM. Thus confusing "nested Virt" (which could also mean, QEMU on KVM) with "nested KVM" (KVM on KVM).

Information to collect (generic)

The following is not an exhaustive list, but a very good starting point:

- Kernel, libvirt, and QEMU version from L0
- Kernel, libvirt and QEMU version from L1
- QEMU command-line of L1 -- when using libvirt, you'll find it here: `/var/log/libvirt/qemu/instance.log`
- QEMU command-line of L2 -- as above, when using libvirt, get the complete libvirt-generated QEMU command-line
- `cat /sys/cpuinfo` from L0
- `cat /sys/cpuinfo` from L1
- `lscpu` from L0
- `lscpu` from L1
- Full `dmesg` output from L0
- Full `dmesg` output from L1

x86-specific info to collect

Both the below commands, `x86info` and `dmidecode`, should be available on most Linux distributions with the same name:

- Output of: `x86info -a` from L0
- Output of: `x86info -a` from L1
- Output of: `dmidecode` from L0
- Output of: `dmidecode` from L1

s390x-specific info to collect

Along with the earlier mentioned generic details, the below is also recommended:

- `/proc/sysinfo` from L1; this will also include the info from L0