13.2. USING SR-IOV

This section covers the use of PCI passthrough to assign a Virtual Function of an SR-IOV capable multiport network card to a virtual machine as a network device.

SR-IOV Virtual Functions (VFs) can be assigned to virtual machines by adding a device entry in *`<hostdev>`* with the `virsh edit` or `virsh attach-device` command. However, this can be problematic because unlike a regular network device, an SR-IOV VF network device does not have a permanent unique MAC address, and is assigned a new MAC address each time the host is rebooted. Because of this, even if the guest is assigned the same VF after a reboot, when the host is rebooted the guest determines its network adapter to have a new MAC address. As a result, the guest believes there is new hardware connected each time, and will usually require re-configuration of the guest's network settings.

libvirt-0.9.10 and later contains the *`<interface type='hostdev'>`* interface device. Using this interface device, **libvirt** will first perform any network-specific hardware/switch initialization indicated (such as setting the MAC address, VLAN tag, or 802.1Qbh virtualport parameters), then perform the PCI device assignment to the guest.

Using the *`<interface type='hostdev'>`* interface device requires:

- an SR-IOV-capable network card,

- host hardware that supports either the Intel VT-d or the AMD IOMMU extensions, and
- the PCI address of the VF to be assigned.

For a list of network interface cards (NICs) with SR-IOV support, see https://access.redhat.com/articles/1390483 .

Important

Assignment of an SR-IOV device to a virtual machine requires that the host hardware supports the Intel VT-d or the AMD IOMMU specification.

To attach an SR-IOV network device on an Intel or an AMD system, follow this procedure:

Procedure 13.1. Attach an SR-IOV network device on an Intel or AMD system

1. **Enable Intel VT-d or the AMD IOMMU specifications in the BIOS and kernel**

   On an Intel system, enable Intel VT-d in the BIOS if it is not enabled already. Refer to Procedure 12.1, “Preparing an Intel system for PCI device assignment” for procedural help on enabling Intel VT-d in the BIOS and kernel.

   Skip this step if Intel VT-d is already enabled and working.

   On an AMD system, enable the AMD IOMMU specifications in the BIOS if they are not enabled already. Refer to Procedure 12.2, “Preparing an AMD system for PCI device assignment” for procedural help on enabling IOMMU in the BIOS.
2. **Verify support**

   Verify if the PCI device with SR-IOV capabilities is detected. This example lists an Intel 82576 network interface card which supports SR-IOV. Use the `lspci` command to verify whether the device was detected.

```
ci
.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

Note that the output has been modified to remove all other devices.

3. **Start the SR-IOV kernel modules**

   If the device is supported the driver kernel module should be loaded automatically by the kernel. Optional parameters can be passed to the module using the `modprobe` command. The Intel 82576 network interface card uses the `igb` driver kernel module.

   ```
   # modprobe igb [<option>=<VAL1>,<VAL2>,]
   # lsmod |grep igb
   igb    87592  0
   dca    6708    1 igb
   ```

4. **Activate Virtual Functions**

   The *`max_vfs`* parameter of the `igb` module allocates the maximum number of Virtual Functions. The *`max_vfs`* parameter causes the driver to spawn, up to the value of the parameter in, Virtual Functions. For this particular card the valid range is *`0`* to *`7`*.

   Remove the module to change the variable.

   ```
   # modprobe -r igb
   ```

   Restart the module with the *`max_vfs`* set to *`7`* or any number of Virtual Functions up to the maximum supported by your device.

   ```
   # modprobe igb max_vfs=7
   ```

5. **Make the Virtual Functions persistent**

   Add the line `options igb max_vfs=7` to any file in `/etc/modprobe.d` to make the Virtual Functions persistent. For example:

   ```
   # echo "options igb max_vfs=7" >>/etc/modprobe.d/igb.conf
   ```

6. **Inspect the new Virtual Functions**

Using the `lspci` command, list the newly added Virtual Functions attached to the Intel 82576 network device. (Alternatively, use `grep` to search for `Virtual Function`, to search for devices that support Virtual Functions.)

```
# lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (re
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection(rev
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
```

The identifier for the PCI device is found with the *-n* parameter of the `lspci` command. The Physical Functions correspond to `0b:00.0` and `0b:00.1`. All Virtual Functions have `Virtual Function` in the description.

7. **Verify devices exist with virsh**

   The `libvirt` service must recognize the device before adding a device to a virtual machine. `libvirt` uses a similar notation to the `lspci` output. All punctuation characters, `;` and `.`, in `lspci` output are changed to underscores (`_`).

   Use the `virsh nodedev-list` command and the `grep` command to filter the Intel 82576 network device from the list of available host devices. *0b* is the filter for the Intel 82576 network devices in this example. This may vary for your system and may result in additional devices.

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

The serial numbers for the Virtual Functions and Physical Functions should be in the list.

8. **Get device details with virsh**

   The `pci_0000_0b_00_0` is one of the Physical Functions and `pci_0000_0b_10_0` is the first corresponding Virtual Function for that Physical Function. Use the `virsh nodedev-dumpxml` command to get advanced output for both devices.

```
# virsh nodedev-dumpxml pci_0000_0b_00_0
<device>
   <name>pci_0000_0b_00_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igb</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>0</slot>
      <function>0</function>
      <product id='0x10c9'>82576 Gigabit Network Connection</product>
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>
```

```
# virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
   <name>pci_0000_0b_10_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igbvf</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>16</slot>
      <function>0</function>
      <product id='0x10ca'>82576 Virtual Function</product>
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>
```

This example adds the Virtual Function `pci_0000_0b_10_0` to the virtual machine in Step 9. Note the `bus`, `slot` and `function` parameters of the Virtual Function: these are required for adding the device.

Copy these parameters into a temporary XML file, such as `/tmp/new-interface.xml` for example.

```
   <interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16' function='0'/>
     </source>
   </interface>
```

Note

If you do not specify a MAC address, one will be automatically generated. The *<virtualport>* element is only used when connecting to an 802.11Qbh hardware switch. The *<vlan>* element is new for Red Hat Enterprise Linux 6.4 and this will transparently put the guest's device on the VLAN tagged *42*.

When the virtual machine starts, it should see a network device of the type provided by the physical adapter, with the configured MAC address. This MAC address will remain unchanged across host and guest reboots.

The following *<interface>* example shows the syntax for the optional *<mac address>* , *<virtualport>* , and *<vlan>* elements. In practice, use either the *<vlan>* or *<virtualport>* element, not both simultaneously as shown in the example:

```
...
 <devices>
   ...
   <interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16' function='0'/
     </source>
     <mac address='52:54:00:6d:90:02'>
     <vlan>
        <tag id='42'/>
     </vlan>
     <virtualport type='802.1Qbh'>
       <parameters profileid='finance'/>
     </virtualport>
   </interface>
   ...
 </devices>
```

9. **Add the Virtual Function to the virtual machine**

   Add the Virtual Function to the virtual machine using the following command with the temporary file created in the previous step. This attaches the new device immediately and saves it for subsequent guest restarts.

   ```
   virsh attach-device MyGuest /tmp/new-interface.xml  --config
   ```

   Using the `--config` option ensures the new device is available after future guest restarts.

The virtual machine detects a new network interface card. This new card is the Virtual Function of the SR-IOV device.

Chapter 13. SR-IOV

13.3. Troubleshooting SR-IOV