NVIDIA DOCA Emulated Devices

User Guide

MLNX-15-060468 _v0.1 | April 2021

Table of Contents

Chapter 1. Mediated Devices

NVIDIA mediated devices deliver flexibility in allowing to create accelerated devices without SR-IOV on the BlueField® system. These mediated devices support NIC and RDMA and offer the same level of ASAP2 offloads as SR-IOV VFs. Mediates devices are supported using mlx5 sub-function acceleration technology.

Two sub-function devices are created on the BlueField device upon boot (one per port if the port is in switchdev mode) using commands from `/etc/mellanox/mlnx-sf.conf`:

```
/sbin/mlnx-sf -a create -d 0000:03:00.0 -u 61a59715-aeec-42d5-be83-f8f42ba8b049 --
mac 12:11:11:11:11:11
/sbin/mlnx-sf -a create -d 0000:03:00.1 -u 5b198182-1901-4c29-97a0-6623f3d02065 --
mac 12:11:11:11:11:12
```

The help menu for `mlnx-sf` is presented below:

```
Usage: mlnx-sf [ OPTIONS ]
OPTIONS:
-a, -action,    --action <action>               Perform action
    action:     { enable | create | configure | remove | show | set_max_mdevs |
 query_mdevs_num }

-d, -device,    --device <device>               PCI device
 <domain>:<bus>:<device>.<func> (E.g.: 0000:03:00.0)
-m, -max_mdevs, --max_mdevs <max mdevs number>  Set maximum number of MDEVs
-u, -uuid,      --uuid <uuid>                   UUID to create SF with
-M, -mac,       --mac <MAC>                     MAC to create SF with
-p, -permanent, --permanent [<conf file>]       Store configuration to be used after
 reboot and/or driver restart. Default (/etc/mellanox/mlnx-sf.conf).
-V, -version,   --version                       Display script version and exit
-D, -dryrun,    --dryrun                        Display commands only
-v, -verbose,   --verbose                       Run script in verbose mode (print
 out every step of execution)
-h, -help,      --help                          Display help

"/etc/mellanox/mlnx-sf.conf" can be updated manually or using "mlnx-sf" tool with "-
p" parameter.
```

1.1. Related Configuration

Interface names are configured using the UDEV rule under: `/etc/udev/rules.d/82-net-setup-link.rules`.

```
SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}!="", ATTR{phys_port_name}!="",
 \
        IMPORT{program}="/etc/infiniband/vf-net-link-name.sh $attr{phys_switch_id}
 $attr{phys_port_name}" \
        NAME="$env{NAME}", RUN+="/sbin/ethtool -L $env{NAME} combined 4"
# MDEV network interfaces
```

```
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/
pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:02.0/0000:03:00.0/61a59715-aeec-42d5-
be83-f8f42ba8b049/net/eth[0-9]", NAME="p0m0"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/
pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:02.0/0000:03:00.1/5b198182-1901-4c29-97a0-6623f3d0206
net/eth[0-9]", NAME="p1m0"
```

NVMe SNAP uses `p0m0` as its default interface. See `/etc/nvme_snap/sf1.conf`.

Chapter 2. VirtIO-net Emulated Devices

Note: This feature is supported at beta level. Please contact NVIDIA Support for enablement.

This feature enables users to create VirtIO-net emulated PCIe devices in the system where the NVIDIA® BlueField®-2 DPU is connected. This is done by the virtio-net-controller software module present in the DPU. Virtio-net emulated devices allow users to hot plug up to 127 virtio-net PCIe PF Ethernet NIC devices and virtio-net PCI VF Ethernet NIC devices in the host system where the DPU is plugged in.

DPU software also enables users to create virtio block PCI PF and SR-IOV PCI VF devices. This is covered in the *NVIDIA Mellanox NVMe SNAP and virtio-blk SNAP Documentation*.

2.1. VirtIO-net PF Devices

This section covers managing virtio-net PCI PF devices using virtio-net-controller.

2.1.1. VirtIO-net PF Device Configuration

1. Run the following command on the DPU:

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```

2. Cold reboot the host system.
3. Apply the following configuration on the DPU in three steps:

```
$ mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1

$ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
PCI_SWITCH_EMULATION_ENABLE=1 \
PCI_SWITCH_EMULATION_NUM_PORT=16 \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_VF=0 \
VIRTIO_NET_EMULATION_NUM_PF=0 \
VIRTIO_NET_EMULATION_NUM_MSIX=16 \
VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \
VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \
VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \
ECPF_ESWITCH_MANAGER=1 \
ECPF_PAGE_SUPPLIER=1 \
SRIOV_EN=0 \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64

$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s \
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
```

4. Cold reboot the host system a second time.

2.1.2. Creating Hotplug VirtIO-net PF Device

VirtIO emulated network PCIe devices are created and destroyed using virtio-net-controller application console. When this application is terminated, all created VirtIO-net emulated devices are hot unplugged.

1. Start virtio-net controller on the DPU. Run:
   ```
   $ virtio_net_controller -i mlx5_0
   ```
2. Create a hotplug virtio-net device. Run:
   ```
   $ create virtio-net dev 0x0 0C:C4:7A:FF:22:93 1500 3 1024
   ```
   This creates one hotplug virtio-net device with MAC address 0C:C4:7A:FF:22:93, MTU 1500, and 3 VirtIO queues with a depth of 1024 entries. This device is uniquely identified by its index. This index is used to query and update device attributes. This device is uniquely identified by its index. This index is used to query and update device attributes.
3. Set the link state up of the newly create device. Run:
   ```
   $ change virtio-net dev 0 link 1
   ```
4. Change the SF netdevice's queue parameters. The SF's netdevice is printed by the dump command of the virtio-net-controller. It is also printed during the create command. Run:
   ```
   $ ethtool -L <sf_netdev> combined 1
   $ ethtool -G <sf_netdev> rx 16
   $ ethtool -G <sf_netdev> tx 128
   ```
5. Bring up the representor port of the device. Run:
   ```
   $ ip link set dev pf0sf1 up
   $ ovs-vsctl add-port <bridge> pf0sf1
   ```
   Once steps 1-5 are completed, virtio-net device should be available in the host system.
6. To show all the device configurations of virtio-net device that you created, run:
   ```
   $ show virtio-net dev 0
   ```
7. To list all the virtio-net devices, run:
   ```
   $ list virtio-net devices
   ```
8. Once usage is complete, to hot-unplug a VirtIO net device, run:
   ```
   $ destroy virtio-net dev 0
   ```

Chapter 3. Virtio-net SR-IOV VF Devices

This section covers managing virtio-net PCI SR-IOV VF devices using virtio-net-controller.

3.1. Virtio-net SR-IOV VF Device Configuration

Note: Virtio-net SR-IOV VF is only supported with statically configured PF, hot-plugged PF is not currently supported.

1. On the x86 host, blacklist kernel modules virtio_pci and virtio_net by adding them to `/etc/modprobe.d/blacklist.conf`:
```
blacklist virtio_pci
blacklist virtio_net
```
2. On the x86 host, enable SR-IOV. Please refer to MLNX_OFED documentation under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) > Setting Up SR-IOV for instructions on how to do that. Make sure the parameters `intel_iommu=on iommu=pt pci=realloc` exist in grub.conf file.
3. Reboot x86 host, make sure those modules are not loaded.
```
# lsmod | grep -i virtio
```
   Note: Please make sure those modules are blacklisted before proceeding. If those modules are built into kernel image, they are not blacklistable. In that case, users must have a way to start the `virtio_net_controller` from Arm using SSH or serial/oob interfaces. This can be achieved by connecting the RShim/serial cable to another host, or setting up the OOB interface.
4. Run the following command on the DPU:
```
mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
```
5. Cold reboot the host system.
6. Apply the following configuration on the DPU in three steps.
   a).
   ```
   $ mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1
   ```
   b).
   ```
   $ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
   PCI_SWITCH_EMULATION_ENABLE=1 \
   ```

```
PCI_SWITCH_EMULATION_NUM_PORT=16 \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_VF=125 \
VIRTIO_NET_EMULATION_NUM_PF=4 \
VIRTIO_NET_EMULATION_NUM_MSIX=4 \
VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \
VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \
VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \
ECPF_ESWITCH_MANAGER=1 \
ECPF_PAGE_SUPPLIER=1 \
SRIOV_EN=1 \
PF_SF_BAR_SIZE=8 \
PF_TOTAL_SF=508
```

c).
```
$ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s PF_TOTAL_SF=170 PF_SF_BAR_SIZE=8
```

7. Cold reboot the host system.

3.2. Creating Virtio-net SR-IOV VF Devices

The virtio-net-controller application console must be kept alive to maintain the functionality of the static PF and its VFs.

1. Start virtio-net controller on the DPU. Run:

```
# virtio_net_controller -i mlx5_0
```

2. On the x86 host, make sure the static virtio network device presents.

```
# lspci | grep -i virtio
85:00.3 Network controller: Red Hat, Inc. Virtio network device
```

3. On x86 host, run:

```
# modprobe -v virtio_pci
# modprobe -v virtio_net
```

The net device should be created:

```
# ethtool -i p7p3
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:85:00.3
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

4. To create SR-IOV VF devices on the x86 host, run:

```
# echo 2 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

2 VFs should be created from x86 host:

```
# lspci | grep -i virt
85:00.3 Network controller: Red Hat, Inc. Virtio network device
85:04.5 Network controller: Red Hat, Inc. Virtio network device
85:04.6 Network controller: Red Hat, Inc. Virtio network device
```

5. From the DPU virtio-net controller, run:

```
virtio-net-controller> list virtio-net devices
PF  VF BDF      SF-Rep  Fw Open Stat Features     Queues  Msix MAC          MTU
0      85:0.3   pf0sf1  Y     Y  0x0f 0x2300470028 8       0    0cc47aff2292 1500
    0  85:4.5   pf0sf2  Y     Y  0x0f 0x300470028  8       0    c6237b326745 1500
    1  85:4.6   pf0sf3  Y     Y  0x0f 0x300470028  8       0    724833666998 1500
```

Bring up corresponding SF and add to OVS bridge:

```
# ip link set dev pf0sf2 up
# ovs-vsctl add-port <bridge> pf0sf2
```

Now the VF should be functional.