



DOCA SNAP Virtio-fs Service Guide

Table of contents

SNAP Virtio-fs Service Release Notes

61

This guide provides instructions on using the DOCA SNAP Virtio-fs service on top of the NVIDIA® BlueField®-3 DPU.

Introduction

Note

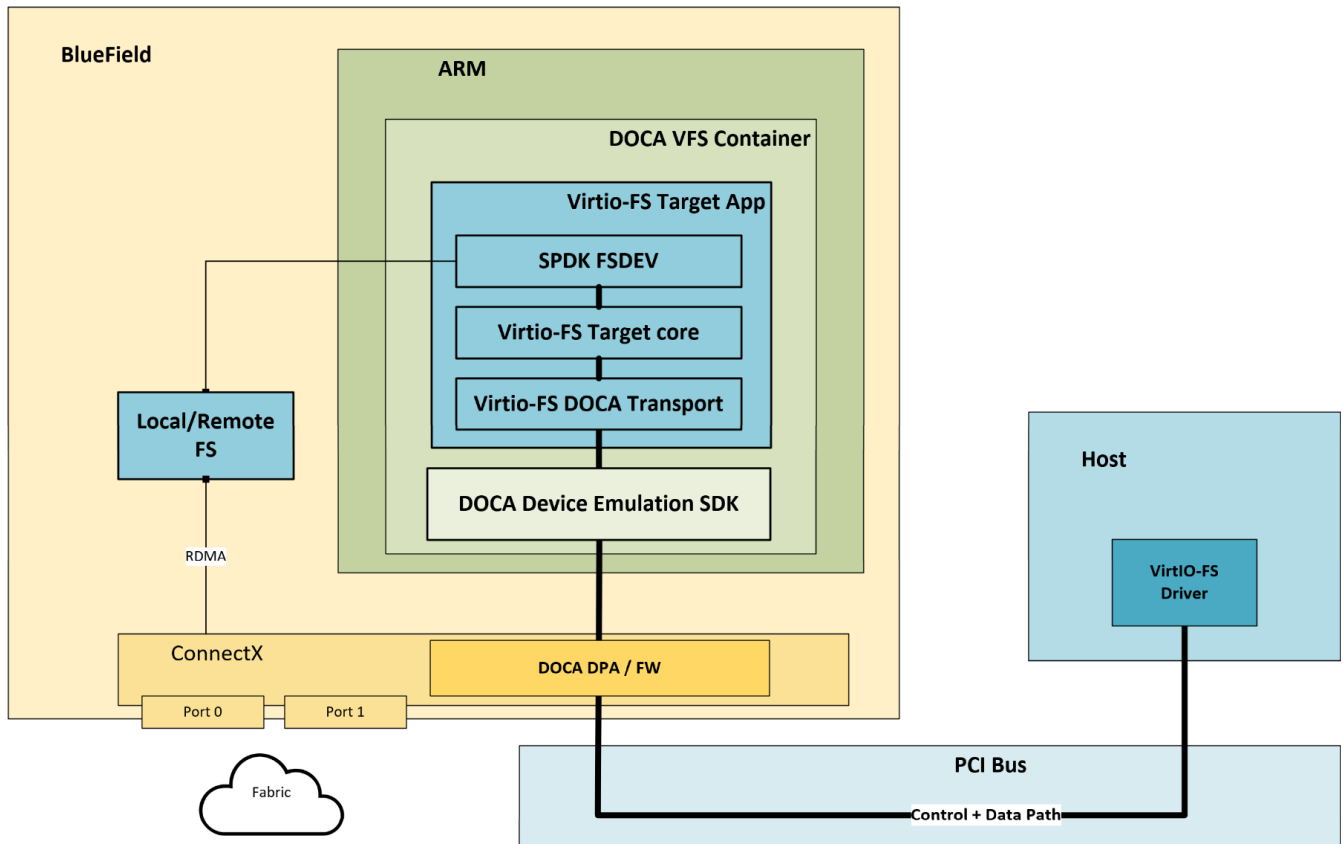
The DOCA SNAP Virtio-fs Service is currently supported at beta level.

NVIDIA® BlueField® enables hardware-accelerated software-defined virtio-fs PCIe device emulation. This leverages the power of BlueField networking platforms (DPUs or SuperNICs) to provide high-performance file system access in bare-metal and virtualized environments. Using BlueField, users can offload and accelerate networked file system operations from the host/guest, freeing up resources for other tasks and improving overall system efficiency. In this solution, the host/guest uses its own standard virtio-fs driver which is fully isolated from the networked filesystem mounted within the BlueField .

Built upon the DOCA and SPDK frameworks, virtio-fs device emulation on BlueField devices offers a comprehensive set of libraries for BlueField-based solutions and for storage solutions. This architecture consists of several key components:

- DOCA DevEmu subsystem and DOCA Virtio-fs library – These core libraries are responsible for the low-level hardware management and the translation of virtio descriptors carrying FUSE (filesystem in userspace) requests into abstract virtio-fs requests, which are then processed by the SPDK virtio-fs DOCA transport component.
- SPDK virtio-fs transport – This component is responsible for the interaction with the low-level DOCA components and translating the incoming abstract DOCA SNAP Virtio-fs requests into generic virtio-fs request which are then processed by the virtio-fs target core.
- SPDK virtio-fs target – This component implements and manages the virtio-fs device, transports, and the interface with a backend file system. Upon arrival on a new generic virtio-fs request from the transport, it processes and translates the requests according to virtio-fs and FUSE specifications, translating FUSE-based commands into the generic filesystem protocol.

- SPDK FSdev – This component provides generic filesystem abstraction and interfaces with the low-level filesystem modules implementing a specific backend filesystem protocol.



DOCA SNAP Virtio-fs as Container

The DOCA SNAP Virtio-fs container image may be downloaded from NVIDIA NGC and easily deployed on the BlueField using a YAML file. The YAML file points to the docker image that includes DOCA SNAP Virtio-fs binaries aligned with the latest `spdk.nvda` version.

DOCA SNAP Virtio-fs is not pre-installed on the BFB but can be downloaded manually on demand. For instructions on how to install the DOCA SNAP Virtio-fs container, refer to section "[DOCA SNAP Virtio-fs Container Deployment](#)".

DOCA SNAP Virtio-fs Deployment

This section describes how to deploy DOCA SNAP Virtio-fs as a container.

Note

DOCA SNAP Virtio-fs does not come pre-installed with the BFB bundle.

Note

The default virtio-blk emulation provider is set to `DPU`. In this mode, SNAP Virtio-fs and SNAP Virtio-blk can operate simultaneously only if they are assigned to different DPA execution units (EUs).

This separation is achieved by setting the environment variable `dpu_helper_core_mask=0x1fffe`, which is configured in the `set_environment_variables.sh` script included in the SNAP Virtio-fs package.

Installing Full DOCA Image on BlueField

To install the BFB on BlueField:

```
[host] sudo bfb-install --rshim <rshimN> --bfb <image_path.bfb>
```

For more information, please refer to section "Installing Full DOCA Image on DPU" in the [*NVIDIA DOCA Installation Guide for Linux*](#).

Firmware Installation

```
[dpu] sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --  
force-fw-update
```

For more information, please refer to section "Upgrading Firmware" in the [NVIDIA DOCA Installation Guide for Linux](#).

Firmware Configuration

Note

Firmware configuration may expose new emulated PCIe functions, which can be later used by the host's OS. As such, the user must make sure all exposed PCIe functions (static/hotplug) are backed by a supporting virtio-fs software configuration. Otherwise, these functions would malfunction and host behavior would be anomalous.

1. Clear the firmware config before implementing the required configuration:

```
[dpu] mst start  
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 reset
```

2. Verify the firmware configuration:

```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 query
```

Output example:

```

mlxconfig -d /dev/mst/mt41692_pciconf0 -e query | grep
VIRTIO_FS
Configurations:                                     Default
Current          Next Boot
*                VIRTIO_FS_EMULATION_ENABLE        False(0)
                 True(1)                True(1)
                 VIRTIO_FS_EMULATION_NUM_VF        0
0                0
*                VIRTIO_FS_EMULATION_NUM_PF        0
                 2                2
                 VIRTIO_FS_EMU_SUBSYSTEM_VENDOR_ID 6900
6900             6900
                 VIRTIO_FS_EMULATION_SUBSYSTEM_ID  4186
4186             4186
*                VIRTIO_FS_EMULATION_NUM_MSIX      2
3                3

```

The output provides 5 columns (listed from left to right):

- Non-default configuration marker (*)
- Firmware configuration name
- Default firmware value
- Current firmware value
- Firmware value after reboot – shows configuration update pending system reboot

3. To enable storage emulation options, BlueField must be set to work in internal CPU model:

```

[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s
INTERNAL_CPU_MODEL=1

```

4. To enable the firmware config with virtio-fs emulation PF:

```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s  
VIRTIO_FS_EMULATION_ENABLE=1 VIRTIO_FS_EMULATION_NUM_PF=1  
VIRTIO_FS_EMULATION_NUM_MSIX=3
```

(i) Note

For a complete list of the DOCA SNAP Virtio-fs firmware configuration options, refer to "[Appendix – BlueField Firmware Configuration](#)".

(i) Note

Power cycle is required to apply firmware configuration changes.

RDMA/RoCE Firmware Configuration

RoCE communication is blocked for the default interfaces of BlueField OS's (named ECPFs), `m1x5_0` and `m1x5_1` typically. If RoCE traffic is required, scalable functions (or SFs) must be added which are network functions which support RoCE transport.

To enable RDMA/RoCE:

```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s PER_PF_NUM_SF=1  
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s PF_SF_BAR_SIZE=8  
PF_TOTAL_SF=2
```



```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0.1 s PF_SF_BAR_SIZE=8
PF_TOTAL_SF=2
```

Note

This is not required when working over TCP or RDMA over InfiniBand.

Hot-plug Firmware Configuration

When PCIe switch emulation is enabled, BlueField can support

`PCI_SWITCH_EMULATION_NUM_PORT` -1 hotplug virtio-fs function. These PCIe functions are shared among all BlueField users and applications and may hold hot-plugged devices of type NVMe, virtio-blk, virtio-fs, and more (e.g., virtio-net).

To enable PCIe switch emulation and configure 31 hot-plugged ports to be used, run:

```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s
PCI_SWITCH_EMULATION_ENABLE=1 PCI_SWITCH_EMULATION_NUM_PORT=32
```

`PCI_SWITCH_EMULATION_NUM_PORT` equals 1 plus the number of hot-plugged PCIe functions.

Note

On AMD machines, hotplug is not guaranteed to work and enabling `PCI_SWITCH_EMULATION_ENABLE` may impact SR-IOV capabilities.

DOCA SNAP Virtio-fs Container Deployment

DOCA SNAP Virtio-fs container is available on the DOCA SNAP Virtio-fs NVIDIA™ [NGC page](#).

To deploy DOCA SNAP Virtio-fs container on top of BlueField, the following procedure is required:

1. Setup preparation and DOCA SNAP Virtio-fs resource download for container deployment. See section "[Preparation Steps](#)" for details.
2. Adjust the `doca_vfs.yaml` for advanced configuration if needed according to section "[Adjusting YAML Configuration](#)".
3. Deploy the container. The image is automatically pulled from NGC. See section "[Spawning DOCA SNAP Virtio-fs Container](#)" for details.

Preparation Steps

Step 1: Allocate Hugepages

Allocate `4GiB` hugepages for the DOCA SNAP Virtio-fs container according to the DPU OS's `Hugepagesize` value:

1. Query the `Hugepagesize` value:

```
[dpu] grep Hugepagesize /proc/meminfo
```

In Ubuntu, the value should be 2048KB.

2. Append the following line to the end of the `/etc/sysctl.conf` file:

```
vm.nr_hugepages = 2048
```

3. Run the following:

```
[dpu] sysctl --system
```

Note

If live upgrade is utilized in this deployment, it is necessary to allocate twice the amount of resources listed above for the upgraded container.

Warning

If other applications are running concurrently within the setup and are consuming hugepages, make sure to allocate a number of hugepages appropriate to accommodate all applications.

Step 2: Create /etc/virtiofs Folder

The folder `/etc/virtiofs` is used by the container for automatic configuration after deployment.

Note

The default YAML configuration only mounts the `/etc/virtiofs` folder for exposure and sharing between the container and the BlueField. This folder is used to expose configuration files or local file backends (e.g., AIO FSdev) from the DPU to the container.

Downloading YAML from NGC

The `.yaml` configuration file for the DOCA SNAP Virtio-fs container, `doca_vfs.yaml`, is uploaded to DOCA NGC.

Note

Internet connectivity is necessary to download DOCA SNAP Virtio-fs resources.

Adjusting YAML Configuration

The `.yaml` file can easily be edited for advanced configuration.

- The DOCA SNAP Virtio-fs `.yaml` file is configured by default to support Ubuntu setups (i.e., `Hugepagesize` = 2048 kB) by using `hugepages-2Mi`.

To support other setups, edit the `hugepages` section according to the relevant `Hugepagesize` value for the BlueField OS. For example, to support CentOS 8.x configure `Hugepagesize` to 512MB:

```
limits:  
  hugepages-512Mi: "<number-of-hugepages>Gi"
```

- The following example edits the `.yaml` file to request 8 CPU cores for the DOCA SNAP Virtio-fs container:

```
resources:
```

```
    cpu: "8"
  limits:
    cpu: "8"
  env:
    - name: APP_ARGS
      value: "-m 0xff"
```

(i) Note

If all BlueField-3 cores are requested, the user must verify no other containers are in conflict over CPU resources.

- To automatically configure the DOCA SNAP Virtio-fs container upon deployment:

1. Add the `spdk_rpc_init.conf` file under `/etc/virtiofs/`. File example:

```
fsdev_aio_create aio0 /etc/virtiofs/test
virtio_fs_transport_create -t DOCA
virtio_fs_transport_start -t DOCA
virtio_fs_device_create --transport-name DOCA --dev-name
vfsdev0 --tag docatag --fsdev aio0 --num-request-queues
1 --queue-size 32 --driver-platform x86_64
virtio_fs_doca_device_modify --dev-name vfsdev0 --
manager mlx5_0 --vuid "MT2251XZ02WZVFSS0D0F3"
virtio_fs_device_start --dev-name vfsdev0
```

2. Edit the `.yaml` file accordingly (uncomment):

```
env:
  - name: SPDK_RPC_INIT_CONF
```

```
value: "/etc/virtiofs/spdk_rpc_init.conf"
```

Note

It is user responsibility to make sure DOCA SNAP Virtio-fs configuration matches firmware configuration. That is, an emulated controller must be opened on all existing (static/hotplug) emulated PCIe functions (either through automatic or manual configuration). A PCIe function without a supporting controller is considered malfunctioned, and host behavior with it is anomalous.

Spawning DOCA SNAP Virtio-fs Container

Run the Kubernetes tool:

```
[dpu] systemctl restart containerd
[dpu] systemctl restart kubelet
[dpu] systemctl enable kubelet
[dpu] systemctl enable containerd
```

Copy the updated `doca_vfs.yaml` file to the `/etc/kubelet.d` directory.

Kubelet automatically pulls the container image from NGC described in the YAML file and spawns a pod executing the container.

```
cp doca_vfs.yaml /etc/kubelet.d/
```

The DOCA SNAP Virtio-fs Service starts initialization immediately, which may take a few seconds.

To verify whether DOCA SNAP Virtio-fs is running, send `spdk_rpc.py spdk_get_version` to confirm whether DOCA SNAP Virtio-fs is operational or still initializing.

Debug and Log

View currently active pods, and their IDs (it might take up to 20 seconds for the pod to start):

```
crictl pods
```

Example output:

POD ID	CREATED	STATE	NAME
0379ac2c4f34c	About a minute ago	Ready	virtiofs

View currently active containers, and their IDs:

```
crictl ps
```

View existing containers and their ID:

```
crictl ps -a
```

Examine the logs of a given container (virtio-fs logs):

```
crictl logs <container_id>
```

Examine the kubelet logs if something does not work as expected:

```
journalctl -u kubelet
```

The container log file is saved automatically by Kubelet under `/var/log/containers`.

Stop, Start, Restart DOCA SNAP Virtio-fs Container

- To stop the container, remove the `.yaml` file from `/etc/kubelet.d/`.
- To start the container, copy the `.yaml` file to the same path:

```
cp doca_vfs.yaml /etc/kubelet.d
```

- To restart the container (with sig-term), use the `-t` (timeout) option:

```
crictl stop -t 10 <container-id>
```

Note

After containers in a pod exit, the kubelet restarts them with an exponential back-off delay (10s, 20s, 40s, etc.) which is capped at five minutes. Once a container has run for 10 minutes without an issue, the kubelet resets the restart back-off timer for that container.

DOCA SNAP Virtio-fs with SNAP Support

The DOCA SNAP Virtio-fs container, along with its associated packages, natively supports [DOCA SNAP-4](#), which is implemented as an SPDK subsystem module. This design enables the concurrent operation of both **virtio-fs** and **virtio-blk** as a unified service. Additionally, DOCA SNAP is integrated as part of the DOCA SNAP Virtio-fs deployment.

DOCA SNAP deployment sets `snap`, `snap_nvme`, and `snap_vblk` as SPDK subsystems which can be disabled as needed.

Info

Refer to [DOCA SNAP-4 Service Guide](#) documentation for more information.

Note

DOCA SNAP RPCs can be used as an SPDK plugin, the recommended method for running RPCs with the SPDK RPC script. Users may need to set the `PYTHONPATH` environment variable to include the path to `snap_rpc.py`.

This command creates a Virtio block controller using the DOCA SNAP RPC plugin, specifying `--pf_id 0` and using `Null0` as the block device:

```
spdk_rpc.py --plugin snap_rpc
virtio_blk_controller_create --pf_id 0 --bdev
Null0
```

For further details on using RPC plugins, refer to the SPDK official documentation.

RPC Commands

Like other standard SPDK applications, the remote procedure call (RPC) protocol is used to control the DOCA SNAP Virtio-fs Service and supports JSON-based RPC protocol commands to control any resources and create, delete, query, or modify commands easily from the CLI.

DOCA SNAP Virtio-fs supports all standard SPDK RPC commands in addition to an extended DOCA SNAP Virtio-fs-specific command set. Standard SPDK commands are executed by the `spdk_rpc.py` tool.

To invoke the extended DOCA SNAP Virtio-fs-specific command set, users must add the `--plugin rpc_virtio_fs_tgt` flag to the SPDK's `rpc.py` command. The SPDK RPC plugin `rpc_virtio_fs_tgt.py` is implemented as an RPC plugin. This flag is not needed when working with containers.

The following is an example of an RPC when using DOCA SNAP Virtio-fs from the source:

```
/opt/nvidia/spdk-  
subsystem/src/spdk/install-$(hostname)/bin/spdk_rpc --plugin  
rpc_virtio_fs_tgt --help
```

Note

Users may need to define the path to the virtio-fs -target folder using the `PYTHONPATH` environment variable. More details on the RPC plugins can be found in [SPDK's official documentation](#).

Info

Full `spdk_rpc.py` command set documentation can be found in the [SPDK official documentation site](#).

DOCA SNAP Virtio-fs extended commands are detailed in the following subsections.

Using JSON-based RPC Protocol

The JSON-based RPC protocol can be used with the `rpc.py` script inside the DOCA SNAP Virtio-fs container and `crictrl` tool.

Info

The DOCA SNAP Virtio-fs container is CRI-compatible.

- To query the active container ID:

```
crictrl ps -s running -q --name virtiofs
```

- To post RPCs to the container using `crictrl`:

```
crictrl exec <container-id> spdk_rpc.py -v <RPC-method>
```

The flag `-v` controls verbosity. For example:

```
crictrl exec 0379ac2c4f34c spdk_rpc.py -v  
virtio_fs_doca_get_functions
```

Alternatively, an alias can be used:

```
cricctl exec -it $(cricctl ps -s running -q --name virtiofs)
spdk_rpc.py -v virtio_fs_doca_get_functions
```

- To open a bash shell to the container that can be used to post RPCs:

```
cricctl exec -it <container-id> bash
```

PCIe Function Management

Emulated PCIe functions are managed through DOCA devices called emulation managers. Emulation managers have special privileges to control, manipulate, and expose the emulated PCIe devices towards the host PCIe subsystem.

To operate a virtio-fs device/function by the DOCA transport, it is necessary to locate the appropriate emulation manager for it. The emulation manager maintains a list of the emulated PCIe functions it controls. Each of those functions is assigned a globally unique serial called a vendor unique identifier or VUID (e.g., MT2251XZ02WZVFSS0D0F2), which serves as unambiguous reference for identification and tracking purposes.

Command	Description
virtio_fs_doca_get_managers	List existing emulation managers for virtio-fs
virtio_fs_doca_get_functions	List functions for virtio-fs
virtio_fs_doca_get_possible_managers	List possible emulation managers for virtio-fs
virtio_fs_doca_manager_create	Create emulation manager for virtio-fs
virtio_fs_doca_manager_destroy	Destroy emulation manager for virtio-fs

virtio_fs_doca_get_managers

List existing emulation managers for virtio-fs . This method has no input parameters.

Example response:

```

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": [
    {
      "name": "mlx5_0"
    }
  ]
}

```

virtio_fs_doca_get_functions

List functions for virtio-fs with their characteristics. The user may specify no parameters to list all emulated virtio-fs functions managed by any emulation manager device, or specify an emulation manager device name to list virtio-fs functions managed by that emulation manager device.

Example response:

```

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": [
    {
      "manager": "mlx5_0",
      "Function List": [
        {
          "hot pluggable": "false",
          "pci_address": "0000:29:00.2",
          "vuid": "MT2251XZ02WZVFSS0D0F2",
          "function_type": "PF"
        }
      ]
    }
  ]
}

```

```
]
}
```

Parameter Name	Optional/Mandatory	Type	Description
manager	Optional	String	Emulation manager device name to list emulated virtio-fs functions specific to it

virtio_fs_doca_get_possible_managers

List possible emulation managers for virtio-fs. This method has no input parameters.

Example response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": [
    {
      "name": "mlx5_0",
    }
  ]
}
```

virtio_fs_doca_manager_create

Create a virtio-fs emulation manager.

Parameter Name	Optional/Mandatory	Type	Description
manager	Mandatory	String	Emulation manager device name

virtio_fs_doca_manager_destroy

Destroy a virtio-fs emulation manager.

Parameter Name	Optional/Mandatory	Type	Description
<code>manager</code>	Mandatory	String	Emulation manager device name

Hot-pluggable PCIe Functions Management

Hotplug PCIe functions are configured dynamically at runtime using RPCs.

The commands outlined in the following subsections hot plug a new PCIe function to the system.

virtio_fs_doca_get_functions

List DOCA transport functions for virtio-fs with their characteristics.

Users may specify no parameters to list all emulated virtio-fs functions managed by any emulation manager device, or an emulation manager device name to list virtio-fs functions managed by a specific emulation manager device.

Parameter Name	Optional/Mandatory	Type	Description
<code>manager</code>	Mandatory	String	Emulation manager device name for creating a new Virtio FS function

virtio_fs_doca_function_create

Create a DOCA virtio FS function. The return value of this method is a VUID. This is not needed for static functions as the VUID can be retrieved from

`virtio_fs_doca_get_functions`.

Parameter Name	Optional/Mandatory	Type	Description
<code>manager</code>	Mandatory	String	Emulation manager device name for creating a new virtio-fs function

virtio_fs_doca_function_destroy

Destroy a DOCA SNAP Virtio-fs function.

Note

This function should not be associated to any virtio-fs device.

Parameter Name	Optional/Mandatory	Type	Description
<code>manager</code>	Mandatory	String	Emulation manager device name for destroying a virtio-fs function
<code>vuid</code>	Mandatory	String	VUID of the function to destroy

virtio_fs_doca_device_hotplug

Hot plug a DOCA SNAP Virtio-fs device. The virtio-fs device must be started.

Parameter Name	Optional/Mandatory	Type	Description
<code>dev_name</code>	Mandatory	String	Virtio-fs device name to hot plug

Parameter Name	Optional/Mandatory	Type	Description
wait-for-done	Optional	Flag	If used, the method waits until the device is visible by the host PCIe subsystem. Otherwise, only issue hot-plug operation and exit.

virtio_fs_doca_device_hotunplug

Hot unplug a DOCA virtio FS device. The virtio FS device must be started.

Parameter Name	Optional/Mandatory	Type	Description
dev_name	Mandatory	String	Virtio-fs device name to hot unplug
wait-for-done	Optional	Flag	If exists, the method waits until the device is non-visible by the host PCIe subsystem. Otherwise, only issue hot-unplug operation and exit.

SPDK FSdev Module Configuration

fsdev_set_opts

Set SPDK FSdev module options.

Parameter Name	Optional/Mandatory	Type	Description
fsdev_io_pool_size	Mandatory	int	SPDK FSdev IO objects pool size
fsdev_io_cache_size	Mandatory	int	SPDK FSdev IO per-thread objects cache size

fsdev_get_opts

Get SPDK FSdev module options.

SPDK FSDEV Management

DOCA SNAP Virtio-fs uses the SPDK file system (FSdev) device framework as a backend for its virtio-fs controllers. Therefore, an SPDK FSdev must be created and configured in advance.

Although the SPDK FSdev framework is generic and allows different types of the backend file system devices to be implemented. Currently, the only available backend device is AIO. This is the file system device that provides passthrough access to a local folder using either the [Linux-native async I/O](#) or [POSIX async I/O](#).

fsdev_get_fsdevs

Get information about the SPDK filesystem devices (fsdevs). The user may specify no parameters to list all filesystem devices, or a filesystem device may be specified by name.

Parameter Name	Optional/Mandatory	Type	Description
<code>name</code>	Optional	string	Name of the fsdev of interest

fsdev_aio_create

Create an SPDK AIO FSdev ,

Parameter Name	Optional/Mandatory	Type	Description
<code>name</code>	Mandatory	string	Name of the AIO FSdev to create
<code>root_path</code>	Mandatory	string	Path on the system directory to be exposed as an SPDK filesystem
<code>enable_xattr</code>	Optional	boolean	Enable extended attributes if set to <code>true</code> ; <code>false</code> by default

Parameter Name	Optional/Mandatory	Type	Description
<code>enable_writeback_cache</code>	Optional	bool	Enable the writeback cache if set to <code>true</code> ; <code>false</code> by default
<code>max_write</code>	Optional	int	Maximum write size in bytes; <code>0x00020000</code> by default
<code>enable_skip_rw</code>	Optional	bool	Enable skipping read/write IOs if set to <code>true</code> ; <code>false</code> by default <div style="background-color: #ffffcc; padding: 5px;"> <p>Note For debug purposes only.</p> </div>

fsdev_aio_delete

Delete an AIO FSdev .

Parameter Name	Optional/Mandatory	Type	Description
<code>name</code>	Mandatory	string	Name of the AIO FSdev to delete

Virtio-fs Emulation Management

Virtio-fs emulation is a protocol belonging to the virtio family of devices. These mount points are found in virtual environments yet by design look like physical mount points to the user within the virtual machine. Each virtio-fs mount point (e.g., virtio-fs PCIe entry) exposed to the host, whether it is PF or VF, must be backed by a virtio-fs controller.

Note

Probing a virtio-fs driver on the host without an already functioning virtio-fs controller may cause the host to hang until such controller is opened successfully (no timeout mechanism exists).

Command	Description
virtio_fs_transport_create	Create a virtio-fs transport
virtio_fs_transport_destroy	Destroy a virtio-fs transport
virtio_fs_transport_start	Start a virtio-fs transport
virtio_fs_transport_stop	Stop a virtio-fs transport
virtio_fs_get_transports	Display virtio-fs transports or requested transport
virtio_fs_device_create	Create a virtio-fs device
virtio_fs_device_start	Start a virtio-fs device
virtio_fs_device_stop	Stop a virtio-fs device
virtio_fs_device_destroy	Destroy a virtio-fs device
virtio_fs_get_devices	Display virtio-fs devices with their characteristics
virtio_fs_doca_device_modify	Modify a virtio-fs device created from DOCA transport

virtio_fs_transport_create

Create a virtio-fs transport. This RPC includes all the common parameters/options for all transports. The transport becomes operational once it is started.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.

virtio_fs_transport_destroy

Destroy a virtio-fs transport.

Note

The transport must be stopped for destruction.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.

virtio_fs_transport_start

Start a virtio-fs transport. This RPC finalizes the transport configuration. From this point, the transport is fully operational and can be used to create new devices.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.

virtio_fs_transport_stop

Stop a virtio-fs transport. This RPC makes the transport configurable again.

Note

A transport cannot be stopped if any devices are associated to it.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.

virtio_fs_get_transports

Display virtio-fs transports or requested transport.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Optional	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.

virtio_fs_device_create

Create a virtio-fs device. This RPC creates a device with common parameters which are acceptable to all the transport types. To configure transport-specific parameters, users should use the `virtio_fs_doca_device_modify` command. The device becomes operational once it is started.

Parameter Name	Optional/Mandatory	Type	Description
<code>transport_name</code>	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, <code>transport_name</code> should be DOCA.
<code>dev_name</code>	Mandatory	String	Virtio-fs device name to use
<code>tag</code>	Optional	String	Virtio-fs tag according to the virtio specification.

Note

Parameter Name	Optional/Mandatory	Type	Description
			Must be provided during the <code>virtio_fs_device_create</code> RPC before the <code>virtio_fs_device_start</code> RPC.
<code>num_request_queues</code>	Optional	Number	Virtio-fs <code>num_request_queues</code> according to the virtio specification (default 31, range 1-254)
<code>queue_size</code>	Optional	Number	The maximal queue size for all virtio queues (default 64, range 8-256)
<code>fsdev</code>	Optional	String	<p>The name of the SPDK filesystem backend device</p> <p>(i) Note Must be provided during the <code>virtio_fs_device_create</code> RPC before the <code>virtio_fs_device_start</code> RPC.</p> <p>(i) Note RPC does not verify if FSdev is valid. If a wrong FSdev is attached to the device, the user would experience failure during mount of the FS on the host.</p>
<code>packed_vq</code>	Optional	Boolean	Expose packed virtqueues feature to the driver for negotiation.

Parameter Name	Optional/Mandatory	Type	Description
<code>drive_r_platform</code>	Optional	String	Set the driver's platform architecture. Possible values: <code>native</code> ; <code>x86</code> ; <code>x86_64</code> ; <code>aarch32</code> ; <code>aarch64</code> . Using the <code>native</code> platform option sets the driver platform to be identical to the device platform.
<code>notification_vq</code>	Optional	Boolean	Expose notification virtqueue feature to the driver for negotiation

virtio_fs_device_start

Start a virtio-fs device. This RPC finalizes the device configuration. From this point, the transport is fully operational.

Parameter Name	Optional/Mandatory	Type	Description
<code>dev_name</code>	Mandatory	String	Virtio-fs device name

virtio_fs_device_stop

Stop a virtio-fs device.

Parameter Name	Optional/Mandatory	Type	Description
<code>dev_name</code>	Mandatory	String	Virtio-fs device name

Note

The RPCs `virtio_fs_device_stop` and `virtio_fs_device_start` are supported during traffic. The user can stop the device while traffic is ongoing using `virtio_fs_device_stop`, then restart it using

`virtio_fs_device_start`, and the device would continue handling I/O without any errors.

virtio_fs_device_destroy

Destroy a virtio-fs device.

Note


The device must be stopped before destruction.

Parameter Name	Optional/Mandatory	Type	Description
<code>dev_name</code>	Mandatory	String	Virtio-fs device name

virtio_fs_device_modify

Modify a virtio-fs device. This RPC is used to modify/set common properties of the device which are acceptable to all the transports.

Parameter Name	Optional/Mandatory	Type	Description
<code>dev_name</code>	Mandatory	String	Virtio-fs device name to use
<code>tag</code>	Optional	String	Virtio-fs tag according to the virtio specification

 **Note**
Must be provided during `virtio_fs_device_create` or

Parameter Name	Optional/Mandatory	Type	Description
			<code>virtio_fs_device_modify</code> RPCs, before <code>virtio_fs_device_start</code> RPC.
<code>num_request_queues</code>	Optional	Number	Virtio-fs <code>num_request_queues</code> according to the virtio specification (default 31; range 1-254)
<code>queue_size</code>	Optional	Number	The maximal queue size for all virtio queues (default 64; range 8-256)
<code>fsdev</code>	Optional	String	<p>The name of the SPDK filesystem backend device</p> <p>(i) Note Must be provided during <code>virtio_fs_device_create</code> or <code>virtio_fs_device_modify</code> RPCs, before <code>virtio_fs_device_start</code> RPC.</p> <p>(i) Note RPC does not verify if FSdev is valid. If a wrong FSdev is attached to the device, the user would experience failure during mount of the FS on the host.</p>
<code>packed_vq</code>	Optional	Boolean	Expose packed virtqueues feature to the driver for negotiation

Parameter Name	Optional/Mandatory	Type	Description
drive_r_platform	Optional	String	Set the driver's platform architecture. Possible values: <code>native</code> ; <code>x86</code> ; <code>x86_64</code> ; <code>aarch32</code> ; <code>aarch64</code> . Using the <code>native</code> platform option sets the driver platform to be identical to the device platform.

virtio_fs_get_devices

Display virtio-fs devices with their characteristics.

- The user may specify no parameters to list the virtio-fs devices associated with all transports
- The user may specify the name of a transport to list the virtio-fs devices associated with it
- The user may specify the name of a virtio-fs device to display its characteristics

Transport name and device name parameters should be mutually exclusive.

Example response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": [
    {
      "name": "vfsdev0",
      "transport_name": "DOCA",
      "state": "idle",
      "fsdev": "aio0",
      "tag": "docatag",
      "queue_size": 256,
      "num_request_queues": 1,
      "packed_ring": true
    }
  ]
}
```

```

    }
  ]
}

```

Parameter Name	Optional/Mandatory	Type	Description
transport_name	Optional	String	Name of transport whose associated virtio-fs devices to list
dev_name	Optional	String	Virtio-fs device name


virtio_fs_doca_device_modify

Modify a virtio-fs device created from DOCA transport.

Info

This RPC is for configuring DOCA target specific parameters.

Parameter Name	Optional/Mandatory	Type	Description
dev_name	Mandatory	String	Virtio-fs device name
manager	Optional (must be provided before start)	String	Emulation manager
vuid	Optional (must be provided before start)	String	Vendor unique identifier

 **Note**

Parameter Name	Optional/Mandatory	Type	Description
			VUID validation is not done. If an invalid VUID is set, <code>virtio_fs_device_start</code> RPC fails.

Configuration Example

Static Function – Bring up

The following is an example of creating virtio-fs DOCA transport and associating it to a virtio-fs device using a static physical function.

- In BlueField:

1. Create an AIO FSdev backend:

```
rpc.py fsdev_aio_create aio0 /etc/virtiofs
```

2. List possible emulation managers:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_possible_managers
```

3. Create DOCA transport, emulation manager and start transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_create -t DOCA
```

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_manager_create -m mlx5_0  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_start -t DOCA
```

4. Get transport information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_transports
```

5. Get managers information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_managers
```

6. Get function information, including their VUIDs:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_functions
```

7. Create the virtio-fs device associated with DOCA transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_create --transport-name DOCA --dev-name  
vfsdev0 --tag doca_test --fsdev aio0 --num-request-  
queues 8 --queue-size 256 --driver-platform x86_64
```

8. Set and modify virtio-fs parameters (VUID must be provided before calling `virtio_fs_device_start` RPC):

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_device_modify --dev-name vfsdev0 --  
manager mlx5_0 --vuid MT2333XZ0VJQVFSS0D0F2
```

9. Start the virtio-fs device:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_start --dev-name vfsdev0
```

10. Get device information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_devices
```

- In VM/host:
 - To mount a device with the tag `docatag` and load `virtio_pci` driver if not loaded:

```
mkdir "/tmp/test"  
modprobe -v virtioFS  
mount -t virtiofs docatag /tmp/test
```

Static Function – Teardown

- In BlueField:
 1. Get device information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_devices
```

2. Stop and destroy the virtio-fs device:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_stop --dev-name vfsdev0  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_destroy --dev-name vfsdev0
```

3. Stop the DOCA transport and destroy emulation manager and transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_stop -t DOCA  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_manager_destroy -m mlx5_0  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_destroy -t DOCA
```

- In VM/host:
 - To unmount the device:

```
umount /tmp/test  
modprobe -rv virtiofs
```

Hotplug Function

The following is an example of creating virtio-fs DOCA transport, creating a virtio-fs function, associating it to a virtio-fs device, and hot-plugging it:

- In BlueField:

1. Create AIO FSdev backend:

```
rpc.py fsdev_aio_create aio0 /etc/virtiofs
```

2. List possible emulation managers:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_possible_managers
```

3. Create the DOCA transport and emulation manager and start the transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_create -t DOCA  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_manager_create -m mlx5_0  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_transport_start -t DOCA
```

4. Get transport information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_transports
```

5. Get managers information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_managers
```

Some managers would show hotplug capability.

6. Get functions information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_functions
```

7. Create virtio-fs function:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_function_create --manager mlx5_0
```

Returns VUID `MT2333XZ0VJQVFSS0D0F2`.

8. Get functions information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_get_functions
```

Returns the function that has been created with the appropriate VUID.

9. Create the virtio-fs device associated with DOCA transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_create --transport-name DOCA --dev-name
```

```
vfsdev0 --tag doca_test --fsdev aio0 --num-request-  
queues 8 --queue-size 256 --driver-platform x86_64
```

10. Set and modify virtio-fs parameters:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_create --transport-name DOCA --dev-name  
vfsdev0 --tag doca_test --fsdev aio0 --num-request-  
queues 8 --queue-size 256 --driver-platform x86_64
```

The VUID must be provided before calling the `virtio_fs_device_start` RPC.

11. Start the virtio FS device:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_start --dev-name vfsdev0
```

12. Get device information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_devices
```

The output for `vfsdev0` would show it is not yet plugged.

13. Hot plug the DOCA device to the host and wait until it becomes visible by the host:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_device_hotplug --dev-name vfsdev0 --wait-
```

```
for-done
```

- In VM/host:
 - To mount a device with the tag `docatag` and load `virtio_pci` driver if not loaded:

```
mkdir "/tmp/test"  
modprobe -v virtiofs  
mount -t virtiofs docatag /tmp/test
```

Hot-unplug Function

The following is an example of cleaning up and destroying the flow described under section "[Hotplug Function](#)":

- In VM/host:
 - To unmount the device:

```
umount /tmp/test  
modprobe -rv virtiofs
```

- In BlueField:
 1. Get device information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_devices
```

- Hot unplug the DOCA device from the host and wait until it becomes non-visible by the host:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_device_hotunplug --dev-name vfsdev0 --  
wait-for-done
```

- Get device information:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_get_devices
```

- Stop and destroy the virtio-fs DOCA device:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_stop --dev-name vfsdev0  
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_device_destroy --dev-name vfsdev0
```

- Destroy the virtio-fs function:

```
rpc.py --plugin rpc_virtio_fs_tgt -v  
virtio_fs_doca_function_destroy --manager mlx5_0 --vuid  
MT2333XZ0VJQVFSS0D0F2
```

Stop the DOCA transport and destroy the emulation manager and transport:

```
rpc.py --plugin rpc_virtio_fs_tgt -v virtio_fs_transport_stop -t  
DOCA
```

```
rpc.py --plugin rpc_virtio_fs_tgt -v
virtio_fs_doca_manager_destroy -m mlx5_0
rpc.py --plugin rpc_virtio_fs_tgt -v virtio_fs_transport_destroy
-t DOCA
```

Saving and Loading Configuration

To recover and reuse configurations for virtio-fs, you can save the desired configuration and load it later using a saved configuration file. This process allows for quick recovery and ensures consistent device/application setup.

To save the configuration of virtio-fs devices:

1. Use RPCs to configure your virtio-fs device/application as needed.
2. Save the configuration after each configuration cycle using the `save_config` RPC. This creates a JSON configuration file for future use:

```
spdk_rpc.py save_config -o <json_config>.json
```

To initiate a configuration cycle, create a new configuration file using the same format as `spdk_rpc_init.conf` files:

```
cat <new_config>.conf | spdk_rpc.py
```

Info

The last line of the `<new_config>.conf` file must include a call to save the RPC configuration. Example:

```
--plugin rpc_virtio_fs_tgt -v fsdev_aio_create
aio0 /tmp/fsdev-test
--plugin rpc_virtio_fs_tgt -v
virtio_fs_transport_create -t DOCA
--plugin rpc_virtio_fs_tgt -v
virtio_fs_transport_start -t DOCA
--plugin rpc_virtio_fs_tgt -v
virtio_fs_device_create --transport-name DOCA --
dev-name vfsdev0 --tag docatest --fsdev aio0 --
num-request-queues 1 --queue-size 256 --driver-
platform x86_64
--plugin rpc_virtio_fs_tgt -v
virtio_fs_doca_device_modify --dev-name vfsdev0 -
-manager mlx5_0 --vuid MT2306XZ00HKVFSS0D0F4
--plugin rpc_virtio_fs_tgt -v
virtio_fs_device_start --dev-name vfsdev0
save_config -o <json_config>.json
```

Tip

Saving configuration at the end of the script ensures the configuration is saved only after all the script changes are successfully applied.

To load a saved configuration for virtio-fs, use the following methods based on your environment:

- When using virtio-fs from a container, set the following in the YAML configuration file:

```
SPDK_RPC_INIT_CONF_JSON: <json_config>.json
APP_ARGS: --json-ignore-init-errors
```

- When using virtio-fs from source, re-run the DOCA SNAP Virtio-fs application, recovering the configuration from the saved file:

```
/opt/nvidia/spdk-subsystem/src/virtio-fs-
target/app/virtio_fs_tgt -c <json_config>.json --json-ignore-
init-errors
```

Warning

When modifying controller/function configurations, ensure that the host driver remains unloaded until the configuration change is fully applied. Otherwise, configuration recovery may fail if changes are incomplete or interrupted.

Disabling/enabling SPDK Subsystem

Virtio-fs is defined as an SPDK subsystem and can be disabled or enabled as required.

The Virtio-fs subsystem is enabled by default. To disable it:

1. Start the Virtio-fs application in a waiting state for RPC commands:

```
/opt/nvidia/spdk-subsystem/src/virtio-fs-
target/app/virtio_fs_tgt --wait-for-rpc
```


2. Use the following RPC command to disable the `virtio_fs` subsystem:

```
spdk_rpc.py framework_disable_subsystem virtio_fs
```

3. Restart the SPDK framework initialization:

```
spdk_rpc.py framework_start_init
```

To view the status of all subsystems:

```
spdk_rpc.py framework_get_subsystems
```

This subsystem configuration can be saved in SPDK configuration for future use with the command `spdk_rpc.py save_config`. This ensures the configuration persists across application restarts.

Appendix – BlueField Firmware Configuration

Before configuring DOCA SNAP Virtio-fs, the user must ensure that all firmware configuration requirements are met. By default, virtio-fs is disabled and must be enabled by running both common DOCA SNAP Virtio-fs configurations and additional protocol-specific configurations depending on the expected usage of the application (e.g., hot-plug, SR-IOV, UEFI boot, etc).

After configuration is finished, the host must be power cycled for the changes to take effect.

Note

To verify that all configuration requirements are satisfied, users may query the current/next configuration by running the following:

```
mlxconfig -d /dev/mst/mt41692_pciconf0 -e query
```

System Configuration Parameters

Parameter	Description	Possible Values
INTERNAL_CPU_MODEL	<p>Enable BlueField to work in internal CPU model</p> <p>Note Must be set to 1 for storage emulations.</p>	0/1
PCI_SWITCH_EMULATION_ENABLE	<p>Enable PCIe switch for emulated PFs</p>	0/1
PCI_SWITCH_EMULATION_NUM_PORT	<p>The maximum number of hotplug emulated PFs which equals <code>PCI_SWITCH_EMULATION_NUM_PORT</code> minus 1. For example, if <code>PCI_SWITCH_EMULATION_NUM_PORT=32</code>, then the maximum number of hotplug emulated PFs would be 31.</p> <p>Note One switch port is reserved for all static PFs.</p>	[0-32]

RDMA/RoCE Configuration

BlueField's RDMA/RoCE communication is blocked for BlueField's default OS interfaces (nameds ECPFs, typically mlx5_0 and mlx5_1). If RoCE traffic is required, additional network functions (scalable functions) must be added which support RDMA/RoCE traffic.

Note

The following is not required when working over TCP or even RDMA/IB.

To enable RoCE interfaces, run the following from within the BlueField device:

```
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s PER_PF_NUM_SF=1
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0 s PF_SF_BAR_SIZE=8
PF_TOTAL_SF=2
[dpu] mlxconfig -d /dev/mst/mt41692_pciconf0.1 s PF_SF_BAR_SIZE=8
PF_TOTAL_SF=2
```

Virtio-fs Configuration

Warning

Due to virtio-fs protocol limitations, using bad configuration while working with static virtio-fs PFs may cause the host server OS to fail on boot.

Before continuing, make sure you have configured:

- A working channel to access Arm even when the host is shut down. Setting such channel is out of the scope of this document. Please refer to [NVIDIA BlueField DPU BSP](#) documentation for more details.
- Use the initial configure file to create a controller on the static virtio-fs PF.

00000196-6a22-dea4-abb7-ef62c97b0001

Parameter	Description	Possible Values
VIRTIO_FS_EMULATION_ENABLE	Enable virtio-fs device emulation	0/1
VIRTIO_FS_EMULATION_NUM_PF	Number of static emulated virtio-fs PFs Note See WARNING above.	[0-2]
VIRTIO_FS_EMULATION_NUM_MSIX	Number of MSIX assigned to emulated virtio-fs PF/VF	[0-63]

Appendix – Host OS Configuration

With Linux environment on host OS, additional kernel boot parameters may be required to support DOCA SNAP Virtio-fs related features:

- To use PCIe hotplug, `pci=realloc` must be added
- `modprobe.blacklist=virtio_pci,virtiofs` for the virtio-fs driver which is not built-in

- `modprobe.blacklist=virtio_pci` for the `virtio_pci` driver which is not built-in

To view boot parameter values, run:

```
cat /proc/cmdline
```

It is recommended to use the following command with virtio-fs :

```
[dpu] cat /proc/cmdline BOOT_IMAGE ... pci=realloc  
modprobe.blacklist=virtio_pci,virtiofs
```

Intel Server Performance Optimizations

```
cat /proc/cmdline  
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.15.0_mlnx root=UUID=91528e6a-  
b7d3-4e78-9d2e-9d5ad60e8273 ro crashkernel=auto  
resume=UUID=06ff0f35-0282-4812-894e-111ae8d76768 rhgb quiet  
pci=realloc modprobe.blacklist=virtio_pci,virtiofs
```

AMD Server Performance Optimizations

```
cat /proc/cmdline  
cat /proc/cmdline BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.15.0_mlnx  
root=UUID=91528e6a-b7d3-4e78-9d2e-9d5ad60e8273 ro  
crashkernel=auto resume=UUID=06ff0f35-0282-4812-894e-111ae8d76768  
rhgb quiet pci=realloc modprobe.blacklist=virtio_pci,virtiofs
```

Appendix – Custom Fsdev Modules

This section explains how to extend the Virtio-fs service with custom fsdev modules. Two methods are provided:

- Building a new production container (recommended for production)
- Manual build and run inside a development container (ideal for development and testing)

The Dockerfiles and commands provided in this chapter are intended as examples and are designed to work with the external fsdev module example included with SPDK, located at `spdk/test/external_code/fsdev_passthru`. These examples demonstrate the integration process and should be adapted to suit real-world use cases, including adjustments for custom fsdev modules, deployment environments, and specific application requirements.

Building a New Production Container

This option allows to create a new container that includes both the original service and the custom fsdev module. The resulting container will run Virtio-fs service with custom functionality built-in.

Environment Preparation

```
# Current working directory contain spdk sources (repo -
https://github.com/Mellanox/spdk; branch - v24.01.4.nvda) and
Dockerfile
git clone https://github.com/Mellanox/spdk --branch v24.01.4.nvda

root@bf:/opt/build# pwd
/opt/build
root@bf:/opt/build# ll
total 16
drwxr-xr-x  3 root root 4096 Dec  2 14:16 ./
drwxr-xr-x  6 root root 4096 Dec  2 13:30 ../
-rw-r--r--  1 root root  723 Dec  2 14:15 Dockerfile
```

```
drwxr-xr-x 37 root root 4096 Dec  2 14:01 spdk/
```

Dockerfile Example

```
FROM nvcr.io/nvstaging/doca/doca_vfs:1.2.0-2-doca2.10.0 as
builder

RUN apt-get update && apt-get install -y autoconf libtool
python3-pyelftools libaio-dev libncurses-dev libfuse3-dev
patchelf libcmocka-dev make

COPY spdk/test/external_code /external_code

WORKDIR /external_code
ENV SPDK_HEADER_DIR=/opt/nvidia/spdk-subsystem/virtiofs/include
ENV SPDK_LIB_DIR=/opt/nvidia/spdk-subsystem/virtiofs/lib
ENV DDPK_LIB_DIR=/opt/nvidia/spdk-subsystem/virtiofs/include
RUN make fsdev_passthru_shared

FROM nvcr.io/nvstaging/doca/doca_vfs:1.2.0-2-doca2.10.0

COPY --from=builder
/external_code/fsdev_passthru/libfsdev_passthru_external.so
/opt/nvidia/spdk-subsystem/virtiofs/lib/
ENV VIRTIOFS_LD_PRELOAD=/opt/nvidia/spdk-
subsystem/virtiofs/lib/libfsdev_passthru_external.so

COPY --from=builder
/external_code/fsdev_passthru/fsdev_passthru.py
/usr/lib/python3/dist-packages/spdk/rpc/
ENV
SPDK_RPC_PLUGIN="spdk.rpc.rpc_virtio_fs_tgt:spdk.rpc.fsdev_passthru"
```

Building and Running the Container

1. Build the new container:

```
docker build -t doca_vfs_custom_fsdev:latest -f Dockerfile .
```

2. Run the new container with Kubelet service:

```
docker save doca_vfs_custom_fsdev:latest >
doca_vfs_custom_fsdev.tar
ctr -n=k8s.io images import doca_vfs_custom_fsdev.tar

# Copy doca_vfs.yaml into /etc/kubelet.d/
# Open doca_vfs.yaml and change "image" section:
    image: doca_vfs_custom_fsdev:latest
```

3. To configure running container with custom fsdev module:

```
# DPU
mkdir /etc/virtiofs/AI00
crictl exec $(crictl ps -s running -q --name virtiofs)
spdk_rpc.py fsdev_aio_create AI00 /etc/virtiofs/AI00
crictl exec $(crictl ps -s running -q --name virtiofs)
spdk_rpc.py fsdev_passthru_ext_create --name CUSTOM_FSDEV_PT0
--base-fsdev-name AI00
crictl exec $(crictl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_transport_create -t DOCA
crictl exec $(crictl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_doca_manager_create -m mlx5_0
crictl exec $(crictl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_transport_start -t DOCA
```



```

crictrl exec $(crictrl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_device_create --transport-name DOCA --
dev-name vfsdev0 --tag CUSTOM_FSDEV_PT0 --fsdev
CUSTOM_FSDEV_PT0 --num-request-queues 62 --queue-size 64 --
driver-platform x86_64
crictrl exec $(crictrl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_doca_device_modify --dev-name vfsdev0 -
-manager mlx5_0 --vuid MT2334XZ0LEDVFSS0D0F3
crictrl exec $(crictrl ps -s running -q --name virtiofs)
spdk_rpc.py virtio_fs_device_start --dev-name vfsdev0

# x86
modprobe -v virtio_pci
modprobe -v virtiofs
test -d /tmp/CUSTOM_FSDEV_PT0 || mkdir -p /tmp/CUSTOM_FSDEV_PT0
mount -t virtiofs CUSTOM_FSDEV_PT0 /tmp/CUSTOM_FSDEV_PT0

```

Manual Build and Run in a Development Container

This approach is suitable for development workflows where frequent updates, builds, and tests are required. It sets up a development container with the necessary environment to manually build and run the Virtio-fs service with a custom fsdev module.

Environment Preparation

```

# Current working directory contain spdk sources (repo -
https://github.com/Mellanox/spdk; branch - v24.01.4.nvda) and
Dockerfile
git clone https://github.com/Mellanox/spdk --branch v24.01.4.nvda

root@bf:/opt/build# pwd
/opt/build
root@bf:/opt/build# ll

```

```
total 16
drwxr-xr-x  3 root root 4096 Dec  2 14:16 ./
drwxr-xr-x  6 root root 4096 Dec  2 13:30 ../
-rw-r--r--  1 root root  723 Dec  2 14:15 Dockerfile
drwxr-xr-x 37 root root 4096 Dec  2 14:01 spdk/
```

Dockerfile Example

```
FROM nvcr.io/nvstaging/doca/doca_vfs:1.2.0-2-doca2.10.0 as
builder

RUN apt-get update && apt-get install -y autoconf libtool
python3-pyelftools libaio-dev libncurses-dev libfuse3-dev
patchelf libcmocka-dev make

ENTRYPOINT /bin/bash
```

Building and Running the Development Container

1. Build the container using the provided `Dockerfile`:

```
docker build -t doca_vfs_custom_fsdev_dev:latest -f
Dockerfile .
```

2. Start the container and enter the bash shell:

```
docker run -ti --privileged --net=host \
  --volume /dev/hugepages:/dev/hugepages \
  --volume /dev/shm:/dev/shm \
  --volume /dev/infiniband:/dev/infiniband \
```

```
--volume /etc/virtiofs:/etc/virtiofs \  
--volume ${PWD}/spdk/test/external_code:/external_code \  
doca_vfs_custom_fsdev_dev:latest
```

3. Once inside the container, one can manually build and test custom module:

```
cd /external_code/  
export SPDK_HEADER_DIR=/opt/nvidia/spdk-  
subsystem/virtiofs/include  
export SPDK_LIB_DIR=/opt/nvidia/spdk-subsystem/virtiofs/lib  
export DPDK_LIB_DIR=/opt/nvidia/spdk-subsystem/virtiofs/include  
make fsdev_passthru_shared
```

4. Start the virtio-fs service with the custom module for testing:

```
cp /external_code/fsdev_passthru/fsdev_passthru.py /usr/bin/  
/opt/nvidia/spdk-  
subsystem/doca_snap/bin/set_environment_variables.sh  
LD_PRELOAD=/external_code/fsdev_passthru/libfsdev_passthru_ext  
\  
/opt/nvidia/spdk-subsystem/virtiofs/bin/virtio_fs_tgt -m  
0xff &
```

5. To configure the service with custom fsdev module:

```
# DPU  
mkdir /etc/virtiofs/AI00  
spdk_rpc.py fsdev_aio_create AI00 /etc/virtiofs/AI00  
spdk_rpc.py --plugin fsdev_passthru fsdev_passthru_ext_create  
--name CUSTOM_FSDEV_PT0 --base-fsdev-name AI00  
spdk_rpc.py virtio_fs_transport_create -t DOCA
```

```

spdk_rpc.py virtio_fs_doca_manager_create -m mlx5_0
spdk_rpc.py virtio_fs_transport_start -t DOCA
spdk_rpc.py virtio_fs_device_create --transport-name DOCA --
dev-name vfsdev0 --tag CUSTOM_FSDEV_PT0 --fsdev
CUSTOM_FSDEV_PT0 --num-request-queues 62 --queue-size 64 --
driver-platform x86_64
spdk_rpc.py virtio_fs_doca_device_modify --dev-name vfsdev0 -
-manager mlx5_0 --vuid MT2334XZ0LEDVFSS0D0F2
spdk_rpc.py virtio_fs_device_start --dev-name vfsdev0

# x86
modprobe -v virtio_pci
modprobe -v virtiofs
test -d /tmp/CUSTOM_FSDEV_PT0 || mkdir -p /tmp/CUSTOM_FSDEV_PT0
mount -t virtiofs CUSTOM_FSDEV_PT0 /tmp/CUSTOM_FSDEV_PT0

```

References

Title	Description
NVIDIA DOCA	NVIDIA DOCA™ SDK enables developers to rapidly create applications and services on top of NVIDIA® BlueField® networking platform, leveraging industry-standard APIs
NVIDIA BlueField BSP	BlueField Board Support Package includes the bootloaders and other essentials for loading and setting software components
BlueField DPU Hardware User Manual	This document provides details as to the interfaces of the BlueField DPU, specifications, required software and firmware for operating the device, and a step-by-step plan for bringing the DPU up
NVIDIA BlueField BSP	This document provides product release notes as well as information on the BlueField software distribution and how to develop and/or customize applications, system software, and file system images for the BlueField platform

Title	Description
Documentation	
DOCA Device Emulation	DOCA Device Emulation library documentation. The DOCA Device Emulation subsystem provides a low-level software API for users to develop PCIe devices and their controllers.
DOCA DevEmu Virtio-fs	DOCA Device Emulation Virtio-fs library documentation. The DOCA DevEmu Virtio-fs library is part of the DOCA DevEmu Virtio subsystem. It provides low-level software APIs that provide building blocks for developing and manipulating virtio filesystem devices using the device emulation capability of BlueField platforms.
DOCA DevEmu PCI	DOCA Device Emulation PCI library documentation. DOCA DevEmu PCI is part of the DOCA Device Emulation subsystem. It provides low-level software APIs that allow management of an emulated PCIe device using the emulation capability of NVIDIA® BlueField® networking platforms.

SNAP Virtio-fs Service

Release Notes

The release notes provide information for the DOCA SNAP Virtio-fs Service such as changes and new features, software known issues, and bug fixes.

Changes and New Features

Key Features in Version 1.2.0-doca2.10.0

- Virtio-fs container with customizable SPDK FSDEV.
- Virtio-fs support for cache invalidate and fuse notification queue - required NVIDIA modified Virtio-fs driver.
- Virtio-fs 254 IO queues per emulation function, total limitation is 2000.

Limitations

The following features are currently not supported in this version of the application:

- End-to-end container solution does not support: device recovery, live update and live migration
- Dynamic MSIX

Known Issues

DOCA SNAP Virtio-fs Issues

The following are known limitations of DOCA SNAP Virtio-fs software version.

Ref #	Issue
-	<p>Description: Due to the lack of recovery support, it is not possible to perform any negative/resilience operations during IO traffic (e.g., destroy and restore).</p> <div data-bbox="217 438 1463 680" style="background-color: #ffffcc; padding: 10px;"> <p>i Info Restarting the device using <code>virtio_fs_device_stop</code> and <code>virtio_fs_device_start</code> is supported.</p> </div> <p>Workaround: N/A</p> <p>Keywords: Recovery; negative/resilience operations</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
-	<p>Description: The following FUSE commands are unsupported: BMAP, SETUPMAPPING, REMOVEMAPPING.</p> <p>Workaround: N/A</p> <p>Keywords: FUSE</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
-	<p>Description: Application restart is not allowed if the application controller has processed FUSE commands.</p> <p>Workaround: Unload the virtio-fs driver on the host, then restart the application.</p> <p>Keywords: FUSE</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
-	<p>Description: The total number of virtio queues the application can create is limited to 2,000.</p> <p>Workaround: N/A.</p> <p>Keywords: Virtio queues</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
-	<p>Description: The following operations are not supported when using Linux's virtio-fs inbox/upstream kernel driver: FLR and the virtio-fs notification queue.</p>

Ref #	Issue
	<p>Workaround: N/A</p> <p>Keywords: FLR; virtio-fs; inbox/upstream kernel driver</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
-	<p>Description: IO errors on host can be seen under high scale. Due to lack of mempool resources, more likely to be seen with 8 PFs.</p> <p>Workaround: Reduce overall outstanding IO traffic.</p> <p>Keywords: IO error, host error</p> <p>Discovered in version: 1.2.0-doca2.10.0</p>
-	<p>Description: I/O with size greater than 255 KB is not supported due to the data pool size restriction of 255 KB in the VirtioFS process running on the DPU.</p> <p>Workaround: Set the "max_xfer_size" in the <code>fsdev_aio_create</code> RPC to a value less than 255 KB. This value is used as "max_write" in the FUSE layer of the host kernel.</p> <p>If the I/O size exceeds <code>max_xfer_size</code>, the FUSE layer in the host kernel will split the I/O into smaller chunks.</p> <p>The default value of "max_xfer_size" in fsdev_aio_create RPC is 128KB.</p> <p>For example, If the user attempts an I/O operation of 1MB while "max_xfer_size" is set to 64KB, the FUSE layer on the host will split the request into 64KB chunks. As a result, the VirtioFS process on the DPU will receive 16 separate I/O requests (1MB ÷ 64KB).</p> <p>Keywords: IO error, host error, io_size</p> <p>Discovered in version: 1.2.0-doca2.10.0</p>

OS or Vendor Issues

Info

The following are not DOCA SNAP Virtio-fs limitations.

R e f #	Issue
–	<p>Description: If the FLR is initiated from the host by writing <code>1</code> to the <code>reset</code> file under <code>/sys/bus/pci/devices/bdf/</code> using the command <code>echo 1 > /sys/bus/pci/devices/0000\ :29\ :00.2/reset</code>, then the host driver does not create virtqueues, causing the mount point to remain stuck indefinitely.</p> <p>Workaround: FLR should only be performed without any mount over virtio-fs on the host. To run IO after FLR, reload the virtio-fs host driver.</p> <p>Keywords: Driver; FLR</p> <p>Discovered in version: 1.0.0-doca2.8.0</p>
–	<p>Description: On the host, when the virtio-fs mount is idle (i.e., no I/O operations), dmesg logs are filled with repeated AppArmor DENIED messages. These messages indicate that the ntpd service is being denied access to specific files by AppArmor. The ntpd service is trying to access <code>/snap/bin/</code> and <code>/etc/ssl/openssl.cnf</code>, but the AppArmor profile for ntpd does not permit these accesses, resulting in denied requests.</p> <p>Workaround: Modify the AppArmor profile for ntpd to grant the required read permissions.</p> <ol style="list-style-type: none"> 1. Locate the AppArmor profile for ntpd. It is typically located in <code>/etc/apparmor.d/</code> and named <code>usr.sbin.ntpd</code>. 2. Edit the profile and add the required permissions by using a text editor. For example: <ol style="list-style-type: none"> 1. Run: <pre style="border: 1px solid #ccc; padding: 10px; margin: 5px 0;">sudo nano /etc/apparmor.d/usr.sbin.ntpd</pre> 2. Add the following lines to allow ntpd to read the necessary files: <pre style="border: 1px solid #ccc; padding: 10px; margin: 5px 0;">/snap/bin/ r, /etc/ssl/openssl.cnf r`</pre> 3. Apply the changes by reloading the AppArmor profile:

R e f #	Issue
	<pre>sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.ntpd</pre>
	Keywords: AppArmor, ntpd
	Discovered in version: 1.0.0-doca2.8.0
-	Description: With a kernel version older than 6.10, if stress loading and unloading of the <code>virtio_pci</code> and <code>virtiofs</code> drivers is done in a loop, or mount and umount in a loop, then unmounting or unloading the driver may hang.
	Workaround: Add a delay of 1 second between loading and unloading of the drivers.
	Keywords: virtio_pci; virtiofs
	Discovered in version: 1.0.0-doca2.8.0

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document. NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk. NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or

endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2025, NVIDIA. PDF Generated on 05/05/2025