NVIDIA DOCA SNAP Virtio-fs Service Guide

Table of contents

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, t he host/guest uses its own standard virtio-fs driver which is fully isolated from the networked filesystem mounted within the BlueField .

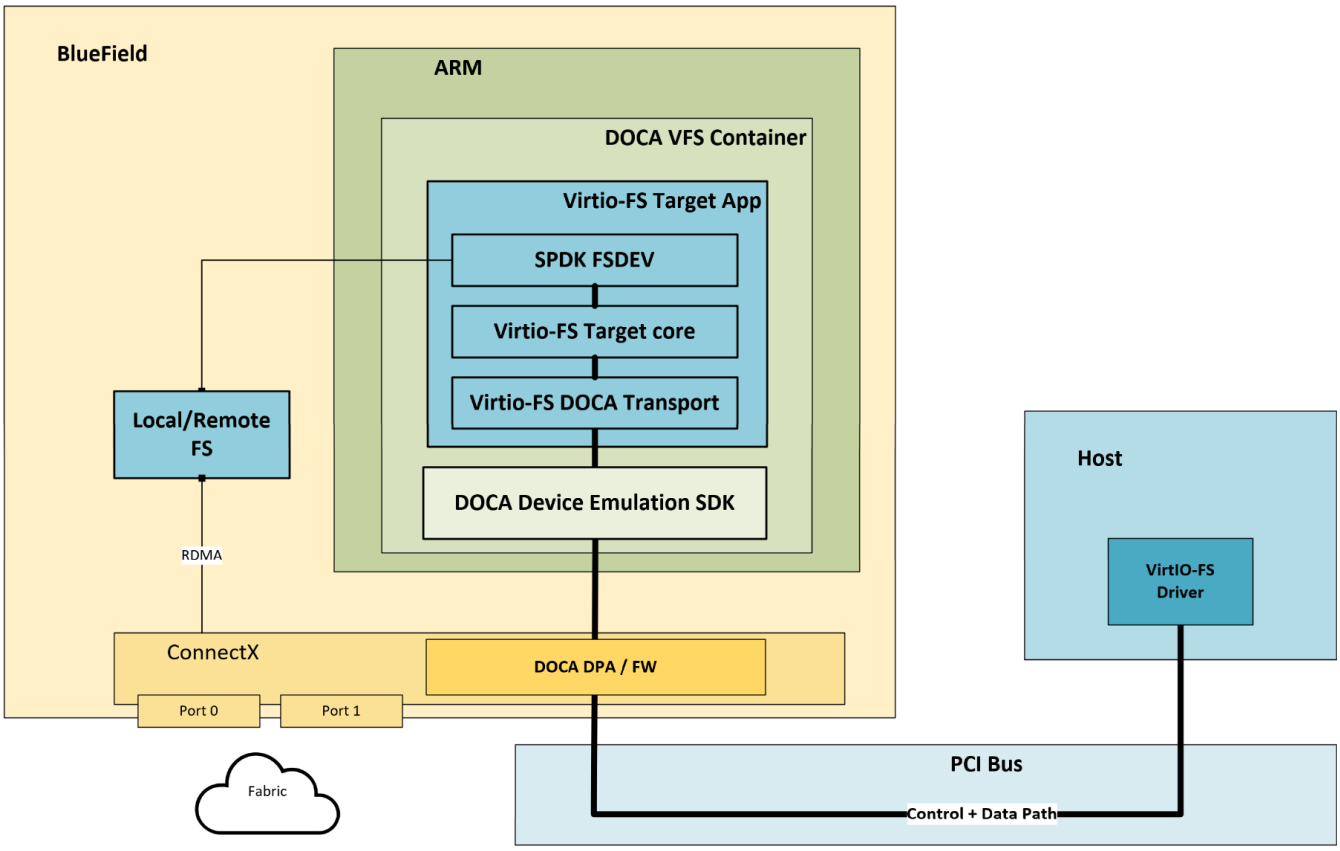
Built upon the DOCA and SPDK frameworks, virtio-fs device emulation on BlueField devices offers a comprehensive set of libraries for BlueField-based solutions and for storage solutions. This architecture consists of several key components:

- 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.

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
```

Note

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

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), `mlx5_0` and `mlx5_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"
  ```

  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 form `/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 associated packages, natively supports DOCA SNAP 4.x.x, which is implemented as an SPDK subsystem module, allowing the concurrent operation of both virtio-fs and virtio-blk as a unified service. Additionally, DOCA SNAP is deployed as part of the DOCA SNAP Virtio-fs deployment.

> **Info**

Refer to NVIDIA BlueField-3 SNAP for NVMe and Virtio-blk documentation here.

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 `crictl` tool.

Info

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

- To query the active container ID:

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

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

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

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

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

Alternatively, an alias can be used:

```
crictl exec -it $(crictl 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:

```
crictl 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
`manager`	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
`manager`	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 n ot needed for static functions as the VUID can be retrieved from `virtio_fs_doca_get_functions`.

Parameter Name	Optional/Mandatory	Type	Description
`manager`	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
`manager`	Mandatory	String	Emulation manager device name for destroying a virtio-fs function
`vuid`	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
`dev_name`	Mandatory	String	Virtio-fs device name to hot plug
`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

Parameter Name	Optional/Mandatory	Type	Description
`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 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
`name`	Optional	string	Name of the fsdev of interest

fsdev_aio_create

Create an SPDK AIO FSdev ,

Parameter Name	Optional/Mandatory	Type	Description
`name`	Mandatory	string	Name of the AIO FSdev to create
`root_path`	Mandatory	string	Path on the system directory to be exposed as an SPDK filesystem
`enable_xattr`	Optional	bool	Enable extended attributes if set to `true`; `false` by default
`enable_writeback_cache`	Optional	bool	Enable the writeback cache if set to `true`; `false` by default
`max_write`	Optional	int	Maximum write size in bytes; `0x00020000` by default
`enable_skip_rw`	Optional	bool	Enable skipping read/write IOs if set to `true`; `false` by default **Note** For debug purposes only.

fsdev_aio_delete

Delete an AIO FSdev .

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

virtio_fs_get_transports

Display virtio-fs transports or requested transport.

Parameter Name	Optional/Mandatory	Type	Description
`transport_name`	Optional	String	Transport type name. For DOCA SNAP Virtio-fs, `transport_name` 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
`transport_name`	Mandatory	String	Transport type name. For DOCA SNAP Virtio-fs, `transport_name` should be DOCA.
`dev_name`	Mandatory	String	Virtio-fs device name to use
`tag`	Optional	String	Virtio-fs tag according to the virtio specification. **Note** Must be provided during the `virtio_fs_device_create` RPC before the `virtio_fs_device_start` RPC.
`num_request_queues`	Optional	Number	Virtio-fs `num_request_queues` according to the virtio specification (default 31, range 1-62)
`queue_size`	Optional	Number	The maximal queue size for all virtio queues (default 64, range 8-256)
`fsdev`	Optional	String	The name of the SPDK filesystem backend device **Note** Must be provided during the `virtio_fs_device_create` RPC before the `virtio_fs_device_start` RPC. **Note**

Parameter Name	Optional/Mandatory	Type	Description
			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.
`packed_vq`	Optional	Bool	Expose packed virtqueues feature to the driver for negotiation.
`driver_platform`	Optional	String	Set the driver's platform architecture. Possible values: `native`; `x86`; `x86_64`; `aarch32`; `aarch64`. Using the `native` platform option sets the driver platform to be identical to the device platform.

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
`dev_name`	Mandatory	String	Virtio-fs device name

virtio_fs_device_stop

Stop a virtio-fs device.

Parameter Name	Optional/Mandatory	Type	Description
`dev_name`	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
`dev_name`	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
`dev_name`	Mandatory	String	Virtio-fs device name to use

Parameter Name	Optional/Mandatory	Type	Description
`tag`	Optional	String	Virtio-fs tag according to the virtio specification **Note** Must be provided during `virtio_fs_device_create` or `virtio_fs_device_modify` RPCs, before `virtio_fs_device_start` RPC.
`num_request_queues`	Optional	Number	Virtio-fs `num_request_queues` according to the virtio specification (default 31; range 1-62)
`queue_size`	Optional	Number	The maximal queue size for all virtio queues (default 64; range 8-256)

Parameter Name	Optional/Mandatory	Type	Description
`fsdev`	Optional	String	The name of the SPDK filesystem backend device **Note** Must be provided during `virtio_fs_device_create` or `virtio_fs_device_modify` RPCs, before `virtio_fs_device_start` RPC. **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.
`packed_vq`	Optional	Bool	Expose packed virtqueues feature to the driver for negotiation
`driver_platform`	Optional	String	Set the driver's platform architecture. Possible values: `native`; `x86`; `x86_64`; `aarch32`; `aarch64`. Using the `native` 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** VUID validation is not done. If an invalid VUID is set, `virtio_fs_device_start` 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
    ```

    2. 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
    ```

    3. Get device information:

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

    4. 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
    ```

    5. Destroy the virtio-fs function:

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

6. 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
```

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`	Enable BlueField to work in internal CPU model **Note** Must be set to `1` for storage emulations.	0/1
`PCI_SWITCH_EMULATION_ENABLE`	Enable PCIe switch for emulated PFs	0/1
`PCI_SWITCH_EMULATION_NUM_PORT`	The maximum number of hotplug emulated PFs which equals `PCI_SWITCH_EMULATION_NUM_PORT` minus 1. For example, if `PCI_SWITCH_EMULATION_NUM_PORT=32`, then the maximum number of hotplug emulated PFs would be 31. **Note** One switch port is reserved for all static PFs.	[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.

  00000193-dbcf-d1b6-afb3-fbcf743d0000

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
```

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 Documentation	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
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 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 1.1.0-doca2.9.0

- Fuse commands support
- 31 hotplug functions support
- Signed DPA
- Resolved a known issue where restarting the application or controller was not permitted after the controller had processed FUSE commands

Limitations

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

- The ability to handle device recovery
- Live update
- Live migration
- Dynamic MSIX
- Only 62 queues per emulation function are supported
- Building a custom container via the SDK

Known Issues

DOCA SNAP Virtio-fs Issues

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

Ref #	Issue
_	Description: Currently, the only supported protocol is NFS-over-TCP. NFS-over-RDMA is not supported.
	Workaround: N/A
	Keywords: NFS-over-TCP; NFS-over-RDMA
	Discovered in version: 1.0.0-doca2.8.0
_	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). **Info** Restarting the device using `virtio_fs_device_stop` and `virtio_fs_device_start` is supported.
	Workaround: N/A
	Keywords: Recovery; negative/resilience operations
	Discovered in version: 1.0.0-doca2.8.0
_	Description: The following FUSE commands are unsupported: BMAP, SETUPMAPPING, REMOVEMAPPING.
	Workaround: N/A
	Keywords: FUSE
	Discovered in version: 1.0.0-doca2.8.0
–	Description: Application restart is not allowed if the application controller has processed FUSE commands.
	Workaround: Unload the virtio-fs driver on the host, then restart the application.

Ref #	Issue
	Keywords: FUSE
	Discovered in version: 1.0.0-doca2.8.0
_	Description: The total number of virtio queues the application can create is limited to 2,000.
	Workaround: N/A.
	Keywords: Virtio queues
	Discovered in version: 1.0.0-doca2.8.0
_	Description: The following operations are not supported when using Linux's virtio-fs inbox/upstream kernel driver: FLR and the virtio-fs notification queue.
	Workaround: N/A
	Keywords: FLR; virtio-fs; inbox/upstream kernel driver
	Discovered in version: 1.0.0-doca2.8.0

OS or Vendor Issues

Info

The following are not DOCA SNAP Virtio-fs limitations.

Ref #	Issue
–	Description: If the FLR is initiated from the host by writing `1` to the `reset` file under `/sys/bus/pci/devices/bdf/` using the command `echo 1 > /sys/bus/pci/devices/0000\:29\:00.2/reset`, then the host driver does not create virtqueues, causing the mount point to remain stuck indefinitely.

Ref #	Issue
	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.
	Keywords: Driver; FLR
	Discovered in version: 1.0.0-doca2.8.0
–	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 `/snap/bin/` and `/etc/ssl/openssl.cnf`, but the AppArmor profile for ntpd does not permit these accesses, resulting in denied requests.
	Workaround: Modify the AppArmor profile for ntpd to grant the required read permissions. 1. Locate the AppArmor profile for ntpd. It is typically located in `/etc/apparmor.d/` and named `usr.sbin.ntpd`. 2. Edit the profile and add the required permissions by using a text editor. For example: 1. Run: ``` sudo nano /etc/apparmor.d/usr.sbin.ntpd ``` 2. Add the following lines to allow ntpd to read the necessary files: ``` /snap/bin/ r, /etc/ssl/openssl.cnf r` ``` 3. Apply the changes by reloading the AppArmor profile: ``` sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.ntpd ```
	Keywords: AppArmor, ntpd
	Discovered in version: 1.0.0-doca2.8.0

Ref #	Issue
–	Description: With a kernel version older than 6.10, if stress loading and unloading of the `virtio_pci` and `virtiofs` 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.