NVIDIA BlueField-2 SNAP for NVMe and Virtio-blk v3.8.0-5

Table of contents

Overview 4

Release Notes 6

- Changes and New Features 6
- Known Issues 6
- Changes and New Features History Log 13

SNAP Installation 19

SNAP Commands 23

Advanced Features 38

Appendixes 53

- Appendix – Live Upgrade 53
- Appendix – Host OS Configuration 56
- Appendix – SPDK Configuration 57
- Appendix – Firmware Configuration 57
- Appendix – JSON File Format 61
- Appendix – Frequently Asked Questions 68

Document Revision History 75

About This Document

This document describes the configuration parameters of NVIDIA® BlueField®-2 SNAP and virtio-blk SNAP in detail.

> ⚠ **Attention**
> In the future, documentation for SNAP-3 will be integrated into DOCA.

Audience

This manual is intended for BlueField SNAP or virtio-blk SNAP users who need to install and configure it.

Technical Support

Customers who purchased NVIDIA products directly from NVIDIA are invited to contact us through the following methods:

- E-mail: enterprisesupport@nvidia.com
- Enterprise Support page: https://www.nvidia.com/en-us/support/enterprise

Customers who purchased NVIDIA M-1 Global Support Services, please see your contract for details regarding technical support.

Customers who purchased NVIDIA products through an NVIDIA-approved reseller should first seek assistance through their reseller.

Glossary

Term	Description
CLI	Command line interface
BFB	BlueField bootstream
DMA	Direct memory access
ETH	Ethernet
FW	Firmware
I/O	Input/output
IB	InfiniBand
NVMe	Non-volatile memory express
OS	Operating system
PF	Physical function
RPC	Remote procedure call
SF	Scalable function
SNAP	Software-defined network accelerated processing

Related Documents

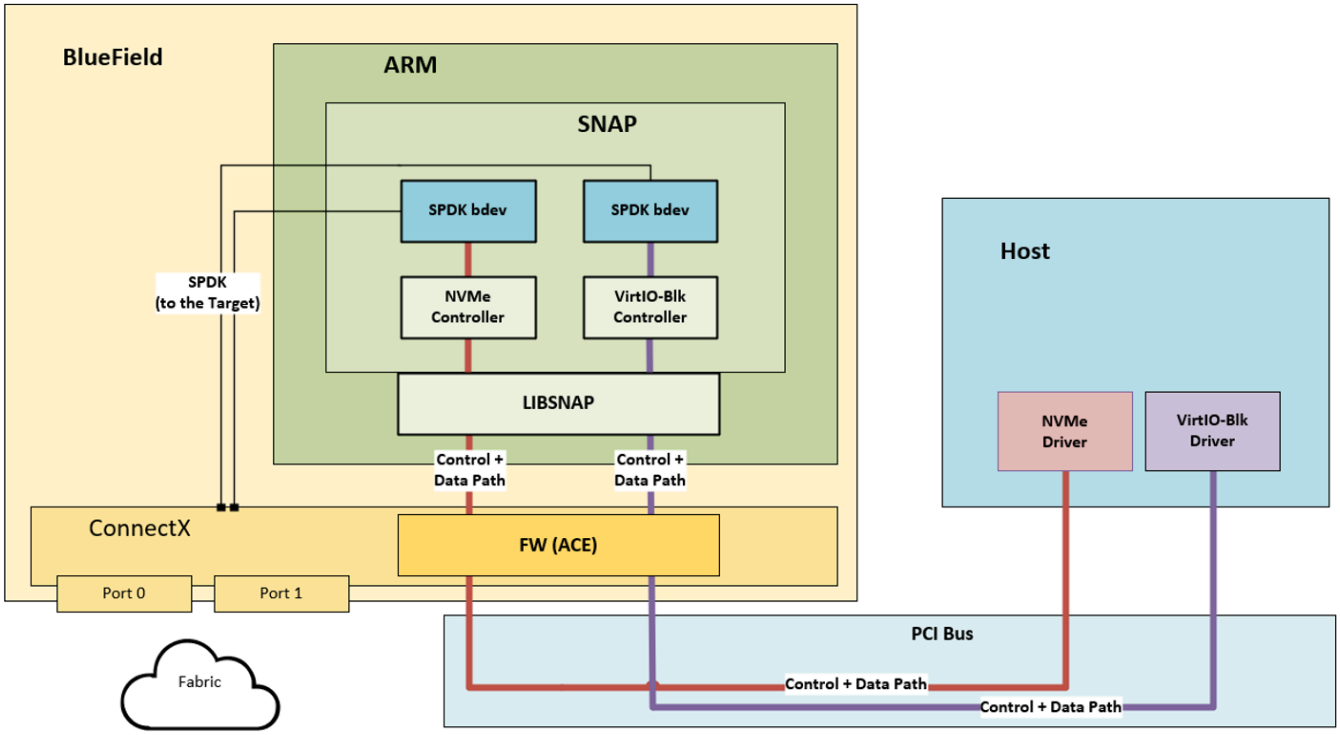
Title	Description
BlueField DPU Hardware User Manual	This document provides details as to the interfaces of the BlueField DPU, specifications, required software and firmware for operating the device, and a step-by-step plan for bringing the DPU up
NVIDIA BlueField DPU Platform Operating System Documentation	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
NVIDIA Accelerated IO (XLIO) Documentation	This document covers product release notes as well as features of XLIO. XLIO is a user-space software library that exposes standard socket APIs with kernel-bypass architecture, enabling a hardware-based direct copy between an application's user-space memory and the network interface.

Overview

SNAP

NVIDIA® BlueField® SNAP and virtio-blk SNAP (Storage-defined Network Accelerated Processing) technology enables hardware-accelerated virtualization of local storage. NVMe/virtio-blk SNAP presents networked storage as a local block-storage-device such as an SSD, emulating a local drive on the PCIe bus. The host OS/hypervisor uses its standard storage driver, unaware that communication is done, not with a physical drive, but with NVMe/virtio-blk SNAP framework. Any logic may be applied to the I/O requests or to the data via the NVMe/virtio-blk SNAP framework prior to redirecting the request and/or data over a fabric-based network to remote or local storage targets.

NVMe/virtio-blk SNAP is based on NVIDIA® BlueField-2 DPU family technology and combines unique hardware-accelerated storage virtualization with the advanced networking and programmability capabilities of the DPU. NVMe/virtio-blk SNAP together with the BlueField DPU enable a world of applications addressing storage and networking efficiency and performance.



The traffic from a host-emulated PCIe device is redirected to its matching storage controller opened on the mlnx_snap service. The controller, from its side, holds at least one

open backend device (usually SPDK block device). When a command is received, the controller executes it. Admin commands are answered immediately, while I/O commands are redirected to the backend device for processing. The request handling pipeline is completely asynchronous and the workload is distributed across all Arm cores (allocated to SPDK application) to achieve the best performance.

The following are key concepts for SNAP:

- Full flexibility in fabric/transport/protocol (e.g. NVMe-oF/iSCSI/other, RDMA/TCP, ETH/IB)
- NVMe and virtio-blk emulation support
- Easy data manipulation
- Using Arm cores for data path

> **Note**
>
> BlueField SNAP/virtio-blk SNAP are licensed software. Users must purchase a license per BlueField device to use them.

Libsnap

Libsnap is a common library designed to assist in common tasks for applications wishing to interact with emulated hardware over BlueField DPUs and take the most advantage from the hardware capabilities. As such, libsnap exposes a simple API for the upper layer application to create, modify, query, and destroy different emulation objects, such PCIe BAR management, emulated queues etc.

In addition, the library provides a set of helper functions to perform efficient DMA transactions between host and DPU memory.

SNAP application makes extensive usage of the libsnap library for resource management and efficient DMA operations required by the storage controllers.

Release Notes

- Changes and New Features
- Known Issues
- Changes and New Features History Log

Changes and New Features

Key Features in Version 3.8.0-5

- Added a fix for SNAP crash in the error flow
- Resolved issue where SN changes after replacing bdev
- Removed NVMe-oF TCP (XLIO) support on NVIDIA® BlueField®-2

Known Issues

The following are known limitations of this NVMe/virtio-blk SNAP software version.

Ref #	Issue
–	Description: NVMeTCP XLIO is currently not supported when running 64K page size kernels on the DPU Arm cores (as is the case for CentOS 8.x, Rocky 8.x, or openEuler 20.x).
	Workaround: N/A
	Keywords: 64K page size; NVMeTCP XLIO
	Discovered in version: 3.6.0
–	Description: When running with virtio-blk and virtio-net protocols in parallel, performance may be negatively impacted.
	Workaround: N/A

Ref #	Issue
	Keywords: Performance
	Discovered in version: 3.7.2
2957317	Description: Due to an upstream kernel bug that exists in some Linux kernel distributions, the command emulation_device_detach times out, which causes any inflight traffic to hang.
	Workaround: It is recommended to ensure that all inflight traffic on the device is stopped before performing a hotunplug.
	Keywords: PCIe Hotplug
	Discovered in version: 3.6.0
3046440	Description: NVMe full-offload mode does not work properly over the first generation of BlueField SoCs
	Workaround: N/A
	Keywords: NVMe full-offload mode
	Discovered in version: 3.6.0
2879262	Description: Due to a kernel bug that exists in some Linux kernel distributions, configuring large number of virtio queues along with a small number of MSIX may lead the kernel to a soft lock-up (on top of causing significant performance degradation).
	Workaround: It is recommended that to keep virtio-blk controller's --num_queues value in snap_rpc.py controller_virtio_blk_create is smaller than the value of VIRTIO_BLK_EMULATION_NUM_MSIX (which is configured through mlxconfig).
	Keywords: Virtio-blk; kernel hang
	Discovered in version: 3.6.0
_	Description: SPDK multipath is supported only with NVMe over RDMA (and not with NVMe over TCP).
	Workaround: N/A
	Keywords: SPDK; NVMe
	Discovered in version: 3.6.0
3055119	Description: Windows driver does not work with Virtio-blk SNAP-Direct feature.
	Workaround: To disable the feature when working with Windows OS, user must set VIRTIO_BLK_SNAP_ZCOPY=0 in /etc/default/mlnx_snap.

Ref #	Issue
	Keywords: Windows
	Discovered in version: 3.5.0
_	Description: NVMe multipath features cannot be obtained when using SNAP in full-offload mode configuration
	Workaround: N/A
	Keywords: NVMe full-offload mode; multipath
	Discovered in version: 3.4.0
_	Description: After each PCIe device hot-plug, a matching controller must be immediately opened. Specifically, hot-unplugging the device before a controller is created may cause the host kernel driver to malfunction on some Linux distributions.
	Workaround: N/A
	Keywords: Hot-plug; controller
	Discovered in version: 3.3.0
_	Description: SR-IOV on hot-plugged PFs is not supported
	Workaround: N/A
	Keywords: PCIe Hotplug
	Discovered in version: 3.2.0
_	Description: Any PCIe emulated device exposed to the host must have a matching controller opened on it in mlnx_snap service prior to loading its kernel driver. This includes virtio-net devices too.
	Workaround: N/A
	Keywords: VF; PF; virtio-net; kernel driver
	Discovered in version: 3.1.0
–	Description: It is not possible to attach block devices using the same nsid to different NVMe controllers which are linked to the same NVMe subsystem. For example, the following commands will result with an error as both controllers are attached with NSID 1:

```
snap_rpc.py controller_nvme_namespace_attach NvmeEmu2pf0
spdk Null0 1
```

Ref #	Issue
	`snap_rpc.py controller_nvme_namespace_attach NvmeEmu2pf1 spdk Null1 1`
	Workaround: N/A
	Keywords: Block device; controller
	Discovered in version: 3.0.0
_	Description: mlnx_snap NVMe controller supports an admin queue with a maximum size of 1024 towards the host.
	Workaround: N/A
	Keywords: Admin queue; controller
	Discovered in version: 3.0.0
_	Description: The DPU expansion ROM includes NVMe and virtio-blk UEFI drivers certified by NVIDIA, which should be used by the BIOS. Any other BIOS drivers are not guaranteed to work properly.
	Workaround: N/A
	Keywords: BIOS; certified drivers
	Discovered in version: 3.0.0
_	Description: Legacy interrupts are not supported.
	Workaround: N/A
	Keywords: Block device; controller
	Discovered in version: 3.0.0

Windows OS Known Issues

Note

The following are not BlueField SNAP limitations.

Ref #	Issue
3543249	Description: When using hotplugged PCIe devices, after all devices are plugged, the host must be rebooted for Windows to detect all devices.
	Workaround: N/A
	Keywords: Hotplug
	Discovered in version: 3.7.4
3521378	Description: For a successful emulation_device_detach RPC command, it is recommended to use directio=1 (O_DIRECT) with virtio-blk controller created on hot-plugged emulation. **Note** If directio=0 is used, the IO must be stopped manually. Otherwise, emulation_device_detach may fail.
	Workaround: N/A
	Keywords: Virtio-blk; RPC
	Discovered in version: 3.7.4
2957317	Description: Setting virtio-blk emulation on bare metal will end with server crash.
	Workaround: Set the seg_max flag of the virtio-blk controller to at least 16 (default is 1) using the following RPC: `controller_virtio_blk_create … --seg_max 16`
	Keywords: Virtio-blk; bare-metal; seg_max
	Discovered in version: 3.7.2
3056533	Description: When using NVMe driver in Windows, if I/O is not completed for more than 120 seconds, Windows starts ignoring the NVMe device and its disks disappear.
	Workaround: N/A
	Keywords: NVMe device disappears

Ref #	Issue
	Discovered in version: 3.6.1
N/A	Description: There is a Windows driver known issue that it may crash when attaching multiple namespaces simultaneously. Users must attach namespaces one-by-one, and verify each namespace is discovered by the OS before attaching a new one.
	Workaround: N/A
	Keywords: Attaching multiple namespaces simultaneously
	Discovered in version: 3.4.0
N/A	Description: There is a known Windows NVMe driver bug which causes Windows initiators to crash if the NVMe driver is started and no target is up and ready. Therefore, if users work with Windows OS on top of the emulated NVMe device, they must make sure that mlnx_snap NVMe controller is connected to the remote target before running the driver on the host side.
	Workaround: N/A
	Keywords: Windows initiators crash
	Discovered in version: 3.1.0
N/A	Description: There is a known Windows driver bug in which namespaces hotplug is not supported. On newer Windows builds, NVMe controller quirks must be set to 0x5. For more information, please see section "Controller Parameters".
	Workaround: N/A
	Keywords: Namespaces hotplug
	Discovered in version: 3.1.0

Virtio-blk Linux Driver Limitations

Ref #	Issue
3066750	Description: Driver does not support PCIe function level reset (FLR). Running FLR during IO causes the IO (and kernel) to hang.
	Workaround: N/A
	Keywords: PCIe function; hang
	Discovered in version: 3.6.1

Ref #	Issue
2879262	Description: When working with a large number of virtqueues (⩾ 64) over a single MSIX, the host kernel might experience soft lockup. Specifically, setting --num_queues to a high number, which is also higher than the configured --num_msix value, might cause this issue.
	Workaround:
	Keywords: Kernel; hang; virtqueues
	Discovered in version: 3.6.1
2957317	Description: In Linux kernel version 5.4.0-91-generic and above, the command emulation_device_detach times out if I/O traffic is running.
	Workaround: N/A
	Keywords: Command time out
	Discovered in version: 3.6.1

Virtio-blk Transitional Device Limitations

Ref #	Issue
3231721	Description: When using emulation_device_attach RPC to hot plug a virtio-blk transitional device, the capacity and block size attributes must be provided for this hot-plugged virtio-blk transitional device.
	Workaround: Use the --bdev_type spdk and --bdev spdk_bdev options to provide a bdev to the hot-plugged virtio-blk transitional device when using emulation_device_attach RPC.
	Keywords: Hot plugging virtio-blk transitional device
	Discovered in version: 3.7.0
_	Description: L egacy/transitional drivers do not require syncing with the device upon driver initialization. Therefore, it is highly recommended that the SNAP controller is opened on the PCIe function before the driver becomes operational. If the driver becomes operational before the controller, controller configuration options would be very limited.
	Workaround: N/A
	Keywords: Legacy; SNAP controller; SNAP driver
	Discovered in version: 3.7.0

Ref #	Issue
–	Description: L egacy/transitional device support naturally includes backends with 512B block size. Using backends with any other block size (e.g., 4K) can only be achieved when SNAP controller is opened before driver is activated.
	Workaround: N/A
	Keywords: Legacy; backend block size
	Discovered in version: 3.7.0

Changes and New Features History Log

Key Features in Version 3.8.0

N/A

Key Features in Version 3.7.4

N/A

Key Features in Version 3.7.3

- SNAP crash fixes
- Fixed wrong pci_bdf "00:00.0"

Key Features in Version 3.7.2

N/A

Key Features in Version 3.7.0

Protocol	Feature	Description
Virtio-	Virtio-blk Transitional	Added support for virtio transitional devices (i.e., devices supporting drivers conforming to modern specification and

Protocol	Feature	Description
blk	Device (0.95)	legacy drivers)
Virtio-blk	Virtio-blk Live Migration	Added support for live migration which allows system administrators to pass devices between virtual machines in a live running system
All	SNAP-Direct support for multipath SPDK bdev	Added SNAP-Direct support ("zero-copy") for SPDK bdevs which implement SPDK's multipath functionality. When enabled, this feature is supported for both active-passive and active-active modes.
All	SNAP-Direct support for LVOL/RAID SPDK bdev	Added SNAP-Direct support ("zero-copy") for SPDK bdevs which execute SPDK's logical volumes (LVOL) or RAID management

Key Features in Version 3.6.1

Protocol	Feature	Description
TCP XLIO	Bug fixes	N/A

Key Features in Version 3.6.0

Protocol	Feature	Description
NVMe	NVMe/TCP Zero Copy	Implemented as a custom NVDA_TCP transport in SPDK NVMe initiator. This feature is based on a new XLIO socket layer implementation. This feature is disabled by default.

Key Features in Version 3.5.0

Protocol	Feature	Description
NVMe	NVMe UEFI driver improvements	Optimized NVMe UEFI driver to complete operation faster so SNAP NVMe devices do not delay BIOS for a long period during boot time

Protocol	Feature	Description
All	New snap-direct implementation	Implemented snap-direct (or "zero-copy") functionality using non-NVIDIA-proprietary SPDK interface so it can now work over any supported SPDK framework (SPDK ⩾ 21.07).
All	SNAP memory pool	Enabled sharing of memory buffers between all devices when working with multiple SNAP devices (e.g., SR-IOV) to dramatically improve scalability

Key Features in Version 3.4.0

Protocol	Feature	Description
All	Improved scalability for virtual functions (SR-IOV)	Added ability for user to configure a global buffer pool to be used by all (or some) controllers, which removes the memory boundary from creating a large number of controllers simultaneously (like in SR-IOV)
All	Improved hotplug/unplug behavior	Added ability for user to detach (hot-unplug) a PCIe function even when the host driver is still active (see "Working with PCIe Hotplug" for limitations)
NVMe	Support for NVMe 1.4 specification	Added support for all mandatory features in NVMe 1.4 specification. For now, 1.3 specification is still set as default for compatibility reasons.
NVMe	Unique subsystem NQN pre-BlueField	Added a default SNAP subsystem NQN which is unique (and persistent) per BlueField device

Key Features in Version 3.3.0

Protocol	Feature	Description
Virtio-blk	Robustness and Recovery	Added support for SNAP virtio-blk controllers to recover their state post-SNAP process termination and resume the existing communication channels with host drivers
Virtio-	Extended virtio-blk	Extended virtio-blk debug counters command outputs to include additional useful debug parameters, and added a new RPC

Protocol	Feature	Description
blk	RPC support	command to show bdevs connected to a virtio-blk controller
All	Persistent PCIe hot-plug	Added ability to make SNAP hotplugged PCIe functions persistent across SNAP process terminations or even DPU (warm) reboots. Hotplug can also be performed while host is not even alive.
All	Performance optimizations	Extended the flows running in SNAP direct mode (virtio-blk, nvme sgls), and optimized performance for multiple controllers and poll-based (no interrupts) sqs
NVMe	Extended full offload support	Removed limitation for NVMe full-offload mode to be run only on P0

Key Features in Version 3.2.0

Protocol	Feature	Description
All	SNAP statistics	Added dedicated RPC command set to query SNAP controller and get performance and debug statistics
NVMe	NVMe hotplug	Added the option to dynamically attach (hotplug) additional NVMe physical functions to the host server

Key Features in Version 3.1.0

Protocol	Feature	Description
All	SR-IOV support	Added SR-IOV support for both NVMe and virtio-blk protocols. SR-IOV is now able to open 127 PCIe VFs per NVMe/virtio-blk PF configured on the system.
All	Windows support	Added full Windows support to SNAP
Virtio-blk	Static device support	Added support for virtio-blk static devices which can be configured in the firmware configuration stage
Virtio-	UEFI cold boot	Added support for performing UEFI cold boot from virtio-blk devices. This can be achieved by using virtio-blk static devices.

Protocol	Feature	Description
blk		
Virtio-blk	Online backend management	Added support for changing the backends for virtio-blk devices without the need to close the backend controller. Resizing the backends is also supported and can be performed during runtime.
NVMe	BlueField compatibility	Added full support for first generation BlueField devices for both full-offload and non-offload modes
NVMe	Performance optimizations	Dramatically improved performance for NVMe devices, and especially optimized for using SPDK NVMe-oF-RDMA block devices as backends

Key Features in Version 3.0.0

Protocol	Feature	Description
NVMe/virtio-blk	InfiniBand support	In addition to Ethernet, MLNX SNAP supports InfiniBand (starting with BlueField-2). The link type is fully transparent to MLNX SNAP application level – no dedicated configuration changes are needed.
NVMe	NVMe support	Support for NVMe storage interface
virtio-blk	virtio-blk support	Virtio-blk storage interface is introduced on top of BlueField-2. The virtio-blk interface allows customers to extend storage emulation to additional ecosystems.
NVMe/virtio-blk	Multiple devices emulation	Starting with BlueField-2, NVMe/virtio-blk SNAP supports exposing multiple PCIe emulated physical functions (PFs) simultaneously. These PFs can be easily plugged/unplugged (hotplug) to the host PCIe. **Note**

Protocol	Feature	Description
		Hotplug is supported with this release only on virtio-blk.
NVMe/virtio-blk	SPDK-integrated application	NVMe/virtio-blk SNAP operates as an SPDK application, supporting all standard SPDK capabilities (e.g. managing block devices), and extending them with additional NVMe/virtio-blk SNAP-specific abilities (e.g. managing emulated controllers)
NVMe/virtio-blk	Generic SNAP library	NVMe/virtio-blk SNAP is built over a generic library, libsnap, which provides a clean and simple API to control emulation HW/FW capabilities, and translates them into device-specific commands

SNAP Installation

SNAP Installation Process

DPU Image Installation

The BlueField OS image (BFB) includes all needed packages for mlnx_snap to operate: MLNX_OFED, RDMA-CORE libraries, and the supported SPDK version, libsnap and mlnx-snap headers, libraries and binaries.

To see which operating systems are supported, refer to the BlueField Software Documentation under Release Notes → Supported Platforms and Interoperability → Supported Linux Distributions.

RShim must be installed on the host to connect to the NVIDIA® BlueField® DPU. To install RShim, please follow the instructions described in the BlueField Software Documentation → BlueField DPU SW Manual → DPU Operation → DPU Bring-up and Driver Installation → Installing Linux on DPU → Step 1: Set up the RShim Interface.

Use RShim interface from the x86 host machine to install the desired image:

```
BFB=/<path>/latest-bluefield-image.bfb
cat $BFB > /dev/rshim0/boot
```

Optionally, it is possible to connect to the remote console of the DPU and watch the progress of the installation process. Using the `screen` tool, for example:

```
screen /dev/rshim0/console
```

Post-installation Configuration

Firmware Configuration

Refer to Firmware Configuration to confirm your FW configuration matches your SNAP application requirements (SR-IOV support, MSI-X resources, etc).

Network Configuration

Before enabling mlnx_snap or configuring it, users must first verify that uplink ports are configured correctly, and that network connectivity toward the remote target works properly.

By default, two SF interfaces are opened—one over each PF as configured in /etc/mellanox/mlnx-sf.conf—which match RDMA devices mlx5_2 and mlx5_3 respectively. As mentioned, only these interfaces may support RoCE/RDMA transport for the remote storage.

If working with an InfiniBand link, an active InfiniBand port must be made available to allow for InfiniBand support. Once an active IB port is available, users must configure the port RDMA device in the JSON configuration file (see rdma_device under "Configuration File Examples" for mlnx_snap to work on that port).

If working with bonding, it is transparent to MLNX SNAP configuration, and no specific configuration is necessary on NVMe/virtio-blk SNAP level.

Out-of-box Configuration

NVMe/virito-blk SNAP is disabled by default. Once enabled (see section "Firmware Configuration"), the out-of-box configuration of NVMe/virtio-blk SNAP includes a single NVMe controller, backed by a 64MB RAM-based SPDK block device (e.g. RAM drive) in non-offload mode. Out-of-box configuration does not include virtio-blk devices.

A sample configuration file for the out-of-box NVMe controller is located in /etc/mlnx_snap/mlnx_snap.json. For additional information about its values, please see section "Non-offload Mode".

The default initialization command set is described in /etc/mlnx_snap/spdk_rpc_init.conf and /etc/mlnx_snap/snap_rpc_init.conf, as follows:

- spdk_rpc_init.conf

```
bdev_malloc_create 64 512
```

- snap_rpc_init.conf

```
subsystem_nvme_create  Mellanox_NVMe_SNAP "Mellanox NVMe SNAP
Controller"
controller_nvme_create mlx5_0 --subsys_id 0 --pf_id 0
controller_nvme_namespace_attach -c NvmeEmu0pf0 spdk Malloc0
1
```

> **Note**
>
> BlueField out-of-box configuration is slightly different than BlueField-2. For a clean out-of-box experience, /etc/mlnx_snap/snap_rpc_init.conf is a symbolic link pointing to the relevant HW-oriented configuration.

> **Note**
>
> To make any other command set persistent, users may update and modify /etc/mlnx_snap/spdk_rpc_init.conf and /etc/mlnx_snap/snap_rpc_init.conf for according to their needs. Refer to .SNAP Commands v3.5.0 for more information.

SNAP Service Control (systemd)

In order to enable, start, stop, or check the status of mlnx_snap service, run:

```
systemctl {start | stop | status} mlnx_snap
```

Logging

mlnx_snap application output is captured by the SystemD and stored in the internal database. Users are able to get the output from the service console by using the following SystemD commands:

- systemctl status mlnx_snap
- journalctl -u mlnx_snap

SystemD keeps logs in a binary format under the /var/run/log/journal/ directory, which is stored on the tmpfs (i.e. it is not persistent).

SystemD redirects log messages into the "rsyslog" service. Configuration of rsyslog is default to CentOS/RHEL, so users may find all those messages in the /var/log/messages directory.

Note that rsyslog daemon could be configured to send messages to a remote (centralized) syslog server if desired.

SNAP Commands

RPC Command Interface

Remote Procedure Call (RPC) protocol is a very simple protocol defining a few data types and commands. NVMe/virtio-blk SNAP, like other standard SPDK applications, supports JSON-based RPC protocol commands, to control any resources create/delete/query/modify commands easily from CLI.

mlnx_snap supports executing all standard SPDK RPC commands, in addition to an extended SNAP-specific command set. SPDK standard commands are executed by the standard spdk_rpc.py tool, while the SNAP-specific command set extension is executed by an equivalent snap_rpc.py tool.

Full spdk_rpc.py command set documentation can be found in the SPDK official documentation site.

Full snap_rpc.py extended command is detailed later in this chapter.

PCIe Function Management Commands

Emulation Managers Discovery

Emulated PCIe functions are managed through IB devices called "emulation managers". The emulation managers are ordinary IB devices (e.g. mlx5_0, mlx5_1, etc.) with special privileges to also control PCIe communication and device emulations towards the host operating system. Numerous emulation managers may co-exist, each with its own set of capabilities.

The list of emulation managers with their capabilities can be queried using the following command:

```
snap_rpc.py emulation_managers_list
```

Appendix SPDK Configuration includes additional information.

Emulation Devices Configuration (Hotplug)

As mentioned above, each emulation manager holds a list of the emulated PCI functions it controls. The PCI functions may be approached later by either their function index (in emulation manager's list) or their PCIe BDF number (e.g. 88:00.2) as enumerated by the host OS. Some PCIe functions configured at the FW configuration stage are considered "static" (i.e. always present).

In addition, users can dynamically add detachable functions to that list at runtime (and to the host's PCIe devices list accordingly). These functions are called "Hotplugged" PCIe functions.

After a new PCIe function is plugged, it is shown on host PCIe devices list until either explicitly unplugged or the system goes through a cold reboot. Hot-plugged PCIe function will remain persistent even after SNAP process termination.

Some OSs automatically start to communicate with the new function after it is plugged. Some continue to communicate with the function (for a certain time) even after it is signaled to be unplugged. Therefore, users must always keep an open controller (of a matching type) over any existing configured PCIe function (see NVMe Controller Management and Virtio-blk Controller Management for more details).

The following command hotplugs a new PCIe function to the system:

```
snap_rpc.py emulation_device_attach emu_manager {nvme,virtio_blk}
[--id ID] [--vid VID] [--ssid SSID] [--ssvid SSVID] [--revid
REVID] [--class_code CLASS_CODE] [--bdev_type {spdk,none}] [--
bdev BDEV] [--num_queues NUM_QUEUES] [--queue_depth QUEUE_DEPTH]
[--total_vf TOTAL_VF] [--num_msix NUM_MSIX]
```

The following command hot-unplugs a new PCIe function from the system:

```
snap_rpc.py emulation_device_detach <emu_manager>
{nvme,virtio_blk} [-d PCI_BDF / -i PCI_INDEX / --vuid VUID]
```

Mandatory parameters:

- emu_manager – emulation manager
- {nvme,virtio_blk} – device type

Optional arguments:

- --pci_bdf – PCIe BDF identifier
- --pci_index – PCIe index identifier
- --vuid – PCIe VUID identifier
- --force – forcefully remove device (not recommended)

> **Note**
>
> At least one identifier must be provided to describe the PCIe function to be detached.

> **Note**
>
> Once a PCIe function is unplugged from the host system (when calling emulation_device_detach), its controller will be deleted implicitly also.

The following command lists all existing functions (either static or hotplugged):

```
snap_rpc.py emulation_functions_list
```

NVMe Emulation Management Commands

NVMe Subsystem

The NVMe subsystem, as described in the NVMe specification, is a logical entity which encapsulates sets of NVMe backends (or namespaces) and connections (or controllers). NVMe subsystems are extremely useful when working with multiple NVMe controllers, and especially when using NVMe virtual functions.

Each NVMe subsystem is defined by its serial number (SN), model number (MN), and qualified name (NQN). After creation, each subsystem also gets a unique index number.

The following example creates a new NVMe subsystem with a default generated NQN:

```
snap_rpc.py subsystem_nvme_create <serial_number> <model_number>
```

Mandatory parameters:

- serial_number – subsystem serial number
- model_number – subsystem model number

Optional arguments:

- --nqn – subsystem qualified name (auto-generated if not provided)
- --nn – maximal namespace ID allowed in the subsystem (default 0xFFFFFFFE; range 1-0xFFFFFFFE)
- --mnan – maximal number of namespaces allowed in the subsystem (default 1024; range 1-0xFFFFFFFE)

The following command deletes an NVMe subsystem:

```
snap_rpc.py subsystem_nvme_delete <NQN>
```

Where <NQN> is the subsystem's NQN.

The following command lists all NVMe subsystems:

```
snap_rpc.py subsystem_nvme_list
```

NVMe Controller

Each NVMe device (e.g. NVMe PCIe entry) exposed to the host, whether it is a PF or VF, must be backed by NVMe controller, which is responsible for all protocol communication with the host's driver. Every new NVMe controller must also be linked to NVMe subsystem. After creation, NVMe controllers can be addressed using either their name (e.g., "NvmeEmu0pf0") or both their subsystem NQN and controller ID.

The following command opens a new NVMe controller:

```
snap_rpc.py controller_nvme_create mlx5_0 [--pf_id ID / --pci_bdf
/ --vuid VUID] [--subsys_id ID / --nqn NQN]
```

Mandatory parameters:

- emu_manager – emulation manager

Optional parameters:

- --vf_id VF_ID – PCIe VF index to start emulation on (if the controller is destined to be opened on a VF). --pf_id must also be set for the command to take effect.
- --conf – JSON configuration file path to be used to provide an extended set of configuration parameters. Full information concerning the different parameters of the configuration file can be found under appendix "JSON File Format".
- --nr_io_queues – I/O queues maximal number (default 32, range 0-32)
- --mdts – maximum data transfer size (default 4, range 1-6)
- --max_namespaces – maximum number of namespaces for this controller (default 1024, range 1-0xFFFFFFFE)
- --quirks – bitmask to enable specific NVMe driver quirks to work with non-NVMe spec compliant drivers. For more information, refer to appendix "JSON File Format".

- --mem {static,pool} – use memory from a global pool or from dedicated buffers. See "Mem-pool" for more information.

The following command deletes an existing NVMe controller:

```
snap_rpc.py controller_nvme_delete [--name NAME / --subnqn SUBNQN
--cntlid ID / --vuid VUID]
```

Optional arguments:

- -c NAME, --name NAME – controller Name. Must be set if --nqn and --cntlid are not set.
- -n SUBNQN, --subnqn SUBNQN – NVMe subsystem (NQN). Must be set if --name is not set.
- -i CNTLID, --cntlid CNTLID – controller identifier in NVMe subsystem. Must be set if --name is not set.

The following command lists all NVMe controllers:

```
snap_rpc.py controller_list --type nvme
```

Optional arguments:

- -t {nvme,virtio_blk,virtio_net}, --type {nvme,virtio_blk,virtio_net} – controller type

NVMe Backend (Namespace)

NVMe Namespaces are the representors of a continuous range of LBAs in the local/remote storage device (previously configured in "Backend Configuration" section). Each Namespace must be linked to a controller and have a unique identifier (NSID) across the entire NVMe subsystem (e.g. 2 namespaces cannot share the same NSID, even though linked to different controllers).

SNAP application uses SPDK block device framework as backends for its NVMe namespaces. Therefore, they should be configured in advance. For more information

about SPDK block devices, see SPDK BDEV documentation and appendix "SPDK Configuration".

The following command attaches a new namespace to an existing NVMe controller:

```
snap_rpc.py controller_nvme_namespace_attach [--ctrl CTRL / --
subnqn SUBNQN --cntlid ID] <bdev_type> <bdev> <nsid>
```

Mandatory parameters:

- bdev_type – block device type
- bdev – block device to use as backend
- nsid – namespace ID

Optional parameters:

- -c CTRL, --ctrl CTRL – controller name. Must be set if --nqn and --cntlid are not set.
- -n SUBNQN, --subnqn SUBNQN – NVMe subsystem (NQN). Must be set if --ctrl is not set.
- -i CNTLID, --cntlid CNTLID – controller identifier in NVMe subsystem. Must be set if --ctrl is not set.
- -q QN, --qn QN – QN of remote target which provides this namespace
- -p PROTOCOL, --protocol PROTOCOL – protocol used
- -g NGUID, --nguid NGUID – namespace globally unique identifier
- -e EUI64, --eui64 EUI64 – namespace EUI-64 identifier
- -u UUID, --uuid UUID – namespace UUID

Note

In full-offload mode, backends are acquired from the remote storage automatically and no manual configuration is required.

The following command detaches a namespace from a controller:

```
snap_rpc.py controller_nvme_namespace_detach [--ctrl CTRL / --
subnqn SUBNQN --cntlid ID] <nsid>
```

Mandatory parameters:

- nsid – namespace ID

Optional parameters:

- -c CTRL, --ctrl CTRL – controller name. Must be set if --nqn and --cntlid are not set.
- -n SUBNQN, --subnqn SUBNQN – NVMe subsystem (NQN). Must be set if --ctrl is not set.
- -i CNTLID, --cntlid CNTLID – controller identifier in NVMe subsystem. Must be set if --ctrl is not set.

The following command lists a namespace of a controller:

```
snap_rpc.py controller_nvme_namespace_list [--ctrl CTRL / --
subnqn SUBNQN --cntlid ID]
```

Optional parameters:

- -c CTRL, --ctrl CTRL – controller name. Must be set if --nqn and --cntlid are not set.
- -n SUBNQN, --subnqn SUBNQN – NVMe subsystem (NQN). Must be set if --ctrl is not set.
- -i CNTLID, --cntlid CNTLID – controller identifier in NVMe subsystem. Must be set if --ctrl is not set.

VirtIO-blk Emulation Management Commands

VirtIO-blk Controller

Each virtio-blk device (e.g. virtio-blk PCI entry) exposed to the host, whether it is PF or VF, must be backed by a virtio-blk controller. Virtio-blk is considered a limited storage protocol (compared to NVMe, for instance).

Due to protocol limitations:

- Trying to use a virtio-blk device (e.g. probe virtio-blk driver on host) without an already functioning virtio-blk controller, may cause the host server to hang until such controller is opened successfully (no timeout mechanism exists)
- Upon creation of a virtio-blk controller, a backend device must already exist

The following command creates a new virtio-blk controller:

```
snap_rpc.py controller_virtio_blk_create <emu_manager> [-d
PCI_BDF / --pf_id PF_ID / --vuid VUID]
```

Mandatory parameters:

- emu_manager – emulation manager

Optional parameters:

- --vf_id – PCIe VF index to start emulation on, if controller is destined to be opened on VF
- --num_queues – number of queues (default 64, range 2-256)
- --queue_depth – queue depth (default 128, range 1-256)
- --size_max — maximal SGE data transfer size (default 4096, range 1 – MAX_UINT16). See VirtIO specification for more information.
- --seg_max – maximal SGE list length (default 1, range 1-queue_depth). See VirtIO specification for more information.

- --bdev_type – block device type (spdk/none). Note that opening a controller with none backend means to open it with a backend of size of 0.
- --bdev – SPDK block device to use as backend
- --serial – serial number for the controller
- --force_in_order - force I/O in-order completions. Note that this value is required to ensure future virtio-blk controllers always successfully recover after an application crash.
- --suspend – create controller is in a SUSPENDED state (must be explicitly resumed later)
- --mem {static,pool} – use memory from a global pool or from dedicated buffers. See ".Advanced Features v3.5.0" for more information.

The following command deletes a virtio-blk controller:

```
snap_rpc.py controller_virtio_blk_delete [-c NAME / --vuid VUID]
```

Mandatory arguments:

- name – controller name

Optional arguments:

- --f, --force – force controller deletion

The following command lists all virtio-blk controllers:

```
snap_rpc.py controller_list --type virtio_blk
```

Optional arguments:

- -t {nvme,virtio_blk,virtio_net}, --type {nvme,virtio_blk,virtio_net} – controller type

Virtio-blk controllers can also be suspended and resumed. While suspended, the controller stops receiving new requests from the host driver and only finishes handling of requests

already in flight (without prompting any IO errors back to the driver).

The following command suspends/resumes virtio-blk controller:

```
snap_rpc.py controller_virtio_blk_suspend <name>
snap_rpc.py controller_virtio_blk_resume <name>
```

Mandatory arguments:

- name – controller name

VirtIO-blk Backend Management

Like NVMe, VirtIO BLK also uses SPDK block devices framework as its backend devices, but since virtio-blk is a limited storage protocol as opposed to NVMe, whose backend management abilities are limited as well:

- Virtio-blk protocol supports only one backend device
- Virtio-blk protocol does not support administration commands to add backends, thus all backend attributes are communicated to the host virtio-blk driver over PCIe BAR and must be accessible during driver probing. For that reason, backends can only be changed when PCIe function is not in use by any host storage driver.

For these reasons, when the host driver is active, all backend management operations must occur only when the controller is in suspended state.

The following command attaches a new backend to a controller:

```
snap_rpc.py controller_virtio_blk_bdev_attach <ctrl_name> {spdk}
<bdev_name>
```

Mandatory arguments:

- name – controller name

- {spdk} – block device type
- bdev – block device to use as backend

Optional arguments:

- --size_max – maximal SGE data transfer size (no hard limit). See VirtIO specification for more information.
- --seg_max – maximal SGE list length (no hard limit). See VirtIO specification for more information.

The following command detaches a backend from a controller:

```
snap_rpc.py controller_virtio_blk_bdev_detach <ctrl_name>
```

Mandatory arguments:

- ctrl_name – controller name

> **Note**
>
> Destruction of SPDK block devices using SPDK block devices' API is considered a controller_virtio_blk_bdev_detach and is bound to the same limitations.

The following command lists a backend detail of a controller:

```
snap_rpc.py controller_virtio_blk_bdev_list <name>
```

Mandatory arguments:

- name – controller name

Debug and Statistics

BlueField and virito-blk SNAP provide a set of commands which help customers retrieve performance and debug statistics about the opened emulated devices. The statistics are provided at the SNAP controller level (whether for NVMe or Virtio-blk).

IO Statistics

The following commands are available to measure how many successful/failed IO operations were executed by the controller.

These commands have minimal effect on BlueField SNAP performance and can therefore be used to sample statistics while the controller performs high bandwidth IO operations.

```
snap_rpc.py controller_nvme_get_iostat [-c CTRL_NAME]
snap_rpc.py controller_virtio_blk_get_iostat [-c CTRL_NAME]
```

> **Note**
>
> These commands have minimal effect on BlueField SNAP performance and can therefore be used to sample statistics while the controller performs high-bandwidth IO operations.

Mandatory arguments:

- CTRL_NAME – controller name

NVMe/Virtio IO statistics:

- read_ios – number of read commands handled
- completed_read_ios – number of read commands completed successfully
- err_read_ios – number of read commands completed with error

- write_ios – number of write commands handled
- completed_write_ios – number of write commands completed successfully
- err_write_ios – number of write commands completed with error
- flush_ios – number of flush commands handled
- completed_flush_ios – number of flush commands completed successfully
- err_flush_ios – number of flush commands completed with error

Virtio IO specific statistics:

- fatal_ios – number of commands dropped and never completed
- outstanding_in_ios – number of outstanding IOs at a given moment
- outstanding_in_bdev_ios – number of outstanding IOs at a given moment, pending backend handling
- outstanding_to_host_ios – number of outstanding IOs at a given moment, pending DMA handling

Debug Statistics

The following commands are available to examine the controller and queues with more detailed status and information.

When queried frequently, these commands may impact performance and should therefore be called for debug purposes only.

```
snap_rpc.py controller_nvme_get_debugstat [-c NAME]
snap_rpc.py controller_virtio_blk_get_debugstat [-c NAME]
```

Initialization Scripts

The default initialization scripts /etc/mlnx_snap/spdk_rpc_init.conf and /etc/mlnx_snap/snap_rpc_init.conf allow users to control the startup configuration.

These scripts, which are used for the out-of-box configuration, may be modified by the user to control the SNAP initialization :

- The spdk_rpc_init.conf may be modified with the SPDK commands listed under the SPDK Configuration appendix.
- The snap_rpc_init.conf may be modified with the snap_rpc commands described throughout this chapter (SNAP Commands).

Advanced Features

Performance Optimization

> **Note**
>
> Tuning MLNX SNAP for the best performance may require additional resources from the system (CPU, memory) and may affect SNAP controller scalability.

Increasing Number of Used Arm Cores

By default, MLNX SNAP uses 4 Arm cores, with core mask 0xF0. The core mask is configurable in /etc/default/mlnx_snap (parameter CPU_MASK) for best performance (i.e., CPU_MASK=0xFF).

> **Note**
>
> As SNAP is an SPDK based application, it constantly polls the CPU and therefore occupies 100% of the CPU it runs on.

Disabling Mem-pool

When mem-pool is enabled, that reduces the memory footprint but decreases the overall performance.

To configure the controller to not use mem-pool, set MEM_POOL_SIZE=0 in /etc/default/mlnx_snap.

See section "Mem-pool" for more information.

Maximizing Single IO Transfer Data Payload

Increasing datapath staging buffer sizes improves performance for larger block sizes (>4K):

- For NVMe, this can be controlled by increasing the MDTS value either in the JSON file or the RPC parameter. For more information regarding MDTS, refer to the NVMe specification. The default value is 4 (64K buffer), and the maximum value is 6 (256K buffer).

- For virtio-blk, this can be controlled using the seg_max and size_max RPC parameters. For more information regarding these parameters, refer to the VirtIO-blk specification. No hard-maximum limit exists.

Increasing Emulation Manager MTU

The default MTU for the emulation manager network interface is 1500. Increasing MTU to over 4K on the emulation manager (e.g., MTU=4200) also enables the SNAP application to transfer larger amount of data in a single Host→DPU memory transactions, which may improve performance.

Optimizing Number of Queues and MSIX Vector (virtio-blk only)

SNAP emulated queues are spread evenly across all configured PFs (static and dynamic) and defined VFs per PF (whether functions are being used or not). This means that the larger the total number of functions SNAP is configured with (either PFs or VFs), the less queues and MSIX resources each function will be assigned which would affect its performance accordingly. Therefore, it is recommended to configure in Firmware Configuration the minimal number of PFs and VFs per PF desired for that specific system.

Another consideration is matching between MSIX vector size and the desired number of queues. The standard virtio-blk kernel driver uses an MSIX vector to get events on both control and data paths. When possible, it assigns exclusive MSIX for each virtqueue (e.g.,

per CPU core) and reserves an additional MSIX for configuration changes. If not possible, it uses a single MSIX for all virtqueues. Therefore, to ensure best performance with virtio-blk devices, the condition VIRTIO_BLK_EMULATION_NUM_MSIX > virtio_blk_controller.num_queues must be applied.

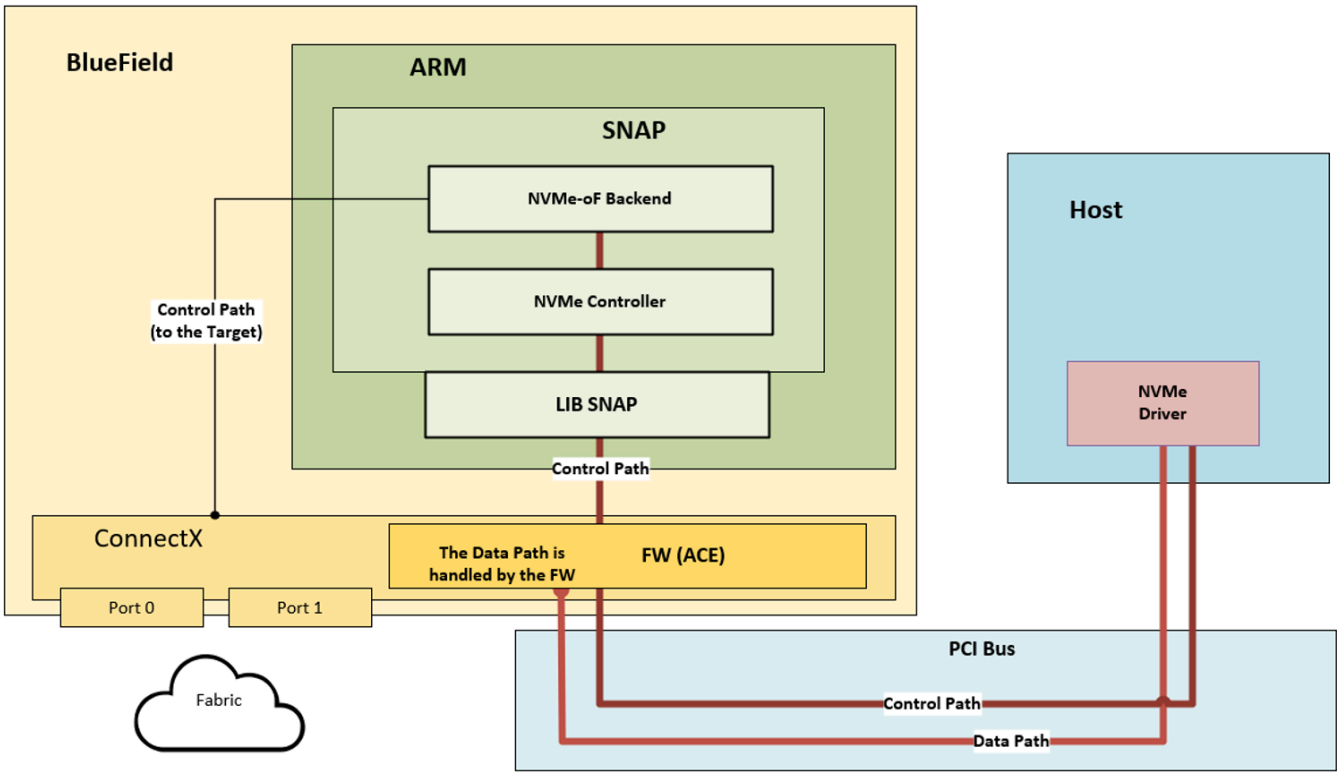
> **Note**
>
> The total number of MSIXs is limited on BlueField-2 cards, so MSIX reservation considerations may apply when running with multiple devices. For more information, refer to this FAQ.

NVMe-RDMA Full Offload Mode

The NVMe-RDMA full offload mode allows reducing the Arm cores CPU cost by offloading the data-path directly to the FW/HW. If the user needs to control the data plane or the backend this mode does not allow it.

In full offload mode the control plane is handled at the SW level, while the data plane is being handled at the FW level and requires no SW interaction. For that reason, the user has no control over the backend devices. Thus they are detected automatically and no namespace management commands are required.

The NVMe-RDMA architecture:



> **Note**
>
> In this mode, a remote target parameter must be provided using a JSON configuration file (a JSON file example can be found in /etc/mlnx_snap/mlnx_snap_offload.json.example) and the NVMe controller can detect and connect to the relevant backends by itself.

As the SNAP application does not participate in the datapath and needs fewer resources, it is recommended to reduce CPU_MASK to a single core (i.e., CPU_MASK=0x80). Refer to "Increasing Number of Used Arm Cores" for CPU_MASK configuration.

After configuration is done, users must create the NVMe subsystem and (offloaded) controller. Note that snap_rpc.py controller_nvme_namespace_attach is not required and --rdma_device mlx5_2 is provided to mark the relevant RDMA interface for the connection.

The following example creates an NVMe full-offload controller:

```
# snap_rpc.py subsystem_nvme_create  "Mellanox_NVMe_SNAP"
"Mellanox NVMe SNAP Controller"
# snap_rpc.py controller_nvme_create mlx5_0 --subsys_id 0 --
pci_bdf 88:00.2 --nr_io_queues 32 --mdts 4 -c
/etc/mlnx_snap/mlnx_snap.json --rdma_device mlx5_2
```

This is the matching JSON file example:

```
{
  "ctrl": {
    "offload": true,
  },
  "backends": [
    {
      "type": "nvmf_rdma",
      "name": "testsubsystem",
      "paths": [
        {
          "addr": "1.1.1.1",
          "port": 4420,
          "ka_timeout_ms": 15000,
          "hostnqn": "r-nvmx03"
        }
      ]
    }
  ]
}
```

Note

Full offload mode requires that the provided RDMA device (given in `--rdma_device` parameter) supports RoCE transport (typically SF

interfaces). Full offload mode for virtio-blk is not supported.

Note

The discovered namespace ID may be remapped to get another ID when exposed to the host in order to comply with firmware limitations.

SR-IOV

SR-IOV configuration depends on the kernel version:

- Optimal configuration may be achieved with a new kernel in which the sriov_drivers_autoprobe sysfs entry exists in /sys/bus/pci/devices/<BDF>/
- Otherwise, the minimal requirement may be met if the sriov_totalvfs sysfs entry exists in /sys/bus/pci/devices/<BDF>/

SR-IOV configuration needs to be done on both the host and DPU side, marked in the following example as [HOST] and [ARM] respectively. This example assumes that there is 1 VF on static virtio-blk PF 86:00.3 (NVMe flow is similar), and that a "Malloc0" SPDK BDEV exists.

Optimal Configuration

```
[ARM] snap_rpc.py controller_virtio_blk_create mlx5_0 -d 86:00.3
--bdev_type none
[HOST] modprobe -v virtio-pci && modprobe -v virtio-blk
[HOST] echo 0 >
/sys/bus/pci/devices/0000:86:00.3/sriov_drivers_autoprobe
[HOST] echo 1 > /sys/bus/pci/devices/0000:86:00.3/sriov_numvfs
[ARM] snap_rpc.py controller_virtio_blk_create mlx5_0 --pf_id 0 -
-vf_id 0 --bdev_type spdk --bdev Malloc0
```

```
\* Continue by binding the VF PCIe function to the desired VM. *\
```

> **Note**
>
> After configuration is finished, no disk is expected to be exposed in the hypervisor. The disk only appears in the VM after the PCIe VF is assigned to it using the virtualization manager. If users want to use the device from the hypervisor, they simply need to bind the PCIe VF manually.

Minimal Requirement

```
[ARM] snap_rpc.py controller_virtio_blk_create mlx5_0 -d 86:00.3
--bdev_type none
[HOST] modprobe -v virtio-pci && modprobe -v virtio-blk
[HOST] echo 1 > /sys/bus/pci/devices/0000:86:00.3/sriov_numvfs
\* the host now hangs until configuration is performed on the DPU
side *\ [ARM] snap_rpc.py controller_virtio_blk_create mlx5_0 --
pf_id 0 --vf_id 0 --bdev_type spdk --bdev Malloc0
\* Host is now released *\
\* Continue by binding the VF PCIe function to the desired VM. *\
```

> **Note**
>
> Hotplug PFs do not support SR-IOV.

Info

It is recommended to add pci=assign-busses to the boot command line when creating more than 127 VFs.

00000191-5389-d31e-adf1-53afd40e0003

Zero Copy (SNAP-direct)

Note

Zero-copy is supported on SPDK 21.07 and higher.

The SNAP-direct feature allows SNAP applications to transfer data directly from the host memory to remote storage without using any staging buffer inside the DPU.

SNAP enables the feature according to the SPDK BDEV configuration only when working against an SPDK NVMe-oF RDMA block device.

To configure the controller to use Zero Copy, set the following in /etc/default/mlnx_snap:

- For virtio-blk:

```
VIRTIO_BLK_SNAP_ZCOPY=1
```

- For NVMe:

```
NVME_SNAP_ZCOPY=1
```

NVMe/TCP Zero Copy

NVMe/TCP Zero Copy is implemented as a custom NVDA_TCP transport in SPDK NVMe initiator and it is based on a new XLIO socket layer implementation.

The implementation is different for Tx and Rx:

- The NVMe/TCP Tx Zero Copy is similar between RDMA and TCP in that the data is sent from the host memory directly to the wire without an intermediate copy to Arm memory
- The NVMe/TCP Rx Zero Copy allows achieving partial zero copy on the Rx flow by eliminating copy from socket buffers (XLIO) to application buffers (SNAP). But data still must be DMA'ed from Arm to host memory.

To enable NVMe/TCP Zero Copy, use SPDK v22.05.nvda --with-xlio.

> **Note**
>
> For more information about XLIO including limitations and bug fixes, refer to the NVIDIA Accelerated IO (XLIO) Documentation.

To configure the controller to use NVMe/TCP Zero Copy, set the following in /etc/default/mlnx_snap:

```
EXTRA_ARGS="-u —mem-size 1200 —wait-for-rpc"
NVME_SNAP_TCP_RX_ZCOPY=1
SPDK_XLIO_PATH=/usr/lib/libxlio.so
MIN_HUGEMEM=4G
```

To connect using NVDA_TCP transport:

- If /etc/mlnx_snap/spdk_rpc_init.conf is not being used, add the following at the start of the file in the given order:

```
sock_set_default_impl -i xlio
framework_start_init
```

When the mlnx_snap service is started, run the following command:

```
[ARM] spdk_rpc.py bdev_nvme_attach_controller -b <NAME> -t
NVDA_TCP -f ipv4 -a <IP> -s <PORT> -n <SUBNQN>
```

- If /etc/mlnx_snap/spdk_rpc_init.conf is not being used, once the service is started, run the following commands in the given order:

```
[ARM] spdk_rpc.py sock_set_default_impl -i xlio
[ARM] spdk_rpc.py framework_start_init
[ARM] spdk_rpc.py bdev_nvme_attach_controller -b <NAME> -t
NVDA_TCP -f ipv4 -a <IP> -s <PORT> -n <SUBNQN>
```

Note

NVDA_TCP transport is fully interoperable with other implementations based on the NVMe/TCP specifications.

Note

NVDA_TCP limitations:

- SPDK multipath is not supported
- NVMe/TCP data digest is not supported

- SR-IOV is not supported

Robustness and Recovery

As SNAP is a standard user application running on the DPU OS, it is vulnerable to system interferences, like closing SNAP application gracefully (i.e., stopping mlnx_snap service), killing SNAP process brutally (i.e., running `kill -9`), or even performing full OS restart to DPU. If there are exposed devices already in use by host drivers when any of these interferences occur, that may cause the host drivers/application to malfunction.

To avoid such scenarios, the SNAP application supports a "Robustness and Recovery" option. So, if the SNAP application gets interrupted for any reason, the next instance of the SNAP application will be able to resume where the previous instance left off.

This functionality can be enabled under the following conditions:

- Only virtio-blk devices are used (this feature is currently not supported for NVMe protocol)
- By default, SNAP application is programmed to survive any kind of "graceful" termination, including controller deletion, service restart, and even (graceful) Arm reboot. If extended protection against brutal termination is required, such as sending SIGKILL to SNAP process or performing brutal Arm shutdown, the `--force_in_order` flag must be added to the `snap_rpc.py controller_virtio_blk_create` command.

  > **Note**
  >
  > The `force_in_order` flag may impact performance if working with remote targets as it may cause high rates of out-of-order completions or if different queues are served in different rates.

- It is the user's responsibility to open the recovered virtio-blk controller with the exact same characteristics as the interrupted virtio-blk controller (same remote storage device, same BAR parameters, etc.)

Mem-pool

By default, SNAP application pre-allocates all required memory buffers in advance.

A great amount of allocated memory may be required when using:

- Large number of controllers (as with SR-IOV)
- Large number of queues per controller
- High queue-depth
- Large mdfs (for NVMe) or seg_max and size_max (for virtio-blk)

To reduce the memory footprint of the application, users may choose to use mem-pool (a shared memory buffer pool) instead. However, using mem-pool may decrease overall performance.

To configure the controller to use mem-pool rather than private ones:

1. In /etc/default/mlnx_snap, set the parameter MEM_POOL_SIZE to a non-zero value. This parameter accepts K/M/G notations (e.g., MEM_POOL_SIZE=100M). If K/M/G notation is not specified, the value defaults to bytes.
2. Users must choose the right value for their needs—a value too small may cause longer starvations, while a value too large consumes more memory. As a rule of thumb, typical usage may choose to set it as a minimum (num_devices*4MB, 512MB).
3. Upon controller creation, add the option --mempool. For example:

```
snap_rpc.py controller_nvme_create mlx5_0 –subsys_id 0 –pf_id
0 --mem pool
```

> **Note**
>
> The per controller mem-pool configuration is independent from all others. Users can set some controllers to work with mem-pool and other controllers to work without it.

Virtio-blk Transitional Device (0.95)

SNAP supports virtio-blk transitional devices. Virtio transitional devices refer to devices supporting drivers conforming to modern specification and legacy drivers (conforming to legacy 0.95 specifications).

To configure virtio-blk PCIe functions to be transitional devices, special firmware configuration parameters must be applied:

- VIRTIO_BLK_EMULATION_PF_PCI_LAYOUT (0: MODERN / 1: TRANSITIONAL) – configures transitional device support for PFs

  > **Note**
  >
  > This parameter is currently not supported.

- VIRTIO_BLK_EMULATION_VF_PCI_LAYOUT (0: MODERN / 1: TRANSITIONAL) – configures transitional device support for underlying VFs

  > **Note**
  >
  > This parameter is currently not supported.

- VIRTIO_EMULATION_HOTPLUG_TRANS (True/False) – configures transitional device support for hot-plugged virtio-blk devices

To use virtio-blk transitional devices, Linux boot parameters must be set on the host:

- If the kernel version is older than 5.1, set the following Linux boot parameter on the host OS:

  ```
  intel_iommu=off
  ```

- If virtio_pci is built-in from host OS, set the following Linux boot parameter:

```
virtio_pci.force_legacy=1
```

- If virtio_pci is a kernel module rather than built-in from host OS, use force legacy to load the module:

```
modprobe -rv virtio_pci
modprobe -v virtio_pci force_legacy=1
```

For hot-plugged functions, additional configuration must be applied during SNAP hotplug operation:

```
# snap_rpc.py emulation_device_attach mlx5_0 virtio_blk --
transitional_device --bdev_type spdk --bdev BDEV
```

Virtio-blk Live Migration

Live migration is a standard process supported by QEMU which allows system administrators to pass devices between virtual machines in a live running system. For more information, refer to QEMU VFIO device Migration documentation.

Live migration is supported for SNAP virtio-blk devices. It can be activated using a driver with proper support (e.g., NVIDIA's proprietary VDPA-based Live Migration Solution). For more info, refer to TBD.

Note

If the physical function (PF) has been removed, for instance, using vDPA provisioning virtio-blk PF with the command:

```
python ./app/vfe-vdpa/vhostmgmt mgmtpf -a
0000:af:00.3
```

It is advisable to confirm and restore the presence of controllers in SNAP before attempting to re-add them using the command:

```
python dpdk-vhost-vfe/app/vfe-vdpa/vhostmgmt vf -
v /tmp/sock-blk-0 -a 0000:59:04.5
```

Appendixes

- Appendix – Live Upgrade
- Appendix – Host OS Configuration
- Appendix – SPDK Configuration
- Appendix – Firmware Configuration
- Appendix – JSON File Format
- Appendix – Frequently Asked Questions

Appendix – Live Upgrade

Warning

The following procedure is designed for live deployment of small software bug fixes or modifications made in the SNAP application. Using this procedure for other purposes (e.g., bumping SNAP service to a new version on top of an older BFB image) may cause SNAP to malfunction.

To live upgrade SNAP, 2 SNAP processes must be opened in parallel.

Note

All system resources (e.g., hugepages, memory) must be sufficient to temporarily support 2 SNAP application instances operating in parallel during the upgrade procedure.

Passing virtio-blk Controller's Management Between SNAP Processes

1. Open 2 SNAP processes simultaneously on the Arm.

Note

This requires changing the SPDK RPC server path.

Info

For lower downtime, it is highly recommended to run each process on a different CPU mask.

For SNAP Process 1, run:

```
./mlnx_snap_emu -m 0xf0 -r /var/tmp/spdk.sock1
```

For SNAP Process 2, run:

```
./mlnx_snap_emu -m 0x0f -r /var/tmp/spdk.sock2
```

2. Connect to the same bdev with both processes (i.e., with Malloc device).

   For SNAP Process 1, run:

   ```
   spdk_rpc.py -s /var/tmp/spdk.sock1 bdev_malloc_create -b
   Malloc1 1024 512
   ```

   For SNAP Process 2, run:

   ```
   spdk_rpc.py -s /var/tmp/spdk.sock2 bdev_malloc_create -b
   Malloc1 1024 512
   ```

3. Open a virtio-blk controller on the SNAP Process 1:

   ```
   snap_rpc.py -s /var/tmp/spdk.sock1
   controller_virtio_blk_create mlx5_0 --pf_id 0 --bdev_type
   spdk --bdev Malloc1 --num_queues 16
   ```

4. Load virtio-blk driver on the host side and start using it.

5. Delete the virtio-blk controller instance from SNAP Process 1 and immediately open a virtio-blk controller on SNAP Process 2:

   ```
   snap_rpc.py -s /var/tmp/spdk.sock1
   controller_virtio_blk_delete VblkEmu0pf0 --force &&
   snap_rpc.py -s /var/tmp/spdk.sock2
   controller_virtio_blk_create mlx5_0 --pf_id 0 --bdev_type
   spdk --bdev Malloc1  --num_queues 16
   ```

Full "Live Upgrade" Procedure

Assuming there exists a fully configured SNAP service is already running on the system:

1. Create a local copy of SNAP binary file (e.g., under /tmp folder):

```
cp /usr/bin/mlnx_snap_emu /tmp/
```

2. For all active virtio-blk controllers, follow management passing procedure as described in section "Passing virtio-blk Controller's Management Between SNAP Processes".

3. Stop original SNAP service.

```
systemctl stop mlnx_snap
```

4. Upgrade SNAP service.

   - If installed from binary, use Linux official installation framework (apt/yum)

   - If installed from sources, follow the same installation process as done originally

5. Repeat management passing procedure, this time to move back control from the local copy into the official (updated) version of SNAP service.

Appendix – Host OS Configuration

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

- To use SR-IOV, intel_iommu=on iommu=ptmust be added

- To use PCIe hotplug, pci=realloc must be added

- When using SR-IOV, pci=assign-busses must be added

To view boot parameter values, use the command cat /proc/cmdline.

Appendix – SPDK Configuration

SPDK backend (BDEV) management commands:

```
spdk_rpc.py bdev_nvme_attach_controller -b <name> -t rdma -a <ip>
-f ipv4 -s <port> -n <nqn>
spdk_rpc.py bdev_nvme_detach_controller <name>
spdk_rpc.py bdev_null_create <name> <size_mb> <blk_size>
spdk_rpc.py bdev_null_delete <name>
spdk_rpc.py bdev_aio_create <filepath> <name> <blk_size>
spdk_rpc.py bdev_aio_delete <name>
```

For more information, please refer to SPDK BDEV documentation.

Appendix – Firmware Configuration

Before configuring mlnx_snap, the user must ensure all FW configuration requirements are met. By default, mlnx_snap is disabled, and needs to be enabled by running both common mlnx-snap configuration, and additional protocol-specific configuration depending on the expected usage of the application (e.g. Hotplug, SR-IOV, UEFI boot, etc).

After all configuration is finished, power-cycling the host is required for these 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/mt41686_pciconf0 -e query
```

Basic Configuration

1. (Optional) Reset all previous configuration.

```
[dpu] mst start
[dpu] mlxconfig -d /dev/mst/mt41686_pciconf0 reset
```

> **Note**
>
> This will return your product to its default configurations. Do this only if you were not able to get SNAP to work.

2. Set general basic parameters.

   On BlueField-2:

```
[dpu] mlxconfig -d /dev/mst/mt41686_pciconf0 s
INTERNAL_CPU_MODEL=1
```

   On BlueField:

```
[dpu] sudo mlxconfig -d /dev/mst/mt41682_pciconf0 s
INTERNAL_CPU_MODEL=1 PF_BAR2_ENABLE=1 PF_BAR2_SIZE=1
```

3. When using RDMA/RoCE transport, additional parameters must be configured:

```
[dpu] mlxconfig -d /dev/mst/mt41686_pciconf0 s
PER_PF_NUM_SF=1
```

```
[dpu] mlxconfig -d /dev/mst/mt41686_pciconf0 s
PF_SF_BAR_SIZE=8 PF_TOTAL_SF=2
[dpu] mlxconfig -d /dev/mst/mt41686_pciconf0.1 s
PF_SF_BAR_SIZE=8 PF_TOTAL_SF=2
```

System Configuration Parameters

Parameter	Description	Possible Values	Comments
SRIOV_EN	Enable SR-IOV	0/1	
NUM_OF_VFS	Number of VFs per emulated PF	[0-127]	
NUM_PF_MSIX	Number of MSIX assigned to emulated PF	[0-63]	
NUM_VF_MSIX	Number of MSIX assigned to emulated VF	[0-63]	
PCI_SWITCH_EMULATION_ENABLE	Enable PCI switch for emulated PFs	0/1	
PCI_SWITCH_EMULATION_NUM_PORT	Max number of emulated PFs	[0-32]	Single port is reserved for all static PFs

Note

SRIOV_EN is valid only for static PFs

NVMe Configuration

Parameter	Description	Possible Values	Comments
NVME_EMULATION_ENABLE	Enable NVMe device emulation	0/1	

Parameter	Description	Possible Values	Comments
NVME_EMULATION_NUM_PF	Number of static emulated nvme PFs	[0-2]	
NVME_EMULATION_NUM_MSIX	Number of MSIX assigned to emulated NVMe PF/VF	[0-63]	
NVME_EMULATION_NUM_VF	Number of VFs per emulated NVMe PF	[0-127]	If not 0, overrides NUM_OF_VFS; valid only when SRIOV_EN=1
EXP_ROM_NVME_UEFI_x86_ENABLE	Enable NVMe UEFI exprom driver	0/1	Used for UEFI boot process

VirtIO-blk Configuration

Warning

Due to virtio-blk protocol limitations, using bad configuration while working with static virtio-blk 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 Family Software Documentation" for more details.
- Add the following line to /etc/mlnx_snap/snap_rpc_init.conf:

```
controller_virtio_blk_create mlx5_0 --pf_id 0
--bdev_type none
```

For more information, please refer to section “VirtIO-blk Controller Management”.

Parameter	Description	Possible Values	Comments
VIRTIO_BLK_EMULATION_ENABLE	Enable virtio-blk device emulation	0/1	
VIRTIO_BLK_EMULATION_NUM_PF	Number of static emulated virtio-blk PFs	[0-2]	See WARNING above
VIRTIO_BLK_EMULATION_NUM_MSIX	Number of MSIX assigned to emulated virtio-blk PF/VF	[0-63]	
VIRTIO_BLK_EMULATION_NUM_VF	Number of VFs per emulated virtio-blk PF	[0-127]	If not 0, overrides NUM_OF_VFS; valid only when SRIOV_EN=1
EXP_ROM_VIRTIO_BLK_UEFI_x86_ENABLE	Enable virtio-blk UEFI exprom driver	0/1	Used for UEFI boot process

Appendix – JSON File Format

This section is relevant only for the following cases:

- Using legacy mode in which the user prefers to not use the recommended SNAP commands, but to use the JSON file format
- NVMe-RDMA full offload mode in which the configuration is only possible with the JSON file format

The configuration parameters are divided into two categories: Controller and backends.

Configuration File Examples

Legacy Mode

Note

For the non-full offload mode, it is recommended to use the SNAP RPC commands (described in .SNAP Commands v3.5.0) and not the legacy mode of the JSON file format described in this section.

```
{
  "ctrl": {
    "func_num": 0,
    "rdma_device": "mlx5_2",
    "sqes": 0x6,
    "cqes": 0x4,
    "cq_period": 3,
    "cq_max_count": 6,
    "nr_io_queues": 32,
    "mn": "Mellanox BlueField NVMe SNAP Controller",
    "sn": "MNC12",
    "mdts": 4,
    "oncs": 0,
    "offload": false,
    "max_namespaces": 1024,
    "quirks": 0x0,
    "version": "1.3.0"
  },
  "backends": [
    {
      "type": "spdk_bdev",
      "paths": [
        {
        }
      ]
    }
  ]
}
```

NVMe-RDMA Full Offload Mode

> **Note**
>
> For NVMe-RDMA full offload mode, users can only use the JSON file format (and not the SNAP RPC commands).

```
{
  "ctrl": {
    "func_num": 0,
    "rdma_device": "mlx5_2",
    "sqes": 0x6,
    "cqes": 0x4,
    "cq_period": 3,
    "cq_max_count": 6,
    "nr_io_queues": 32,
    "mn": "Mellanox BlueField NVMe SNAP Controller",
    "sn": "MNC12",
    "mdts": 4,
    "oncs": 0,
    "offload": true,
    "max_namespaces": 1024,
    "quirks": 0x0
    "version": "1.3.0"
  },
  "backends": [
    {
      "type": "nvmf_rdma",
      "name": "testsubsystem",
      "paths": [
```

```
        {
          "addr": "1.1.1.1",
          "port": 4420,
          "ka_timeout_ms": 15000,
          "hostnqn": "r-nvmx03"
        }
      ]
    }
  ]
}
```

Configuration Parameters

Controller Parameters

Parameters in SNAP JSON configuration file. Default file is located in /etc/mlnx_snap/mlnx_snap.json.

Parameter	Description	Legal Values	Default
rpc_server	Describes the RPC server socket for passing through RPC commands. Relevant only when using vendor-specific RPC commands from host.	Any	""
offload	Enable full-offload mode	true/false	false
nr_io_queues	Maximum number of I/O queues. **Note** The actual number of queues is limited by number of queues supported by FW.	⩾ 0	32

Parameter	Description	Legal Values	Default
mn	Model number	String (up to 40 chars)	"MLX NVMe Ctrl"
sn	Serial number	String (up to 20 chars)	"MNC12"
nn	Number of namespaces (NN) indicates the maximum value of a valid NSID for the NVM subsystem. If the `mnan` field is cleared to `0h`, then this field also indicates the maximum number of namespaces supported by the NVM subsystem.	0-0xFFFFFFFE	0xFFFFFFFE
mnan	Maximum number of allowed namespaces (MNAN) supported by the NVM subsystem	1-0xFFFFFFFE	1024
mdts	Max data transfer size. This value is in units of the minimum memory page size (CAP.MPSMIN) and is reported as a power of two (2^n). A value of 0h indicates that there is no maximum data transfer size.	1-6	4
quirks	Bitmask for enabling specific NVMe driver quirks in order to work with non-NVMe spec compliant drivers. • Bit 0 – send namespace change async events even if driver does not explicitly request them via the SET_FTRS command **Note** Enable this if the NVMe driver knows to handle namespace change but does not use SET_FTRS. CentOS 7.5 inbox driver does this.	0x0-0x3	0x0

Parameter	Description	Legal Values	Default
	• Bit 1 – send new namespace change events even if previous ones are not yet cleared by the driver **Note** CentOS 7.5 inbox driver requires this. • Bit 2 – force Number of Namespaces (NN) on Identify controller to dynamically track and indicate both the maximum value of a valid NSID and the maximum number of namespaces supported by the NVM controller. There is no limitation on namespaces NSIDs on controller level. • Bit 3 – force OACS to enable namespace management bit **Note** VMWare driver requires this bit to be set.		
max_namespaces	Limit number of available namespaces	Any	1024

Backend Parameters

Theses parameters are used to define the backend server.

Note

Even though a list of backends can be configured, currently only a single backend is supported.

Parameter	Description	Legal Values	Default
type	Backend type: • "memdisk" – RAM based local storage • "nvmf_rdma" – NVMe-oF over RDMA remote storage • "posix_io" – file-based storage • "spdk_bdev" – SPDK block devices	nvmf_rdma, spdk_bdev	"spdk_bdev"
name	Depends on backend type: • "nvmf_rdma" – remote subsystem name • "memdisk"/"spdk_bdev" – unused • "posix_io" – backend filename	Any	Null
size_mb	Represents the desired size (in MB) of the opened backend. Relevant only for memdisk/posix_io backends.	Any	Unused
block_order	Represents the desired block size (in logarithmic scale) of the opened backend. Relevant only for memdisk/posix_io backends.	9, 12	Unused

Path Section

This section is relevant only if backend type is set to "nvmf_rdma". For each backend, a list of paths can be specified using the following parameters:

Parameter	Description	Legal Values	Default
addr	Target IPv4 address	String in x.x.x.x format	"192.168.101.2"
port	Target port number	1024-65534	4420

Parameter	Description	Legal Values	Default
ka_timeout_ms	Keepalive timeout in msec	>0	15000
Nqn	Host NQN	String up to 223-char long	"nqn.2014-08.org.nvmexpress:uuid:11111111-2222-3333-4444-555555555555"

Appendix – Frequently Asked Questions

How do I enable SNAP?

Please refer to chapter "mlnx_snap Installation".

How do I configure SNAP to support VirtIO-blk?

Please refer to section "VirtIO-blk Configuration".

How do I configure SNAP to work with both ports (for the same or for multiple targets)?

Assumptions:

- The remote target is configured with nqn "Test" and 1 namespace, and it exposes it through the 2 RDMA interfaces 1.1.1.1/24 and 2.2.2.1/24
- The RDMA interfaces are 1.1.1.2/24 and 2.2.2.2/24

Non-offload mode configuration:

1. Create the SPDK BDEVS. Run:

```
spdk_rpc.py bdev_nvme_attach_controller -b Nvme0 -t rdma -a
1.1.1.1 -f ipv4 -s 4420 -n Test
spdk_rpc.py bdev_nvme_attach_controller -b Nvme1 -t rdma -a
2.2.2.1 -f ipv4 -s 4420 -n Test
```

2. Create NVMe controller. Run:

```
snap_rpc.py controller_nvme_create mlx5_0 --subsys_id 0 -c
/etc/mlnx_snap/mlnx_snap.json --rdma_device mlx5_2
```

3. Attach the namespace twice, one through each port. Run:

```
snap_rpc.py controller_nvme_namespace_attach -c NvmeEmu0pf0
spdk Nvme0n1 1
snap_rpc.py controller_nvme_namespace_attach -c NvmeEmu0pf0
spdk Nvme1n1 2
```

At this stage, you should see /dev/nvme0n1 and /dev/nvme0n2 on the host "nvme list", both of which are mapped to the same remote disk via 2 different ports.

Full-offload mode configuration:

> **Note**
>
> Full-offload mode currently allows users to connect to multiple remote targets in parallel (but not to the same remote target through different paths).

1. Create 2 separate JSON full-offload configuration files (see section "Full Offload Mode"). Each describe a connection to remote target via different RDMA interface.

2. Configure 2 separate NVMe device entries to be exposed to the host either as hot-plugged PCIe functions (see section "Runtime Configuration"), or “static” ones (see section "Firmware Configuration").

3. Create 2 NVMe controllers, one per RDMA interface. Run:

```
snap_rpc.py subsystem_nvme_create Mellanox_NVMe_SNAP
"Mellanox NVMe SNAP Controller"
snap_rpc.py controller_nvme_create mlx5_0 --subsys_id 0 --
pf_id 0 -c /etc/mlnx_snap/mlnx_snap_p0.json --rdma_device
mlx5_2
snap_rpc.py subsystem_nvme_create  Mellanox_NVMe_SNAP
"Mellanox NVMe SNAP Controller"
snap_rpc.py controller_nvme_create mlx5_0 --subsys_id 1 --
pf_id 1 -c /etc/mlnx_snap/mlnx_snap_p1.json --rdma_device
mlx5_3
```

> **Note**
>
> NVMe controllers may also share the same NVMe subsystem. In this case, users must make sure all namespaces in all remote targets have a distinct NSID.

At this stage, you should see /dev/nvme0n1 and /dev/nvme1n1 on the host "nvme list".

How do I configure offload mode? Which protocols are supported?

Please refer to section "Full Offload Configuration".

For more information on full offload, please refer to section "Full Offload Mode".

How do I configure Firmware for SNAP?

Please refer to section "Firmware Configuration".

Can I work with custom SPDK on Arm?

MLNX SNAP is natively compiled against NVIDIA's internal branch of SPDK. It is possible to work with different SPDK versions, under the following conditions:

- mlnx-snap sources must be recompiled against the new SPDK sources
- The new SPDK version changes do not break any external SPDK APIs

Integration process:

1. Build SPDK (and DPDK) with shared libraries.

```
[spdk.git] ./configure --prefix=/opt/mellanox/spdk-custom --
disable-tests --disable-unit-tests --without-crypto --
without-fio --with-vhost --without-pmdk --without-rbd --with-
rdma --with-shared --with-iscsi-initiator --without-vtune --
without-isal
[spdk.git] make && sudo make install
[spdk.git] cp -r dpdk/build/lib/* /opt/mellanox/spdk-
custom/lib/
[spdk.git] cp -r dpdk/build/include/* /opt/mellanox/spdk-
custom/include/
```

> **Note**
>
> It is also possible to install DPDK in that directory but copying suffices.

> **Note**

Only the flag with-shared is mandatory

2. Build SNAP against the new SPDK.

```
[mlnx-snap.src] ./configure --with-snap --with-
spdk=/opt/mellanox/spdk-custom --without-gtest --prefix=/usr
[mlnx-snap.src] make -j8 && sudo make install
```

3. Append additional custom libraries to the mlnx-snap application. Set LD_PRELOAD="/opt/mellanox/spdk/lib/libspdk_custom_library.so".

> **Note**
>
> Additional SPDK/DPDK libraries required by libspdk_custom_library.so might also need to be attached to LD_PRELOAD.

> **Note**
>
> LD_PRELOAD setting can be added to /etc/default/mlnx_snap for persistent work with the mlnx_snap system service.

4. Run application.

Can I replace my backend storage at runtime?

NVMe protocol has an embedded support for backends (namespaces) attach/detach at runtime.

To change backend storage during runtime for NVMe, run:

```
snap_rpc.py controller_nvme_namespace_detach -c NvmeEmu0pf0 1
snap_rpc.py controller_nvme_namespace_attach -c NvmeEmu0pf0 spdk
nvme0n1 1
```

VirtIO-blk does not have similar support in its protocol's specification. Therefore, detaching while running IO results in error on any IO received between the request to detach and attach.

To change backend storage at runtime for virtio-blk, run:

```
snap_rpc.py controller_virtio_blk_bdev_detach VblkEmu0pf0
snap_rpc.py controller_virtio_blk_bdev_attach VblkEmu0pf0 spdk
nvme0n1
```

I'm suffering from low performance after updating to latest mlnx-snap. How can I fix it?

After adding the option to work with a large number of controllers, resource considerations had to be considered. It was necessary to pay special attention to the MSIX resource, which is limited to ~1K across the whole BlueField-2 card. Therefore, new PCI functions are now opened with limited resources by default (specifically, MSIX is set to 2).

User may choose to assign more resources for a specific function, as detailed in the following:

1. Increase the number of MSIX allowed to be assigned to a function (power-cycle may be required for changes to take effect):

   ```
   [dpu] mlxconfig -d /dev/mst/mt41686_pciconf0 s
   VIRTIO_BLK_EMULATION_NUM_MSIX=63
   ```

2. Hotplug virtio-blk PF with the increased value of MSIX.

```
[dpu] snap_rpc.py emulation_device_attach mlx5_0 virtio_blk --num_msix=63
```

3. Open the controller with increased number of queues (1 queue per MSIX, and leave another free MSIX for configuration interrupts):

```
[dpu] snap_rpc.py controller_virtio_blk_create mlx5_0 --pf_id 0 --bdev_type spdk --bdev Null0 --num_queues=62
```

For more information, please refer to section "Performance Optimization".

Document Revision History

Rev 3.8.0-5 – August 14, 2023

N/A

Rev 3.8.0 – December 15, 2023

Updated:

- Section "Virtio-blk Live Migration"

Rev 3.7.4 – August 10, 2023

Updated:

- Section "Virtio-blk Live Migration"

Rev 3.7.3 – July 05, 2023

N/A

Rev 3.7.2 – May 11, 2023

Updated:

- Connection using NVDA_TCP transport instructions in section "NVMe/TCP Zero Copy"

Rev 3.7.0 – October 31, 2022

Added:

- Section "Basic Configuration"

Updated:

- Section "Virtio-blk Transitional Device (0.95)"
- Section "Virtio-blk Live Migration"
- Section "NVMe/TCP Zero Copy" with MIN_HUGEMEM parameter

Rev 3.6.1 – July 31, 2022

Added:

- Page "Appendix – Live Upgrade"

Updated:

- Section "Zero Copy (SNAP-direct)"
- Section "NVMe/TCP Zero Copy"
- Page "Appendix – Host OS Configuration"

Rev 3.6.0 – May 31, 2022

Added:

- Section "NVMe/TCP Zero Copy"

Updated:

- Section "Related Documents"
- Code boxes in section "Emulation Devices Configuration (Hotplug)"
- Section "NVMe Controller"
- Section "VirtIO-blk Controller"
- Recommendation to section "Minimal Requirement"
- Section "Can I replace my backend storage at runtime?"
- Flag --without-isal and -r to step 1 of section "Can I work with custom SPDK on Arm?"

Rev 3.5.0 – December 16, 2021

Added:

- BFB to "Glossary"
- Section "Related Documents"
- Note to section "Increasing Number of Used Arm Cores"

Updated:

- Document ToC
- Page "SNAP Installation"
- Page "SNAP Commands"
- Page "Advanced Features"
- Section "Controller Parameters" by removing the following parameters:
    - sqes
    - cqes
    - cq_period
    - cq_max_count
    - oncs

Rev 3.4.0 – September 15, 2021

Added:

- Section "Hot-unplug"

Updated:

- Page "Overview"
- Section "PCIe Functions Management"
- Section "NVMe Controller Management"

- Section "Out-of-box Configuration"
- Section "Full Offload"
- Section "SR-IOV"
- Section "Robustness and Recovery"
- Section "Configuration File Examples"
- Section "Controller Parameters"
- Section "Hotplug Management"
- Section "Controller Management"
- FAQ "How do I configure SNAP to work with both ports"
- FAQ "Can I replace my backend storage at runtime?"

Removed section "Using Direct Verbs for Datapath"

Rev 3.3.0 – June 30, 2021

Updated:

- Section "Firmware Configuration"
- Section "Network Configuration"
- Section "NVMe Controller Management"
- Section "Full Offload"
- Section "Robustness and Recovery"
- Section "Hotplug Management"
- Section "Backend Management"
- FAQ "How do I configure SNAP to work with both ports"

Notice