NVIDIA BlueField DPU BSP v4.7.0

Table of Contents

1 About This Document 11
1.1 Intended Audience 11
1.2 Software Download 11
1.3 Technical Support 11
1.4 Glossary 12
1.5 Related Documentation 15
2 Initial Configuration 18
2.1 UEFI Menu 18
2.1.1 System Configuration 19
2.1.2 Secure Boot Configuration 23
2.1.3 RAM Disk Configuration 23
2.1.4 Tls Auth Configuration 24
2.1.5 iSCSI Configuration 25
2.1.6 Network Device List 26
2.2 System Configuration and Services 27
2.2.1 First Boot After BFB Installation 27
2.2.2 RDMA and ConnectX Driver Initialization 27
2.2.3 Firewall Configuration 28
2.3 Host-side Interface Configuration 30
2.3.1 Virtual Ethernet Interface 30
2.3.2 RShim Support for Multiple DPUs 31
2.3.3 Permanently Changing Arm-side MAC Address 34
2.3.4 OOB Ethernet Interface 35
2.4 Secure Boot 38
2.4.1 Supported BlueField DPUs 38
2.4.2 UEFI Secure Boot 39
2.4.3 Updating Platform Firmware 51
2.5 Default Passwords and Policies 53
2.5.1 BMC Passwords 53
2.5.2 UEFI Menu Password 53
2.5.3 GRUB Password 58
3 Release Notes 60

3.1 Changes and New Features 60
3.1.1 Changes and New Features in 4.7.0 60
3.2 Supported Platforms and Interoperability 61
3.2.1 Supported NVIDIA BlueField-3 DPU Platforms 61
3.2.2 Supported NVIDIA BlueField-2 DPU Platforms 63
3.2.3 Embedded Software 65
3.2.4 Supported DPU Linux Distributions (aarch64) 65
3.2.5 Supported Open vSwitch 66
3.3 Bug Fixes In This Version 66
3.4 Known Issues 67
3.5 Validated and Supported Cables and Modules 81
3.5.1 Cables Lifecycle Legend 81
3.5.2 Supported Cables and Modules for BlueField-3 82
3.5.3 Supported Cables and Modules for BlueField-2 82
3.6 Release Notes Change Log History 82
3.6.1 Changes and New Features in 4.6.0 82
3.6.2 Changes and New Features in 4.5.0 82
3.6.3 Changes and New Features in 4.2.0 82
3.6.4 Changes and New Features in 4.0.3 83
3.6.5 Changes and New Features in 4.0.2 83
3.6.6 Changes and New Features in 3.9.3 83
3.6.7 Changes and New Features in 3.9.2 83
3.6.8 Changes and New Features in 3.9.0 83
3.6.9 Changes and New Features in 3.8.5 84
3.6.10 Changes and New Features in 3.8.0 84
3.7 Bug Fixes History 84
4 BlueField Software Overview 95
4.1 Debug Tools 95
4.2 BlueField-based Storage Appliance 95
4.3 BlueField Architecture 96
4.4 System Connections 96
4.4.1 System Consoles 97
4.4.2 Network Interfaces 97
5 Software Installation and Upgrade 99

5.1 Deploying BlueField Software Using BFB from Host 99
5.1.1 Uninstall Previous Software from Host 100
5.1.2 Install RShim on Host 100
5.1.3 Ensure RShim Running on Host 102
5.1.4 Installing Ubuntu on BlueField 102
5.1.5 Ubuntu Boot Time Optimizations 113
5.1.6 DHCP Client Configuration 114
5.1.7 Ubuntu Dual Boot Support 114
5.2 Deploying BlueField Software Using BFB from BMC 116
5.2.1 Ensure RShim is Running on BMC 117
5.2.2 Changing Default Credentials Using bf.cfg 118
5.2.3 Password Policy 118
5.2.4 BFB Installation 119
5.2.5 Verify BFB is Installed 125
5.2.6 Firmware Upgrade 125
5.2.7 Updating NVConfig Params 126
5.3 Deploying BlueField Software Using BFB with PXE 127
5.3.1 PXE Server Preparations 127
5.3.2 PXE Sequence 128
5.3.3 PXE Sequence with Redfish 129
5.4 Deploying NVIDIA Converged Accelerator 130
5.4.1 Configuring Operation Mode 131
5.4.2 Verifying Configured Operational Mode 131
5.4.3 Verifying GPU Ownership 132
5.4.4 GPU Firmware 132
5.5 Installing Repo Package on Host Side 133
5.5.1 Removing Previously Installed DOCA Runtime Packages 133
5.5.2 Downloading DOCA Runtime Packages 133
5.5.3 Installing Local Repo Package for Host Dependencies 138
5.6 Installing Popular Linux Distributions on BlueField 140
5.6.1 Building Your Own BFB Installation Image 140
5.6.2 Installing Linux Distributions 140
5.6.3 BlueField Linux Drivers 140
5.7 Updating DPU Software Packages Using Standard Linux Tools 141

5.7.1 Upgrading Boot Software 142
6 Management 154
6.1 Performance Monitoring Counters 154
6.1.1 Performance Data Collection Mechanisms 155
6.1.2 List of Supported Events 156
6.1.3 Programming Counter to Monitor Events 163
6.2 Intelligent Platform Management Interface 164
6.2.1 DPU BMC IPMI Commands 165
6.2.2 BlueField IPMI Commands 168
6.2.3 External Host IPMI Commands 169
6.2.4 Loading and Using IPMI on BlueField Running CentOS 170
6.3 Redfish 172
6.3.1 BIOS Configuration Schema 172
6.3.2 BlueField Platform Inventory 174
6.3.3 Boot Override 175
6.3.4 Boot Order 176
6.3.5 BIOS Attributes 177
6.4 Logging 178
6.4.1 RShim Logging 178
6.4.2 IPMI Logging in UEFI 182
6.4.3 ACPI BERT Logging 184
6.5 SoC Management Interface 185
6.5.1 Installation and Upgrade 185
6.5.2 Host-side Interface Configuration 186
6.5.3 SoC Management Interface Features and Functionality 191
6.5.4 DPU Configuration File 192
6.6 BlueField OOB Ethernet Interface 192
6.6.1 OOB Interface MAC Address 192
6.6.2 Supported ethtool Options for OOB Interface 194
6.6.3 IP Address Configuration for OOB Interface 195
7 BlueField Operation 196
7.1 Functional Diagram 196
7.2 Modes of Operation 197
7.2.1 DPU Mode 198

7.2.2 Zero-trust Mode 199
7.2.3 NIC Mode 200
7.3 Kernel Representors Model 205
7.4 Multi-Host 206
7.4.1 Representors 207
7.5 Virtual Switch on DPU 209
7.5.1 Verifying Host Connection on Linux 210
7.5.2 Verifying Connection from Host to BlueField 210
7.5.3 Verifying Host Connection on Windows 211
7.5.4 Enabling OVS HW Offloading 211
7.5.5 Enabling OVS-DPDK Hardware Offload 212
7.5.6 Configuring DPDK and Running TestPMD 213
7.5.7 Flow Statistics and Aging 214
7.5.8 Connection Tracking Offload 214
7.5.9 Offloading VLANs 217
7.5.10 VXLAN Tunneling Offload 217
7.5.11 GRE Tunneling Offload 218
7.5.12 GENEVE Tunneling Offload 219
7.5.13 Using TC Interface to Configure Offload Rules 220
7.5.14 VirtIO Acceleration Through Hardware vDPA 221
7.6 Configuring Uplink MTU 221
7.7 Link Aggregation 222
7.7.1 LAG Modes 222
7.7.2 Prerequisites 223
7.7.3 LAG Configuration 224
7.7.4 Removing LAG Configuration 225
7.7.5 LAG on Multi-host 226
7.8 Scalable Functions 227
7.8.1 Scalable Function Configuration 227
7.9 RDMA Stack Support on Host and Arm System 231
7.9.1 Separate Host Mode 231
7.9.2 Embedded CPU Mode 231
7.10 Controlling Host PF and VF Parameters 231
7.10.1 Setting Host PF and VF Default MAC Address 231

7.10.2 Setting Host PF and VF Link State ... 232
7.10.3 Querying Configuration ... 232
7.10.4 Disabling Host Networking PFs ... 232
7.11 DPDK on BlueField DPU ... 233
7.12 BlueField SNAP ... 233
7.13 BlueField SR-IOV ... 233
7.14 Compression Acceleration ... 234
7.14.1 Configuring Compression Acceleration ... 234
7.15 Public Key Acceleration ... 234
7.15.1 PKA Prerequisites ... 235
7.15.2 PKA Use Cases ... 235
7.16 IPsec Functionality ... 236
7.16.1 Transparent IPsec Encryption and Decryption ... 236
7.16.2 IPsec Hardware Offload: Crypto Offload ... 236
7.16.3 IPsec Hardware Offload: Packet Offload ... 236
7.16.4 OVS IPsec ... 243
7.17 fTPM over OP-TEE ... 247
7.17.1 Enabling OP-TEE on BlueField-3 ... 249
7.17.2 Verifying BlueField-3 is Running OP-TEE ... 249
7.18 QoS Configuration ... 250
7.18.1 devlink port function rate add ... 251
7.18.2 devlink port function rate del ... 251
7.18.3 devlink port function rate set tx_max tx_share ... 252
7.18.4 devlink port function rate set parent ... 252
7.18.5 devlink port function rate set noparent ... 253
7.18.6 devlink port function rate show ... 253
7.19 Virtio-net Emulated Devices ... 253
7.20 Shared RQ Mode ... 253
8 Troubleshooting and How-Tos ... 255
8.1 NVIDIA BlueField Reset and Reboot Procedures ... 255
8.1.1 BlueField System Reboot ... 255
8.1.2 BlueField System-level Reset ... 255
8.2 RShim Troubleshooting and How-Tos ... 259
8.2.1 Another backend already attached ... 259

8.2.2 RShim driver not loading ... 259
8.2.3 Change ownership of RShim from NIC BMC to host ... 261
8.2.4 How to support multiple DPUs on the host ... 262
8.2.5 BFB installation monitoring ... 262
8.3 Connectivity Troubleshooting ... 262
8.3.1 Connection (ssh, screen console) to the BlueField is lost ... 262
8.3.2 Driver not loading in host server ... 263
8.3.3 No connectivity between network interfaces of source host to destination device ... 264
8.3.4 Uplink in Arm down while uplink in host server up ... 265
8.4 Performance Troubleshooting ... 265
8.4.1 Degradation in performance ... 265
8.5 PCIe Troubleshooting and How-Tos ... 265
8.5.1 Insufficient power on the PCIe slot error ... 265
8.5.2 HowTo update PCIe device description ... 266
8.5.3 HowTo handle two BlueField DPU devices in the same server ... 266
8.6 SR-IOV Troubleshooting ... 266
8.6.1 Unable to create VFs ... 266
8.6.2 No traffic between VF to external host ... 266
8.7 eSwitch Troubleshooting ... 267
8.7.1 Unable to configure legacy mode ... 267
8.7.2 Arm appears as two interfaces ... 268
8.8 Isolated Mode Troubleshooting and How-Tos ... 269
8.8.1 Unable to burn FW from host server ... 269
8.9 General Troubleshooting ... 269
8.9.1 Server unable to find the DPU ... 269
8.9.2 DPU no longer works ... 269
8.9.3 DPU stopped working after installing another BFB ... 269
8.9.4 Link indicator light is off ... 270
8.9.5 Link light is on but no communication is established ... 270
8.10 Installation Troubleshooting and How-Tos ... 270
8.10.1 BlueField target is stuck inside UEFI menu ... 270
8.10.2 BFB does not recognize the BlueField board type ... 270
8.10.3 Unable to load BL2, BL2R, or PSC image ... 270
8.10.4 CentOS fails into "dracut" mode during installation ... 270

8.10.5 How to find the software versions of the running system 271
8.10.6 How to upgrade the host RShim driver 271
8.10.7 How to upgrade the boot partition (ATF & UEFI) without re-installation ... 271
8.10.8 How to upgrade ConnectX firmware from Arm side 272
8.10.9 How to configure ConnectX firmware 272
8.10.10 How to use the UEFI boot menu 272
8.10.11 How to Use the Kernel Debugger (KGDB)...................................... 272
8.10.12 How to enable/disable SMMU 273
8.10.13 How to change the default console of the install image 273
8.10.14 How to change the default network configuration during BFB installation. 274
8.10.15 Sanitizing DPU eMMC and SSD Storage.. 274
9 Windows Support.. 276
9.1 Network Drivers 276
9.2 RShim Drivers... 276
9.3 Verifying RShim Drivers Installation.. 276
9.4 Accessing BlueField DPU From Host.. 277
9.5 RShim Ethernet Driver 280
9.6 MlxRshimBus Driver.. 281
9.7 RshimCmd Tool 281
9.8 BlueField UEFI System Boot Customizations during Installation 281
9.9 EventLogs and Driver Logging... 282
9.9.1 MlxRShimBus Driver.. 282
9.9.2 MlxRShim Serial Driver 282
9.9.3 MlxRShim Ethernet Driver 283
10 Document Revision History 284
10.1 Rev 4.7.0 - May 06, 2024.. 284
10.2 Rev 4.6.0 - February 08, 2024 284
10.3 Rev 4.5.0 - December 12, 2023... 284
10.4 Rev 4.2.2 - October 24, 2023 285
10.5 Rev 4.2.0 - August 10, 2023 285
10.6 Rev 4.0.2 - May 08, 2023... 285
10.7 Rev 3.9.3 - November 02, 2022... 286
10.8 Rev 3.9.2 - August 02, 2022 286
10.9 Rev 3.9 - May 03, 2022 287

10.10 Rev 3.8.5 – January 19, 2022 287

11 Legal Notices and 3rd Party Licenses 288

1 About This Document

NVIDIA® BlueField® DPU software is built from the BlueField BSP (Board Support Package) which includes the operating system and the DOCA framework. BlueField BSP includes the bootloaders and other essentials for loading and setting software components. The BSP loads the official BlueField operating system (Ubuntu reference Linux distribution) to the DPU. DOCA is the software framework and SDK for the development of applications and infrastructure services. DOCA includes runtime libraries; the DOCA Runtime stack for Arm supports various accelerations for storage, networking, and security. As such, customers can run any Linux-based application in the BlueField software environment seamlessly.

This guide provides product release notes as well as information on the BSP and how to develop and/or customize applications, system software, and file system images for the BlueField platform.

Important: Make sure to download the latest available software packages for the procedures documented in this guide to run as expected.

1.1 Intended Audience

This document is intended for software developers and DevOps engineers interested in creating and/or customizing software applications and system software for the NVIDIA BlueField DPU platform.

1.2 Software Download

To download product software, refer to the DOCA SDK developer zone.

1.3 Technical Support

Firmware Compatibility

For BlueField-3, a firmware version of 32.38.1002 or greater requires a BFB version of 2.2.0 or higher. Downgrading to lower BFB/firmware versions may result in anomalous behavior.

Proper Power Cycle Procedure

Make sure to perform a graceful shutdown of the Arm OS in advance of performing system/host power cycle when required by the manual.

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.

1.4 Glossary

Term	Description
ACE	AXI coherency extensions
ACPI	Advanced configuration and power interface
AMBA®	Advanced microcontroller bus architecture
ARB	Arbitrate
ATF	Arm-trusted firmware
AXI4	Advanced eXtensible Interface 4
BDF address	Bus, device, function address. This is the device's PCIe bus address to uniquely identify the specific device.
BERT	Boot error record table
BF_INST_DIR	The directory where the BlueField software is installed
BFB	BlueField bootstream
BMC	Board management controller
BSD	BlueField software distribution
BSP	BlueField support package
BUF	Buffer
CBS	Committed burst size
CHI	Coherent hub interface; Arm® protocol used over the BlueField Skymesh specification
CIR	Committed information rate
CL	Cache line
CMDQ	Command queue
CMO	Cache maintenance operation
COB	Collision buffer
DAT	Data
DEK	Data encryption key
DMA	Direct memory access
DOCA	DPU SDK
DOT	Device ownership transfer
DPA	Data path accelerator; an auxiliary processor designed to accelerate data-path operations
DPDK	Data plane development kit
DPI	Deep packet inspection
DPU	Data processing unit, the third pillar of the data center with CPU and GPU

Term	Description
DVM	Distributed virtual memory
DW	Dword
EBS	Excess burst size
ECPF	Embedded CPU physical function
EIR	Excess information rate
EMEM/EMI	External memory interface; block in the MSS which performs the actual read/write from the DDR device
eMMC	Embedded multi-media card
ESP	EFI system partition
ESP header	Encapsulating security payload
EU	Execution unit. HW thread; a logical DPA processing unit.
FIPS	Federal Information Processing Standards
FPGA	Field-programmable gate arrays
FS	File system
FW	Firmware
GDB	GNU debugger
GPT	GUID partition table
HCA	Host-channel adapter
HNF	Home node interface
Host	When referring to "the host" this documentation is referring to the **server host**. When referring to the Arm based host, the documentation will specifically call out "Arm host". • Server host OS refers to the Host Server OS (Linux or Windows) • Arm host refers to the AARCH64 Linux OS which is running on the BlueField Arm Cores
HW	Hardware
hwmon	Hardware monitoring
IB	InfiniBand
ICM	Interface configuration memory
IKE	Internet key exchange
IPMB	Intelligent platform management bus
IPMI	Intelligent platform management interface
IR	Intermediate representation
KGDB	Kernel debugger
KGDBOC	Kernel debugger over console
LAT	Latency
LCRD	Link credit
LSO	Large send offload
LTO	Link-time optimization

Term	Description
MMIO	Memory-mapped I/O
MSB	Most significant bit
MSS	Memory subsystem
MST	Mellanox software tools
NAT	Network address translation
NIC	Network interface card
NIST	National Institute of Standards and Technology
NS	Namespace
OCD	On-chip debugger
OOB	Out-of-band
OS	Operating system
OVS	Open vSwitch
PBS	Peak burst size
PCIe	PCI Express; Peripheral Component Interconnect Express
PF	Physical function
PIR	Peak information rate
PK	Platform key
PKA	Public key accelerator
POC	Point of coherence
RD	Read
RDMA	Remote direct memory access
RegEx	Regular expression
REQ	Request
RES	Response
RMC	Remote management controller
RN	Request node RN-F - Fully coherent request node RN-D - IO coherent request node with DVM support RN-I - IO coherent request node
RNG	Random number generator/generation
RoCE	Ethernet and RDMA over converged Ethernet
RQ	Receive queue
RShim	Random Shim
RTT	Round-trip time
RX	Receive
SA	Security association
SBSA	Server base system architecture

Term	Description
SDK	Software development kit
SF	Sub-function or scalable function
SG	Scatter-gather
SHA	Secure hash algorithm
SMMU	System memory management unit
SNP	Snooping
SQ	Send queue
SR-IOV	Single-root IO virtualization
STL	Stall
Sync event	Synchronization event
TBU	Translation buffer unit
TIR	Transport interface receive
TIS	Transport interface send
TLS	Transport layer security
TRB	Trail buffer
TSO	TCP send offload
TSO	Total store order
TX	Transmit
UDS	Unix domain socket
UEFI	Unified extensible firmware interface
UPVS	UEFI persistent variable store
VF	Virtual function
VFE	Virtio full emulation
VM	Virtual machine
VPI	Virtual protocol interconnect
VST	Virtual switch tagging
WorkQ or workq	Work queue
WQE	Work queue elements
WR	Write
WRDB	Write data buffer

1.5 Related Documentation

Document Name	Description
InfiniBand Architecture Specification, Vol. 1, Release 1.3.1	The InfiniBand Architecture Specification that is provided by IBTA

Document Name	Description
Firmware Release Notes	See Firmware Release Notes
MFT Documentation	See Firmware Tools Release Notes and User Manual
NVIDIA OFED for Linux User Manual	Intended for system administrators responsible for the installation, configuration, management and maintenance of the software and hardware of VPI adapter cards
WinOF Documentation	See WinOF Release Notes and User Manual
NVIDIA BlueField BMC Software User Manual	This document provides general information concerning the BMC on the NVIDIA® BlueField® DPU, and is intended for those who want to familiarize themselves with the functionality provided by the BMC
NVIDIA BlueField-3 DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware for operating the board, and a step-by-step plan of how to bring up BlueField-3 DPUs
NVIDIA BlueField-2 Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField-2 Ethernet DPUs
NVIDIA BlueField-2 InfiniBand/Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField-2 InfiniBand/ Ethernet DPUs
NVIDIA BlueField InfiniBand/Ethernet DPU User Guide	This document provides details as to the interfaces of the board, specifications, required software and firmware, and a step-by-step plan of how to bring up BlueField InfiniBand/ Ethernet DPUs
NVIDIA DOCA SDK	The NVIDIA DOCA™ SDK enables developers to rapidly create applications and services on top of NVIDIA® BlueField® data processing units (DPUs), leveraging industry-standard APIs. With DOCA, developers can deliver breakthrough networking, security, and storage performance by harnessing the power of NVIDIA's DPUs.
NVIDIA BlueField Reference Platform Hardware User Manual	Provides details as to the interfaces of the reference platform, specifications and hardware installation instructions
NVIDIA BlueField Ethernet Controller Card User Manual	This document provides details as to the interfaces of the board, specifications, required software and firmware for operating the card, hardware installation, driver installation and bring-up instructions
NVIDIA BlueField UEFI Secure Boot User Guide	This document provides details and directions on how to enable UEFI secure boot and sign UEFI images
NVIDIA BlueField Secure Boot User Guide	This document provides guidelines on how to enable the Secure Boot on BlueField DPUs
NVIDIA BlueField SNAP and virtio-blk SNAP Documentation	This document describes the configuration parameters of NVMe SNAP and virtio-blk SNAP in detail
PKA Driver Design and Implementation Architecture Document	This document provides a description of the design and implementation of the Public Key accelerator (PKA) hardware driver. The driver manages and controls the EIP-154 Public Key Infrastructure Engine, an FIPS 140-3 compliant PKA and operates as a co-processor to offload the processor of the host.

Document Name	Description
PKA Programming Guide	This document is intended to guide a new crypto application developer or a public key user space driver. It offers programmers the basic information required to code their own PKA-based application for NVIDIA® BlueField® DPU.

2 Initial Configuration

The following pages provide instructions regarding general configuration of the BlueField DPU.

- UEFI Menu
- System Configuration and Services
- Host-side Interface Configuration
- Secure Boot
- Default Passwords and Policies

2.1 UEFI Menu

Unified Extensible Firmware Interface (UEFI) is low-level firmware that is part of the NVIDIA® BlueField® bootloader stack. UEFI acts as an interface between the BlueField's Arm-trusted firmware (ATF) bootloader and the OS.

The UEFI specification is available at UEFI.org.

UEFI provides a menu which supports certain configuration options. This section lists and describes configurations supported from the UEFI Device Manager menu.

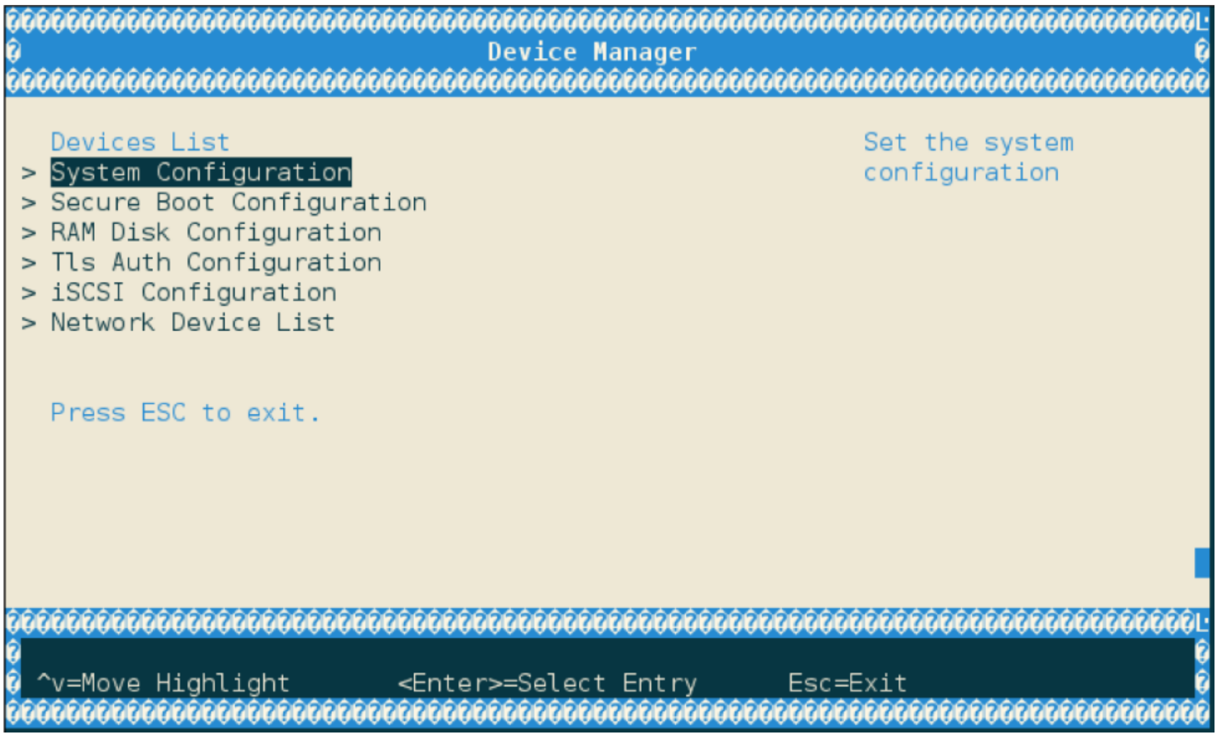
For more complete information beyond the Device Manager menu option, please refer to the NVIDIA Networking Server-Side Documentation of Flexboot & UEFI > User Manual > User Interface > HII (UEFI) System Settings Configuration Options.

Most of these menu items are also configurable via Redfish (when enabled).

To access the UEFI menu, users must have a connection to the BlueField console either through a UART serial port or the virtual RShim console device. To enter the UEFI menu, hit the Esc key twice during the normal boot sequence.

All BlueField platforms ship with a default UEFI menu password, `bluefield`. If the password is set to `bluefield` when you enter the UEFI menu, users are prompted to change it.

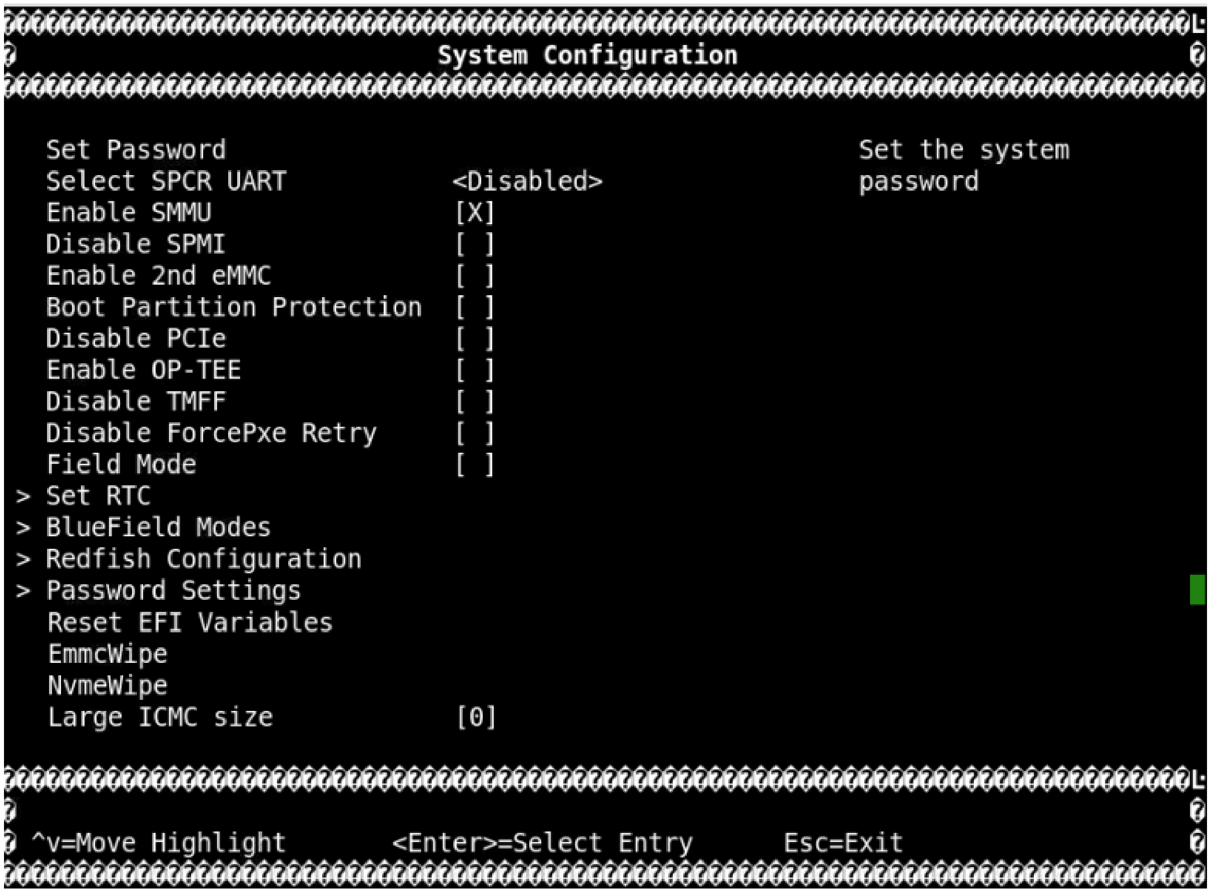
NVIDIA strongly recommends all DPUs have their UEFI password set to a non-default value. This can be done using the UEFI menu or Redfish.


```
                              Device Manager

   Devices List                                          Set the system
 > System Configuration                                  configuration
 > Secure Boot Configuration
 > RAM Disk Configuration
 > Tls Auth Configuration
 > iSCSI Configuration
 > Network Device List


   Press ESC to exit.


  ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

2.1.1 System Configuration

Lists different system configuration options.

Some configuration options may require a system reset to take effect.



```
                         System Configuration

   Set Password                                      Set the system
   Select SPCR UART           <Disabled>             password
   Enable SMMU                [X]
   Disable SPMI               [ ]
   Enable 2nd eMMC            [ ]
   Boot Partition Protection  [ ]
   Disable PCIe               [ ]
   Enable OP-TEE              [ ]
   Disable TMFF               [ ]
   Disable ForcePxe Retry     [ ]
   Field Mode                 [ ]
 > Set RTC
 > BlueField Modes
 > Redfish Configuration
 > Password Settings
   Reset EFI Variables
   EmmcWipe
   NvmeWipe
   Large ICMC size            [0]

  ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit
```

Menu Option	Description
Set Password	Set the system password. Set the UEFI password. All BlueField Platforms ship with a default UEFI menu password, `bluefield`. If the password is set to `bluefield` when you enter the UEFI menu, users are prompted to change it. NVIDIA strongly recommends all DPUs have their UEFI password set to a non-default value. This can be done using the UEFI menu or Redfish.
Select SPCR UART	Choose UART for serial port console redirection `[<Disabled>|<UART Port 0> | <UART Port 1>]`. Users may set the SPCR table (ACPI) to point to UART0, UART1, or disable the feature. The OS can reference this table to steer serial output. For example, Linux uses this table for its earlycon feature. Leave this attribute to its default if you are not certain how to configure it, or you may destabilize your system.

Menu Option	Description
Enable SMMU	Enable/disable the SMMU. BlueField Platforms have an integrated SMMU on the SoC. Users may enable or disable this unit. Enabling it can make the system more secure but, with certain network flows, the enabled SMMU could cause performance issues. Leave this attribute to its default if you do not certain how to configure it.
Disable SPMI	Enable/disable ACPI server platform management interface table. Allows users to enable/disable the ACPI SPMI table. This table instructs the OS on what interface/device to use for the IPMI SSIF. Leave this attribute to its default if you do not certain how to configure it.
Enable 2nd eMMC	Enable/disable the second eMMC. Some legacy BlueField systems have 2 eMMC devices. This feature has been discontinued. Leave this attribute to its default (disabled) if you do not certain how to configure it, or your system will not boot correctly.
Boot Partition Protection	Enable/disable the eMMC boot partition protection. Takes effect after reboot. There are 2 logical "boot partitions" on the eMMC device used to store ATF/UEFI code. These are referred to as the primary/secondary boot partitions. Users can write-protect these partitions using this attribute. These are separate devices from the flash storage used by the OS (for file systems). They do not contain file systems and are only used for storing binary boot code on raw flash. Do not confuse an eMMC boot partition with an EFI System Partition (ESP) used to store boot loaders and OS images on a FAT32 file system. If secure boot is enabled, these partitions are write-protected by default. This menu option is not currently supported for BlueField-3.
Disable PCIe	Enable/disable PCIe root complex. Normally, UEFI enumerates the PCIe bus during the boot process and reports this information to the OS via the ACPI SSDT table. If this attribute is disabled, UEFI does not populate the SSDT with the PCIe root complex information, so the OS does not have visibility to devices on the PCIe bus. This attribute is used for diagnostic purposes and should not be modified.

Menu Option	Description
Enable OP-TEE	Enable/disable support for trusted execution environment. Do not enable this feature. More information will be provided in future releases.
Disable TMFF	Enable/disable the BlueField-specific ACPI TMFIFO table. This can be used by some OSes to perform console/debugging over the BlueField TMFIFO interface. It can override the SPCR table. Leave this attribute to its default if you do not certain how to configure it.
Disable ForcePxe Retry	If enabled, PXE boot option entries are attempted only once instead of retrying them in a loop when "ForcePxe" is requested via IPMI interface
Field Mode	Disable/enable NIC BMC field mode. Allows users to enable/disable NIC BMC field mode. When the NIC BMC has field mode enabled, most of its functionality is disabled (beyond the serial console). The BlueField Platform's OOB interface will also not be functional if field mode is enabled. Leave this attribute to its default unless you are certain you wish to enable field mode on the NIC BMC. Consult the DPU BMC user manual for more information on field mode.
Set RTC	Allows users to set the time and date for the real-time clock.
BlueField Modes	• Internal CPU Model: [<Separated>|<Embedded>] • Host Privilege Level: [<Restricted>|<Privileged>] • NIC Mode - sets the BlueField to operate in either NIC mode or DPU mode Any change to this attribute requires device reset to take effect.
Redfish Configuration	Enable/disable Redfish support. If UEFI is unable to discover a Redfish server, it reverts to using the defined UEFI boot options (i.e., the "normal" UEFI boot sequence). Disabling Redfish helps improve boot time as the Redfish server discovery process is skipped. The `RTCSync` option syncs RTC time with Redfish time under the Manager schema.
Password Settings	• Default Password Policy - mandates the password being set adheres to the new policy of 12 characters minimum and 64 characters maximum. The last 5 passwords cannot be reused. • Set Legacy Password - set password with legacy password policy to accommodate a UEFI firmware downgrade. The new password policy (default) is not compatible with older versions of UEFI firmware.

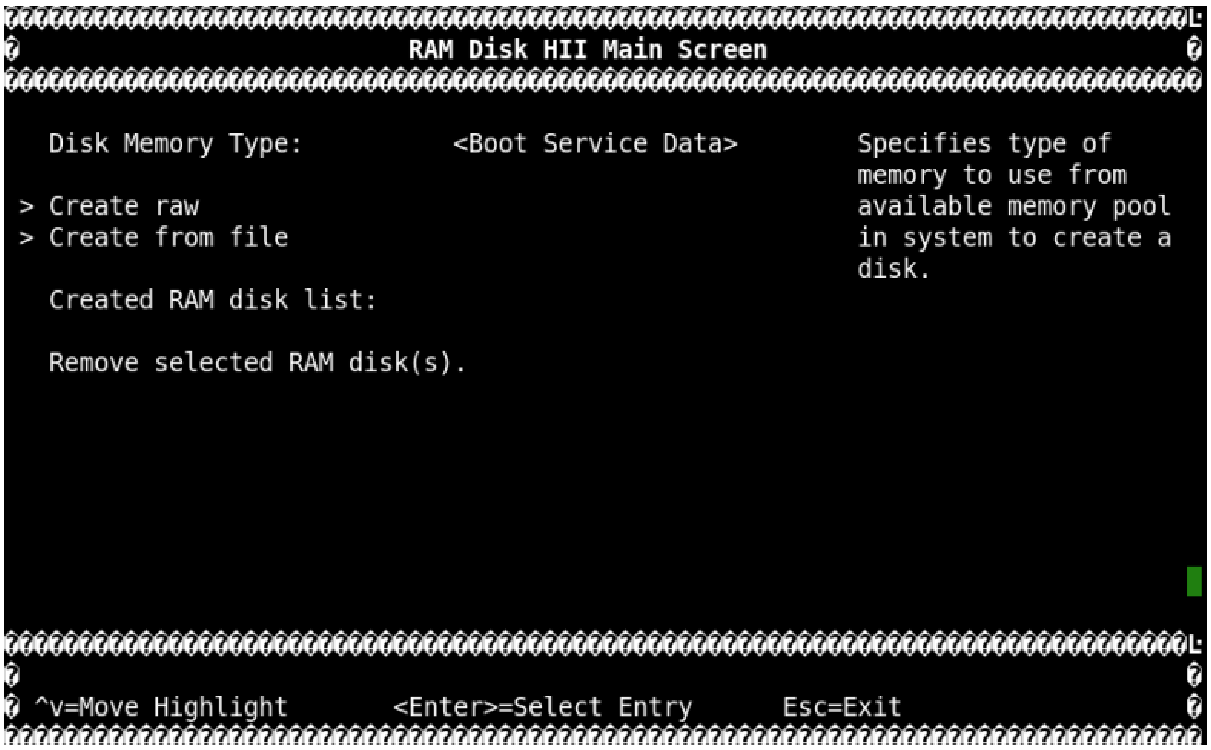
Menu Option	Description
Reset EFI Variables	This action clears all EFI variables to factory default state. Reset the device to take effect. Only reset the EFI variable store under the advice of NVIDIA Enterprise Support. Resetting the EFI variable store deletes all UEFI variables including the boot options and the system may not boot without setting new boot options.
EmmcWipe	Clears the eMMC disk. The action is immutable and all data on eMMC is lost after it is performed.
NvmeWipe	Clears the NVMe SSD. This action is immutable and all data on NVMe SSD is lost after it is performed.
Large ICMC size	Set the large ICMC size in Hex and MB. Valid value: 0-100000h in 80h increments. This menu option is only relevant for BlueField-3 platforms.

2.1.2 Secure Boot Configuration

Please refer to section "UEFI Secure Boot" for more information.

2.1.3 RAM Disk Configuration

Provides option to create/delete RAM disks.

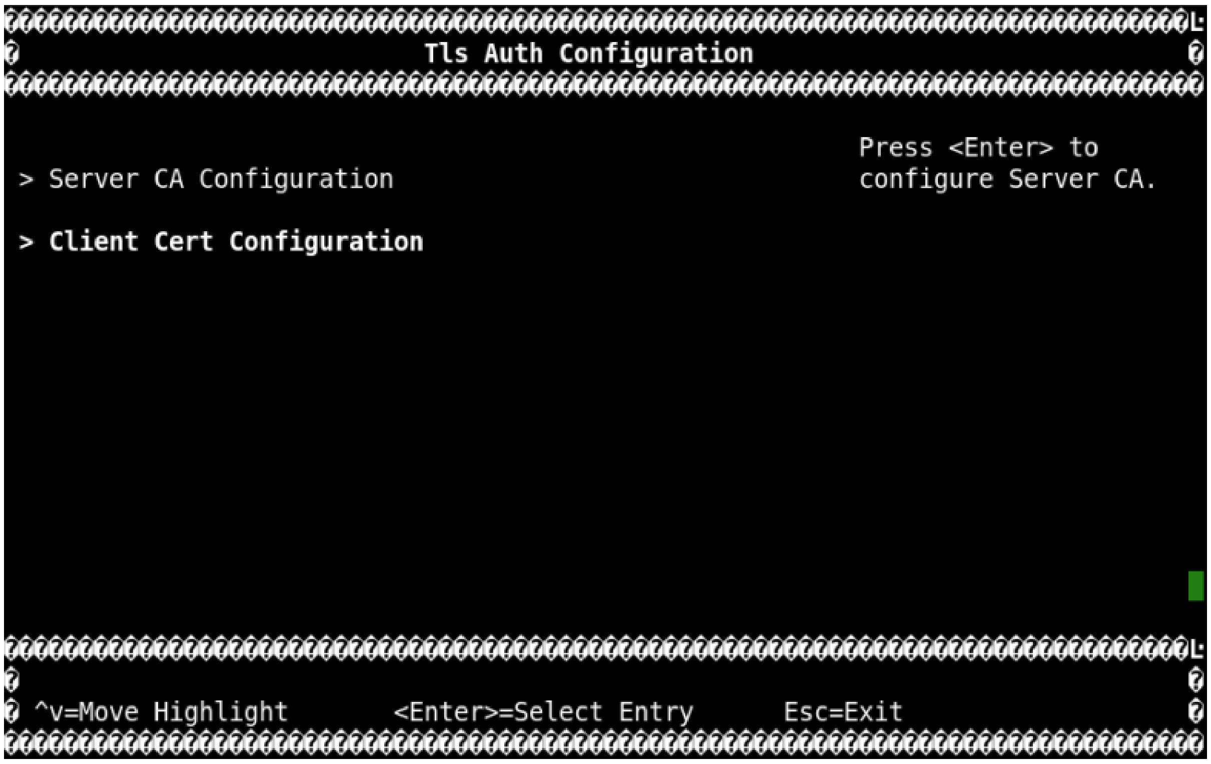


```
                         RAM Disk HII Main Screen

   Disk Memory Type:          <Boot Service Data>         Specifies type of
                                                          memory to use from
 > Create raw                                             available memory pool
 > Create from file                                       in system to create a
                                                          disk.
   Created RAM disk list:

   Remove selected RAM disk(s).



 ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

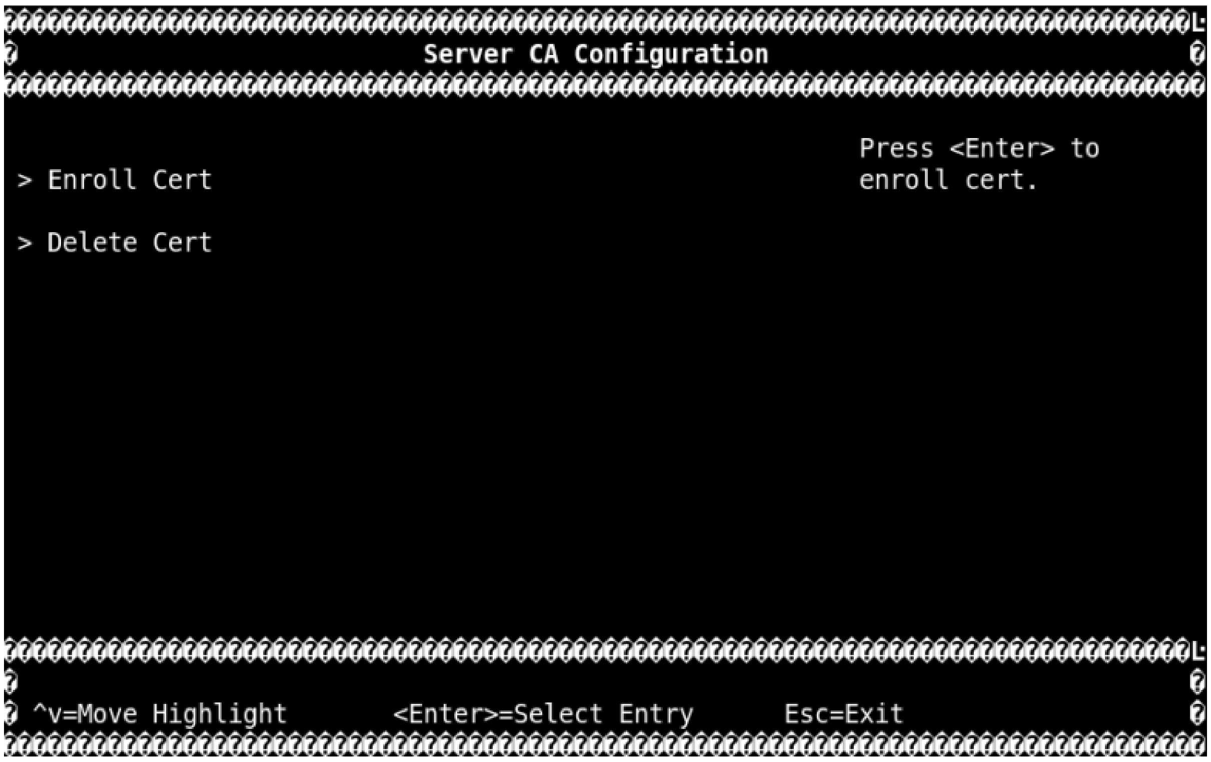
2.1.4 Tls Auth Configuration

Provides configuration (enroll/delete) of TLS auth certificates for HTTPS traffic in UEFI.

If TLS Auth certificate is configured then all HTTPS traffic on all network interfaces will be verified. UEFI only supports Server CA configuration, Client CA configuration is currently not supported.



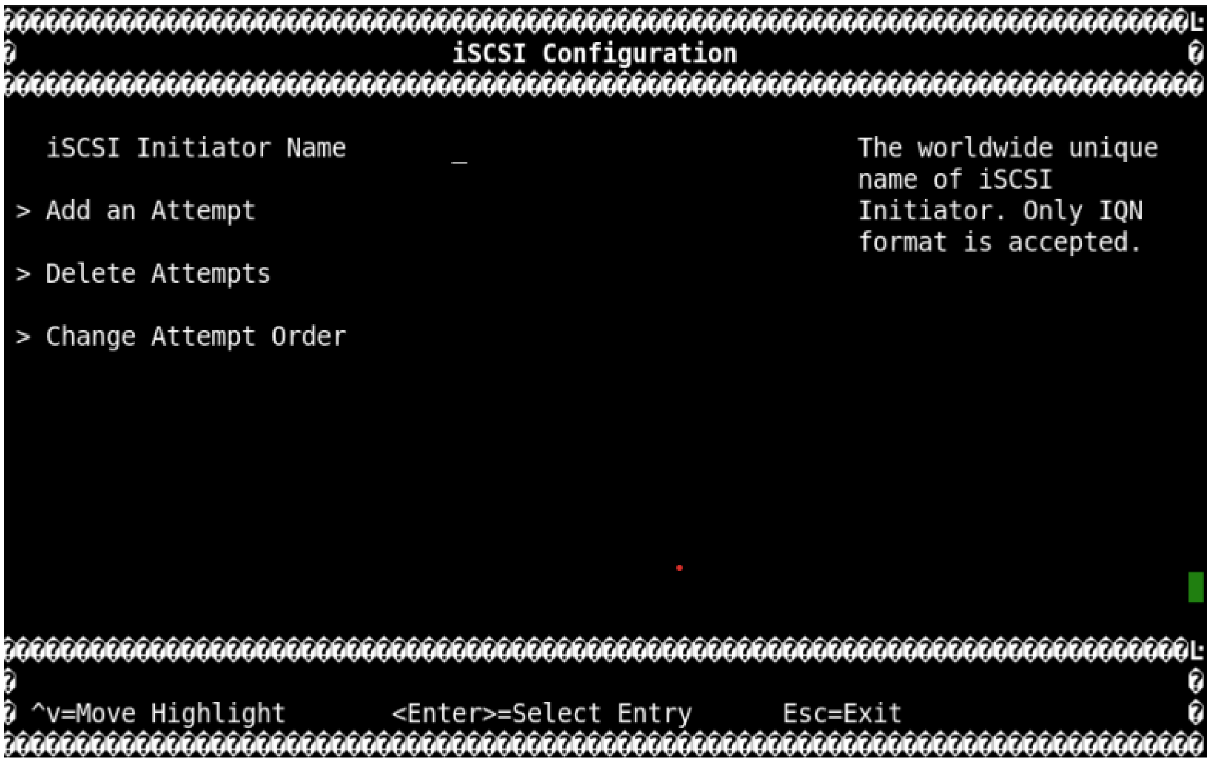
```
                         Tls Auth Configuration

                                                    Press <Enter> to
> Server CA Configuration                           configure Server CA.

> Client Cert Configuration

  ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```



```
                    Server CA Configuration

                                              Press <Enter> to
> Enroll Cert                                 enroll cert.

> Delete Cert


 ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit
```

2.1.5 iSCSI Configuration

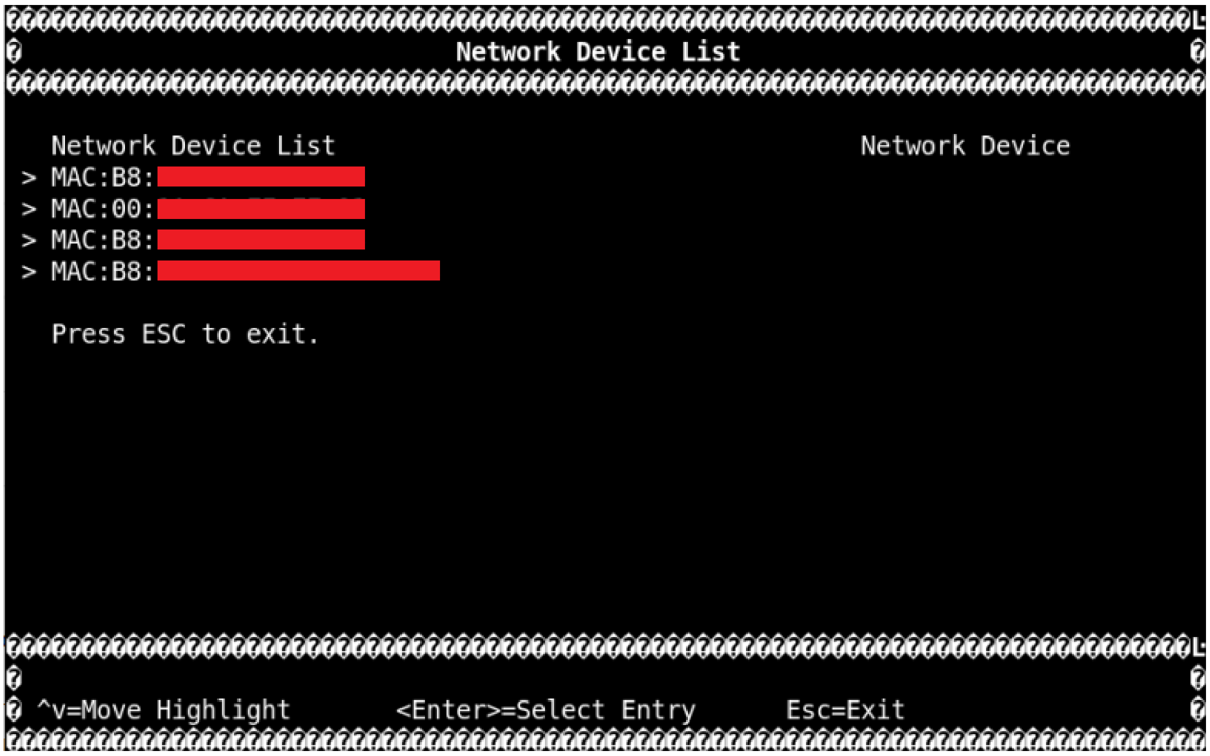
Provides configuration options for iSCSI.



```
                      iSCSI Configuration

  iSCSI Initiator Name        _               The worldwide unique
                                              name of iSCSI
> Add an Attempt                              Initiator. Only IQN
                                              format is accepted.
> Delete Attempts

> Change Attempt Order


 ^v=Move Highlight      <Enter>=Select Entry      Esc=Exit
```

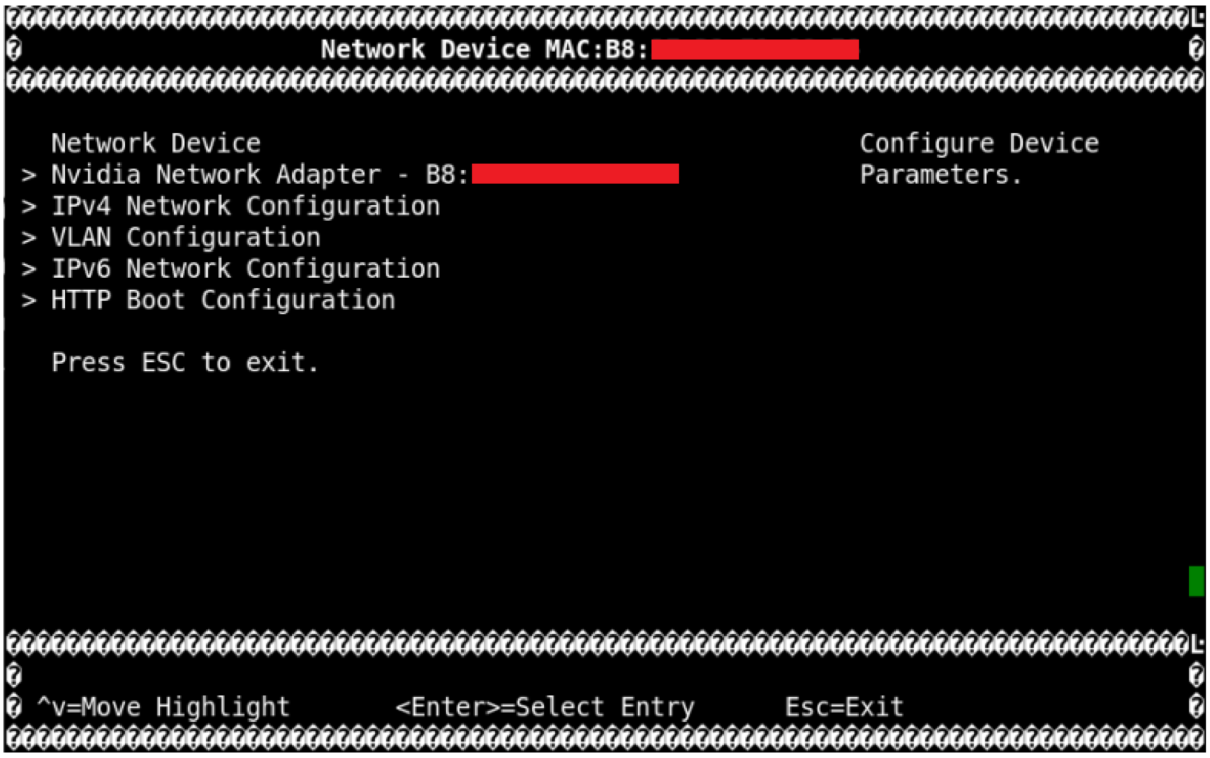
2.1.6 Network Device List

Lists the MAC addresses of the available network interfaces in UEFI.



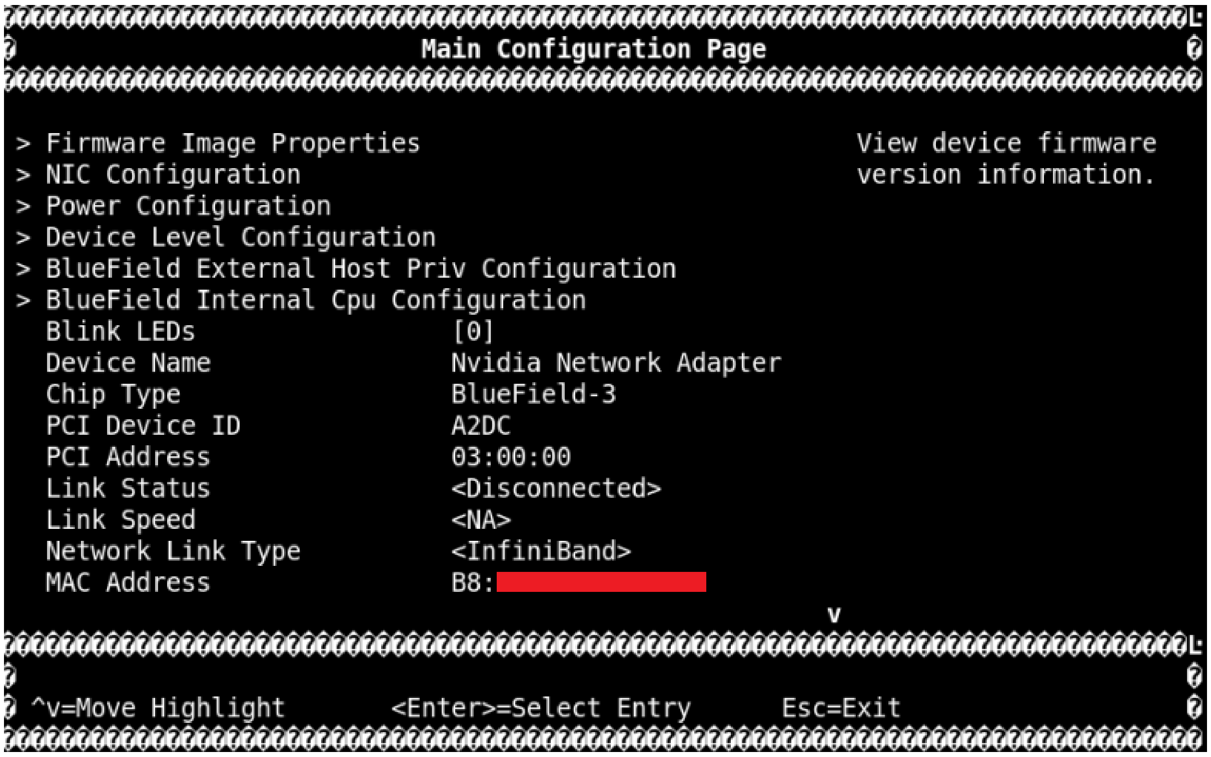
```
                              Network Device List

   Network Device List                                   Network Device
 > MAC:B8:[redacted]
 > MAC:00:[redacted]
 > MAC:B8:[redacted]
 > MAC:B8:[redacted]

   Press ESC to exit.

  ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

Users can find more information (Link status, Link speed, PCI ID, Link type, etc.) on each interface upon selection. Users can also configure the interfaces (IPv4, IPv6, VLAN, HTTP BOOT) as needed.



```
                         Network Device MAC:B8:[redacted]

   Network Device                                        Configure Device
 > Nvidia Network Adapter - B8:[redacted]                Parameters.
 > IPv4 Network Configuration
 > VLAN Configuration
 > IPv6 Network Configuration
 > HTTP Boot Configuration

   Press ESC to exit.

  ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

The following menu can be reached by selecting the `Nvidia Network Adapter - <mac-address>` menu options:



```
                         Main Configuration Page

> Firmware Image Properties                          View device firmware
> NIC Configuration                                  version information.
> Power Configuration
> Device Level Configuration
> BlueField External Host Priv Configuration
> BlueField Internal Cpu Configuration
  Blink LEDs                 [0]
  Device Name                Nvidia Network Adapter
  Chip Type                  BlueField-3
  PCI Device ID              A2DC
  PCI Address                03:00:00
  Link Status                <Disconnected>
  Link Speed                 <NA>
  Network Link Type          <InfiniBand>
  MAC Address                B8:[redacted]
                                                   v

 ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

2.2 System Configuration and Services

This page provides information on system services and scripts based on the default DPU OS (i.e., Ubuntu).

2.2.1 First Boot After BFB Installation

During the first boot, the cloud-init service configures the system based on the data provided in the following files:

- `/var/lib/cloud/seed/nocloud-net/network-config` - network interface configuration
- `/var/lib/cloud/seed/nocloud-net/user-data` - default users and commands to run on the first boot

2.2.2 RDMA and ConnectX Driver Initialization

RDMA and NVIDIA® ConnectX® drivers are loaded upon boot by the `openibd.service`.

The `mlx5_core` kernel module is loaded automatically by the kernel as a registered device driver.

One of the kernel modules loaded by the `openibd.service`, `ib_umad`, triggers modprobe rule from `/etc/modprobe.d/mlnx-bf.conf` file that runs the `/sbin/mlnx_bf_configure` script. See Default Ports and OVS Configuration for more information.

2.2.3 Firewall Configuration

The BFB image includes the following firewall configuration (enabled by default):

```
$ cat /etc/iptables/rules.v4

*mangle
:PREROUTING ACCEPT [45:3582]
:INPUT ACCEPT [45:3582]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [36:4600]
:POSTROUTING ACCEPT [36:4600]
:KUBE-IPTABLES-HINT - [0:0]
:KUBE-KUBELET-CANARY - [0:0]
COMMIT
*filter
:INPUT ACCEPT [41:3374]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [32:3672]
:DOCKER-USER - [0:0]
:KUBE-FIREWALL - [0:0]
:KUBE-KUBELET-CANARY - [0:0]
:LOGGING - [0:0]
:POSTROUTING - [0:0]
:PREROUTING - [0:0]
-A INPUT -j KUBE-FIREWALL
-A INPUT -p tcp -m tcp --dport 111 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -p udp -m udp --dport 111 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -i lo -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -d 127.0.0.0/8 -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m state --state RELATED,ESTABLISHED -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp ! --dport 22 ! --tcp-flags FIN,SYN,RST,ACK SYN -m mark --mark 0xb -m state --state NEW -m
comment --comment MD_IPTABLES -j DROP
-A INPUT -f -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,PSH,ACK,URG -m mark --mark 0xb -m comment --
comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -m mark --mark 0xb -m comment --comment MD_IPTABLES
-j DROP
-A INPUT -m mark --mark 0xb -m state --state INVALID -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --tcp-flags RST RST -m mark --mark 0xb -m hashlimit --hashlimit-above 2/sec --hashlimit-
burst 2 --hashlimit-mode srcip --hashlimit-name hashlimit_0 --hashlimit-htable-expire 30000 -m comment --comment
MD_IPTABLES -j DROP
-A INPUT -p tcp -m mark --mark 0xb -m state --state NEW -m hashlimit --hashlimit-above 50/sec --hashlimit-burst 50
--hashlimit-mode srcip --hashlimit-name hashlimit_1 --hashlimit-htable-expire 30000
-m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m mark --mark 0xb -m conntrack --ctstate NEW -m hashlimit --hashlimit-above 60/sec --hashlimit-
burst 20 --hashlimit-mode srcip --hashlimit-name hashlimit_2 --hashlimit-htable-expire 30000 -m comment --comment
MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m recent --rcheck --seconds 86400 --name portscan --mask 255.255.255.255 --rsource -m
comment --comment MD_IPTABLES -j DROP
-A INPUT -m mark --mark 0xb -m recent --remove --name portscan --mask 255.255.255.255 --rsource -m comment --
comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 50 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP

-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 10 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --set --name DEFAULT --
mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW -m recent --update --seconds 60 --
hitcount 100 --name DEFAULT --mask 255.255.255.255 --rsource -m comment --comment MD_IPTABLES -j DROP
-A INPUT -p tcp -m tcp --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 179 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 68 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 122 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 161 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 6306 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 69 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 389 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 389 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1812:1813 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --
comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 49 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
```

```
-A INPUT -p tcp -m tcp --dport 49 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --sport 53 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --sport 53 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 500 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 4500 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1293 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 1293 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 1707 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 1707 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -i lo -p udp -m udp --dport 3786 -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment MD_IPTABLES
-j ACCEPT
-A INPUT -i lo -p udp -m udp --dport 33000 -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment MD_IPTABLES
-j ACCEPT
-A INPUT -p icmp -m mark --mark 0xb -m comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --sport 5353 --dport 5353 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m
comment --comment MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 33434:33523 -m mark --mark 0xb -m comment --comment MD_IPTABLES -j REJECT --reject-
with icmp-port-unreachable
-A INPUT -p udp -m udp --dport 123 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 514 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p udp -m udp --dport 67 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
MD_IPTABLES -j ACCEPT
-A INPUT -p tcp -m tcp --dport 60102 -m mark --mark 0xb -m conntrack --ctstate NEW,ESTABLISHED -m comment --comment
"MD_IPTABLES: Feature HA port" -j ACCEPT
-A INPUT -m mark --mark 0xb -m comment --comment MD_IPTABLES -j LOGGING
-A FORWARD -j DOCKER-USER
-A OUTPUT -o oob_net0 -m comment --comment MD_IPTABLES -j ACCEPT
-A DOCKER-USER -j RETURN

-A LOGGING -m mark --mark 0xb -m comment --comment MD_IPTABLES -j NFLOG --nflog-prefix  "IPTables-Dropped: " --
nflog-group 3
-A LOGGING -m mark --mark 0xb -m comment --comment MD_IPTABLES -j DROP
-A PREROUTING -i oob_net0 -m comment --comment MD_IPTABLES -j MARK --set-xmark 0xb/0xffffffff
-A PREROUTING -p tcp -m tcpmss ! --mss 536:65535 -m tcp ! --dport 22 -m mark --mark 0xb -m conntrack --ctstate NEW
-m comment --comment MD_IPTABLES -j DROP
COMMIT
*nat
:PREROUTING ACCEPT [1:320]
:INPUT ACCEPT [1:320]
:OUTPUT ACCEPT [8:556]
:POSTROUTING ACCEPT [8:556]
:KUBE-KUBELET-CANARY - [0:0]
:KUBE-MARK-DROP - [0:0]
:KUBE-MARK-MASQ - [0:0]
:KUBE-POSTROUTING - [0:0]
-A POSTROUTING -m comment --comment "kubernetes postrouting rules" -j KUBE-POSTROUTING
-A KUBE-MARK-DROP -j MARK --set-xmark 0x8000/0x8000
-A KUBE-MARK-MASQ -j MARK --set-xmark 0x4000/0x4000
-A KUBE-POSTROUTING -m mark ! --mark 0x4000/0x4000 -j RETURN
-A KUBE-POSTROUTING -j MARK --set-xmark 0x4000/0x0
-A KUBE-POSTROUTING -m comment --comment "kubernetes service traffic requiring SNAT" -j MASQUERADE --random-fully
COMMIT
```

This configuration is provided by the `bf-release` package and is installed during the first boot of the Ubuntu OS after the BFB installation using the `cloud-init` service and the `/var/lib/cloud/seed/nocloud-net/user-data` configuration file.

To disable this default firewall configuration after OS is UP, run:

```
$ rm -f /etc/iptables/rules.v4
$ iptables -F
```

To disable this default firewall configuration during the BFB installation, use `bf.cfg` with the following command in the `bfb_modify_os` function:

```
bfb_modify_os()
{
    perl -ni -e "if(/^write_files:/../^users/) {next unless m{^users}; print} else {print}" /mnt/var/lib/cloud/
seed/nocloud-net/user-data
}
```

2.3 Host-side Interface Configuration

The NVIDIA® BlueField® DPU registers on the host OS a "DMA controller" for DPU management over PCIe. This can be verified by running the following:

```
#  lspci -d 15b3: | grep 'SoC Management Interface'
27:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

A special driver called RShim must be installed and run to expose the various BlueField management interfaces on the host OS. Refer to section "Install RShim on Host" for information on how to obtain and install the host-side RShim driver.

When the RShim driver runs properly on the host side, a sysfs device, `/dev/rshim0/*`, and a virtual Ethernet interface, `tmfifo_net0`, become available. The following is an example for querying the status of the RShim driver on the host side:

```
# systemctl status rshim
● rshim.service - rshim driver for BlueField SoC
     Loaded: loaded (/lib/systemd/system/rshim.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-05-31 14:57:07 IDT; 1 day 1h ago
       Docs: man:rshim(8)
    Process: 90322 ExecStart=/usr/sbin/rshim $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 90323 (rshim)
      Tasks: 11 (limit: 76853)
     Memory: 3.3M
     CGroup: /system.slice/rshim.service
               90323 /usr/sbin/rshim
May 31 14:57:07 …  systemd[1]: Starting rshim driver for BlueField SoC...
May 31 14:57:07  … systemd[1]: Started rshim driver for BlueField SoC.
May 31 14:57:07  … rshim[90323]: Probing pcie-0000:a3:00.2(vfio)
May 31 14:57:07  … rshim[90323]: Create rshim pcie-0000:a3:00.2
May 31 14:57:07  … rshim[90323]: rshim pcie-0000:a3:00.2 enable
May 31 14:57:08  … rshim[90323]: rshim0 attached
```

If the RShim device does not appear, refer to section "RShim Troubleshooting and How-Tos".

2.3.1 Virtual Ethernet Interface

On the host, the RShim driver exposes a virtual Ethernet device called `tmfifo_net0`. This virtual Ethernet can be thought of as a peer-to-peer tunnel connection between the host and the DPU OS. The DPU OS also configures a similar device. The DPU OS's BFB images are customized to configure the DPU side of this connection with a preset IP of 192.168.100.2/30. It is up to the user to configure the host side of this connection. Configuration procedures vary for different OSs.

The following example configures the host side of `tmfifo_net0` with a static IP and enables IPv4-based communication to the DPU OS:

```
#  ip addr add dev tmfifo_net0 192.168.100.1/30
```

For instructions on persistent IP configuration of the tmfifo_net0 interface, refer to step "Assign a static IP to tmfifo_net0" under "Updating Repo Package on Host Side".

Logging in from the host to the DPU OS is now possible over the virtual Ethernet. For example:

```
ssh ubuntu@192.168.100.2
```

2.3.2 RShim Support for Multiple DPUs

Multiple DPUs may connect to the same host machine. When the RShim driver is loaded and operating correctly, each board is expected to have its own device directory on sysfs, `/dev/rshim<N>`, and a virtual Ethernet device, `tmfifo_net<N>`.

The following are some guidelines on how to set up the RShim virtual Ethernet interfaces properly if multiple DPUs are installed in the host system.

There are two methods to manage multiple `tmfifo_net` interfaces on a Linux platform:

- Using a bridge, with all `tmfifo_net<N>` interfaces on the bridge - the bridge device bares a single IP address on the host while each DPU has unique IP in the same subnet as the bridge
- Directly over the individual `tmfifo_net<N>` - each interface has a unique subnet IP and each DPU has a corresponding IP per subnet

Whichever method is selected, the host-side `tmfifo_net` interfaces should have different MAC addresses, which can be:

- Configured using `ifconfig`. For example:

  ```
  $ ifconfig tmfifo_net0 192.168.100.1/24 hw ether 02:02:02:02:02:02
  ```

- Or saved in configuration via the `/udev/rules` as can be seen later in this section.

In addition, each Arm-side `tmfifo_net` interface must have a unique MAC and IP address configuration, as BlueField OS comes uniformly pre-configured with a generic MAC, and 192.168.100.2. The latter must be configured in each DPU manually or by DPU customization scripts during BlueField OS installation.

2.3.2.1 Multi-board Management Example

This example deals with two BlueField DPUs installed on the same server (the process is similar for more DPUs).

This example assumes that the RShim package has been installed on the host server.

2.3.2.1.1 Configuring Management Interface on Host

This example is relevant for CentOS/RHEL operating systems only.

1. Create a `bf_tmfifo` interface under `/etc/sysconfig/network-scripts`. Run:

   ```
   vim /etc/sysconfig/network-scripts/ifcfg-br_tmfifo
   ```

2. Inside `ifcfg-br_tmfifo`, insert the following content:

   ```
   DEVICE="br_tmfifo"
   BOOTPROTO="static"
   IPADDR="192.168.100.1"
   NETMASK="255.255.255.0"
   ```

```
ONBOOT="yes"
TYPE="Bridge"
```

3. Create a configuration file for the first BlueField DPU, `tmfifo_net0`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

4. Inside `ifcfg-tmfifo_net0`, insert the following content:

```
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

5. Create a configuration file for the second BlueField DPU, `tmfifo_net1`. Run:

```
DEVICE=tmfifo_net1
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

6. Create the rules for the `tmfifo_net` interfaces. Run:

```
vim /etc/udev/rules.d/91-tmfifo_net.rules
```

7. Restart the network for the changes to take effect. Run:

```
# /etc/init.d/network restart
Restarting network (via systemctl):             [  OK  ]
```

2.3.2.1.2 Configuring BlueField DPU Side

BlueField DPUs arrive with the following factory default configurations for tmfifo_net0.

Address	Value
MAC	`00:1a:ca:ff:ff:01`
IP	`192.168.100.2`

Therefore, if you are working with more than one DPU, you must change the default MAC and IP addresses.

2.3.2.1.2.1 Updating RShim Network MAC Address

This procedure is relevant for Ubuntu/Debian (`sudo` needed), and CentOS BFBs. The procedure only affects the `tmfifo_net0` on the Arm side.

1. Use a Linux console application (e.g. screen or minicom) to log into each BlueField. For example:

```
# sudo screen /dev/rshim<0|1>/console 115200
```

2. Create a configuration file for `tmfifo_net0` MAC address. Run:

```
# sudo vi /etc/bf.cfg
```

3. Inside `bf.cfg`, insert the new MAC:

```
NET_RSHIM_MAC=00:1a:ca:ff:ff:03
```

4. Apply the new MAC address. Run:

```
sudo bfcfg
```

5. Repeat this procedure for the second BlueField DPU (using a different MAC address).

   Arm must be rebooted for this configuration to take effect. It is recommended to update the IP address before you do that to avoid unnecessary reboots.

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

2.3.2.1.2.2 Updating IP Address

For Ubuntu:

1. Access the file `50-cloud-init.yaml` and modify the tmfifo_net0 IP address:

```
sudo vim /etc/netplan/50-cloud-init.yaml

                tmfifo_net0:
                     addresses:
                     - 192.168.100.2/30     ===>>>    192.168.100.3/30
```

2. Reboot the Arm. Run:

```
sudo reboot
```

3. Repeat this procedure for the second BlueField DPU (using a different IP address).

   Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

For CentOS:

1. Access the file `ifcfg-tmfifo_net0`. Run:

```
# vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

2. Modify the value for `IPADDR`:

```
IPADDR=192.168.100.3
```

3. Reboot the Arm. Run:

```
reboot
```

Or perform `netplan apply`.

4. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

2.3.3 Permanently Changing Arm-side MAC Address

> It is assumed that the commands in this section are executed with root (or `sudo`) permission.

The default MAC address is `00:1a:ca:ff:ff:01`. It can be changed using `ifconfig` or by updating the UEFI variable as follows:

1. Log into Linux from the Arm console.
2. Run:

```
$ "ls /sys/firmware/efi/efivars".
```

3. If not mounted, run:

```
$ mount -t efivarfs none /sys/firmware/efi/efivars
$ chattr -i /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
$ printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > \
  /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

The `printf` command sets the MAC address to `00:1a:ca:ff:ff:03` (the last six bytes of the `printf` value). Either reboot the device or reload the tmfifo driver for the change to take effect.

The MAC address can also be updated from the server host side while the Arm-side Linux is running:

1. Enable the configuration. Run:

```
# echo "DISPLAY_LEVEL 1" > /dev/rshim0/misc
```

2. Display the current setting. Run:

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   1 (0:basic, 1:advanced, 2:log)
BOOT_MODE       1 (0:rshim, 1:emmc, 2:emmc-boot-swap)
BOOT_TIMEOUT    300 (seconds)
DROP_MODE       0 (0:normal, 1:drop)
SW_RESET        0 (1: reset)
DEV_NAME        pcie-0000:04:00.2
DEV_INFO        BlueField-2(Rev 1)
PEER_MAC        00:1a:ca:ff:ff:01 (rw)
PXE_ID          0x00000000 (rw)
VLAN_ID         0 0 (rw)
```

3. Modify the MAC address. Run:

```
$ echo "PEER_MAC  xx:xx:xx:xx:xx:xx" > /dev/rshim0/misc
```

For more information and an example of the script that covers multiple DPU installation and configuration, refer to section "Installing Full DOCA Image on Multiple DPUs" of the *NVIDIA DOCA Installation Guide*.

2.3.4 OOB Ethernet Interface

The OOB interface is a gigabit Ethernet interface which provides TCP/IP network connectivity to the Arm cores. This interface is named `oob_net0` and is intended to be used for management traffic (e.g. file transfer protocols, SSH, etc). The Linux driver that controls this interface is named `mlxbf_gige.ko`, and is automatically loaded upon boot. This interface can be configured and monitored by use of standard tools (e.g. ifconfig, ethtool, etc). The OOB interface is subject to the following design limitations:

- Only supports 1Gb/s full-duplex setting
- Only supports GMII access to external PHY device
- Supports maximum packet size of 2KB (i.e. no support for jumbo frames)

The OOB interface can also be used for PXE boot. This OOB port is not a path for the boot stream. Any attempt to push a BFB to this port will not work. Please refer to How to use the UEFI boot menu for more information about UEFI operations related to the OOB interface.

2.3.4.1 OOB Interface MAC Address

The MAC address to be used for the OOB port is burned into Arm-accessible UPVS EEPROM during the manufacturing process. This EEPROM device is different from the SPI Flash storage device used for the NIC firmware and associated NIC MACs/GUIDs. The value of the OOB MAC address is specific to each platform and is visible on the board-level sticker.

It is not recommended to reconfigure the MAC address from the MAC configured during manufacturing.

If there is a need to re-configure this MAC for any reason, follow these steps to configure a UEFI variable to hold new value for OOB MAC.:

The creation of an OOB MAC address UEFI variable will override the OOB MAC address defined in EEPROM, but the change can be reverted.

1. Log into Linux from the Arm console.
2. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

   ```
   mount -t efivarfs none /sys/firmware/efi/efivars
   ```

3. Run:

   ```
   chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```

4. Set the MAC address to 00:1a:ca:ff:ff:03 (the last six bytes of the printf value).

```
printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > /sys/firmware/efi/efivars/
OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

5. Reboot the device for the change to take effect.

To revert this change and go back to using the MAC as programmed during manufacturing, follow these steps:

1. Log into UEFI from the Arm console, go to "Boot Manager" then "EFI Internal Shell".
2. Delete the OOB MAC UEFI variable. Run:

   ```
   dmpstore -d OobMacAddr
   ```

3. Reboot the device by running "reset" from UEFI.
4. Log into Linux from the Arm console.
5. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

   ```
   mount -t efivarfs none /sys/firmware/efi/efivars
   ```

6. Run:

   ```
   chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```

7. Reconfigure the original MAC address burned by the manufacturer in the format `aa\bb\cc\dd\ee\ff`. Run:

   ```
   printf "\x07\x00\x00\x00\x00\<original-MAC-address>" > /sys/firmware/efi/efivars/
   OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```

8. Reboot the device for the change to take effect.

2.3.4.2 Supported ethtool Options for OOB Interface

The Linux driver for the OOB port supports the handling of some basic ethtool requests: get driver info, get/set ring parameters, get registers, and get statistics.

To use the ethtool options available, use the following format:

```
$ ethtool [<option>] <interface>
```

Where `<option>` may be:

- `<no-argument>` - display interface link information
- `-i` - display driver general information
- `-S` - display driver statistics
- `-d` - dump driver register set
- `-g` - display driver ring information
- `-G` - configure driver ring(s)
- `-k` - display driver offload information
- `-a` - query the specified Ethernet device for pause parameter information
- `-r` - restart auto-negotiation on the specified Ethernet device if auto-negotiation is enabled

For example:

```
$ ethtool oob_net0
Settings for oob_net0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  1000baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 3
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Link detected: yes
```

```
$ ethtool -i oob_net0
driver: mlxbf_gige
version:
firmware-version:
expansion-rom-version:
bus-info: MLNXBF17:00
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: yes
supports-priv-flags: no
```

```
# Display statistics specific to BlueField-2 design (i.e. statistics that are not shown in the output of "ifconfig
oob0_net")
$ ethtool -S oob_net0
NIC statistics:
    hw_access_errors: 0
    tx_invalid_checksums: 0
    tx_small_frames: 1
    tx_index_errors: 0
    sw_config_errors: 0
    sw_access_errors: 0
    rx_truncate_errors: 0
    rx_mac_errors: 0
    rx_din_dropped_pkts: 0
    tx_fifo_full: 0
    rx_filter_passed_pkts: 5549
    rx_filter_discard_pkts: 4
```

2.3.4.3 IP Address Configuration for OOB Interface

The files that control IP interface configuration are specific to the Linux distribution. The udev rules file (`/etc/udev/rules.d/92-oob_net.rules`) that renames the OOB interface to `oob_net0` and is the same for Yocto, CentOS, and Ubuntu:

```
SUBSYSTEM=="net", ACTION=="add", DEVPATH=="/devices/platform/MLNXBF17:00/net/eth[0-9]", NAME="oob_net0"
```

The files that control IP interface configuration are slightly different for CentOS and Ubuntu:

- CentOS configuration of IP interface:
    - Configuration file for `oob_net0`: `/etc/sysconfig/network-scripts/ifcfg-oob_net0`
    - For example, use the following to enable DHCP:

```
NAME="oob_net0"
DEVICE="oob_net0"
NM_CONTROLLED="yes"
PEERDNS="yes"
```

```
ONBOOT="yes"
BOOTPROTO="dhcp"
TYPE=Ethernet
```

- For example, to configure static IP use the following:

```
NAME="oob_net0"
DEVICE="oob_net0"
IPV6INIT="no"
NM_CONTROLLED="no"
PEERDNS="yes"
ONBOOT="yes"
BOOTPROTO="static"
IPADDR="192.168.200.2"
PREFIX=30
GATEWAY="192.168.200.1"
DNS1="192.168.200.1"
TYPE=Ethernet
```

- For Ubuntu configuration of IP interface, refer to section "Default Network Interface Configuration".

2.4 Secure Boot

These pages provide guidelines on how to operate secured NVIDIA® BlueField® DPUs. They provide UEFI secure boot references for the UEFI portion of the secure boot process.

This section provides directions for illustration purposes, it does not intend to enforce or mandate any procedure about managing keys and/or production guidelines. Platform users are solely responsible of implementing secure strategies and safe approaches to manage their boot images and their associated keys and certificates.

Security aspects such as key generation, key management, key protection, and certificate generation are out of the scope of this section.

Secure boot is a process which verifies each element in the boot process prior to execution, and halts or enters a special state if a verification step fails at any point during the boot. It is based on an unmodifiable ROM code which acts as the root-of-trust (RoT) and uses an off-chip public key, to authenticate the initial code which is loaded from an external non-volatile storage. The off-chip public key integrity is verified by the ROM code against an on-chip public key hash value stored in E-FUSEs. Then the authenticated code and each element in the boot process cryptographically verify the next element prior to passing execution to it. This extends the chain-of-trust (CoT) by verifying elements that have their RoT in hardware. In addition, no external intervention in the authentication process is permitted to prevent unauthorized software and firmware from being loaded. There should be no way to interrupt or bypass the RoT with runtime changes.

2.4.1 Supported BlueField DPUs

Secured BlueField devices have pre-installed software and firmware signed with NVIDIA signing keys. The on-chip public key hash is programmed into E-FUSEs.

To verify whether the DPU in your possession supports secure boot, run the following command:

```
# sudo mst start
# sudo flint -d /dev/mst/mt41686_pciconf0 q full | grep "Life cycle"
Life cycle:             GA SECURED
```

“GA SECURED” indicates that the BlueField device has secure boot enabled.

To verify whether the BlueField Arm has secure boot enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mlxbf-bootctl | grep lifecycle
lifecycle state: GA Secured
```

2.4.2 UEFI Secure Boot

This feature is available in the NVIDIA® BlueField®-2 and above.

UEFI Secure Boot is a feature of the Unified Extensible Firmware Interface (UEFI) specification. The feature defines a new interface between the operating system and firmware/BIOS.

When enabled and fully configured on the DPU, UEFI Secure Boot helps the Arm=based software running on top of UEFI resist attacks and infection from malware. UEFI Secure Boot detects tampering with boot loaders, key operating system files, and unauthorized option ROMs by validating their digital signatures. Malicious actions are blocked from running before they can attack or infect the system.

UEFI Secure Boot works as a security gate. Code signed with valid keys (whose public key/certificates exist in the DPU) gets through the gate and executes while blocking and rejecting code that has either a bad or no signature.

The DPU enables UEFI secure boot with the Ubuntu OS included in the platform's software.

2.4.2.1 Verifying UEFI Secure Boot on DPU

To verify whether UEFI secure boot is enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mokutil --sb-state
SecureBoot enabled
```

As UEFI secure boot is not specific to BlueField platforms, please refer to the Canonical documentation online for further information on UEFI secure boot:

- https://wiki.ubuntu.com/UEFI/SecureBoot
- https://wiki.ubuntu.com/UEFI/SecureBoot/Signing

2.4.2.2 Main Use Cases for UEFI Secure Boot

UEFI secure boot can be used in 2 main cases for the DPU:

Method	Pros	Cons
Using the default enabled UEFI secure boot (with Ubuntu OS or any Microsoft-signed boot loader) See "Using Default Enabled UEFI Secure Boot" for more.	Relatively easy	Limited flexibility; only allows executing NVIDIA binary files Dependency on Microsoft or NVIDIA as signing entities

Method	Pros	Cons
Enabling UEFI Secure Boot with a custom OS (other than the default Ubuntu) See "Enabling UEFI Secure Boot with Custom OS" for more.	Autonomy, as you control your own keys (no dependency on Microsoft or NVIDIA as signing entities)	You must create your own capsule files to enroll and customize UEFI secure boot

Signing binaries is complex as you must create X.509 certificates and enroll them in UEFI or shim which requires a fair amount of prior knowledge of how secure boot works. For that reason, BlueField secured platforms are shipped with all the needed certificates and signed binaries (which allows working seamlessly with the first use case in the table above).

NVIDIA strongly recommends utilizing UEFI secure boot in any case due the increased security it enables.

2.4.2.2.1 Verifying UEFI Secure Boot on DPU

To verify whether UEFI secure boot is enabled, run the following command from the BlueField console:

```
ubuntu@localhost:~$ sudo mokutil --sb-state
SecureBoot enabled
```

As UEFI secure boot is not specific to BlueField platforms, refer to the Canonical documentation online for further information on UEFI secure boot to familiarize yourself with the UEFI secure boot concept:

- https://wiki.ubuntu.com/UEFI/SecureBoot
- https://wiki.ubuntu.com/UEFI/SecureBoot/Signing

2.4.2.3 Using Default Enabled UEFI Secure Boot

As part of the default settings of the DPU, UEFI secure boot is enabled and requires no special configuration from the user to use it with the bundled Ubuntu OS.

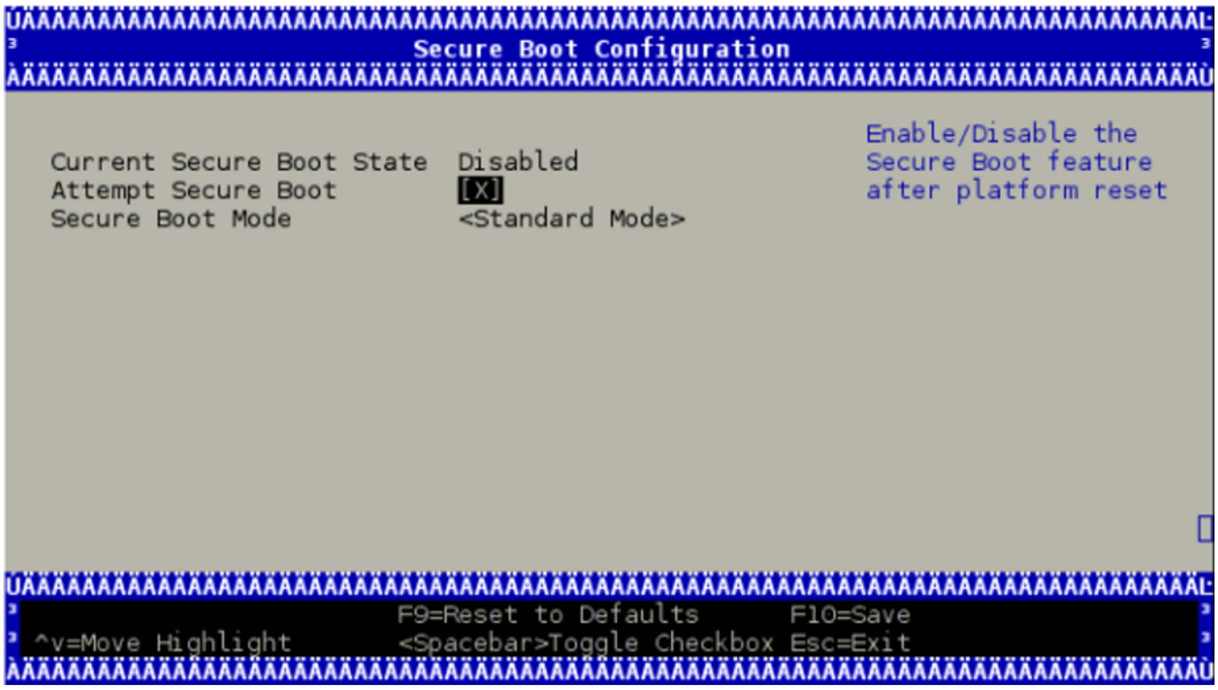
2.4.2.3.1 Disabling UEFI Secure Boot

UEFI secure boot can be disabled per device from the UEFI menu as part of the DPU boot process which requires access to the BlueField console.

To disable UEFI secure boot, reboot the platform and stop at the UEFI menu.

> On BlueField devices with UEFI secure boot enabled, the UEFI menu is password-protected to prevent unwanted changes to the UEFI settings. The default password is `bluefield`.

From the UEFI menu screen, select "Device Manager" then "Secure Boot Configuration". If "Attempt Secure Boot" is checked, then uncheck it and reboot.



```
                    Secure Boot Configuration

  Current Secure Boot State   Disabled               Enable/Disable the
  Attempt Secure Boot         [X]                    Secure Boot feature
  Secure Boot Mode            <Standard Mode>        after platform reset


                              F9=Reset to Defaults       F10=Save
  ^v=Move Highlight           <Spacebar>Toggle Checkbox Esc=Exit
```

Disabling secure boot permanently is not recommended in production environments.

It is also possible to disable UEFI secure boot using Redfish API for DPUs with an on-board BMC. For more details, please refer to your NVIDIA sales representative to receive the *NVIDIA BlueField DPU Initial Deployment Guide*.

2.4.2.3.2 Existing DPU Certificates

As part of having UEFI secure boot enabled, the UEFI databases are populated with NVIDIA self-signed X.509 certificates. The Microsoft certificate is also installed into the UEFI database to ensure that the Ubuntu distribution can boot while UEFI secure boot is enabled (and generally any suitable OS loader signed by Microsoft).

The pre-installed certificate files are:

- NVIDIA PK key certificate
- NVIDIA KEK key certificate
- NVIDIA db certificate
- Microsoft db certificate

2.4.2.4 Enabling UEFI Secure Boot with Custom OS

This section lists the required steps to enable using UEFI secure boot with a custom OS (other than the default Ubuntu).

All processes described in the following subsections require some level of testing and knowledge in how operating system boot flows and bootloaders work.

2.4.2.4.1 Options for Enabling UEFI Secure Boot

There are 3 main ways for signing custom binaries and running them on the DPU with UEFI secure boot enabled:

#	Method	Pros	Cons
1	Sign OS loader (e.g., Shim) by Microsoft. See "Signing OS Loader by Microsoft" for more.	Does not require access to the BlueField console	Dependency on Microsoft as signing entity
2	Shim - enroll a machine owner key (MOK) certificate in the shim and use the private part to sign your files. See "Enrolling MOK Key" for more.	Easy	• Limited flexibility: Only allows executing a custom kernel or load a custom module. It does not allow executing UEFI applications, UEFI drivers, or OS loaders. • Dependency on Microsoft or NVIDIA as signing entities • Not scalable: Requires access to BlueField console per device (i.e., UART console required)
3	UEFI - enroll your own key certificate in the UEFI database and use the private part to sign your files. See "Enrolling Your Own Key to UEFI DB" for more.	Autonomy, as you control your keys (not dependent on Microsoft or NVIDIA as signing entities)	• Requires adding your key certificate to database manually • Requires access to BlueField console per device (i.e., UART console required) • Not scalable: Requires access to BlueField console per device (i.e., UART console required)

For generation of custom keys and certificates, see section "Generation of Custom Keys and Certificates".

Signing binaries for UEFI secure boot is complex as you must create X.509 certificates and enroll them in UEFI or shim which requires a fair amount of prior knowledge of how secure boot works. See the processes used to enroll keys and to sign UEFI binaries in the rest of this document.

Secure booting binaries for executing a UEFI application, UEFI driver, OS loader, custom kernel, or loading a custom module depends on the certificates and public keys available in the UEFI database and the shim's MOK list.

2.4.2.4.2 Signing OS Loader by Microsoft

2.4.2.4.2.1 Custom Kernel Images

One option to boot custom binaries on a DPU is to sign the OS loader (shim) by Microsoft following the Microsoft guidelines which are updated and maintained by Microsoft. The certificates/keys must be embedded within the shim OS loader so it may boot, in addition the custom Kernel binary image and the custom Kernel modules must be signed accordingly.

2.4.2.4.2.2 NVIDIA Kernel Modules

In this option, the NVIDIA db certificates should remain enrolled. This is due to the out-of-tree kernel modules and drivers (e.g., OFED) provided by NVIDIA which are signed by NVIDIA and authenticated by this NVIDIA certificate in the UEFI.

> Signing binaries with Microsoft is a process the involves lead time which must be taken into consideration. This course of action requires testing to making sure the complied BFB image including the signed Microsoft bootloader works properly.

2.4.2.4.3 Enrolling MOK Key

To boot a custom kernel or load a custom module, you must create a MOK key pair. The newly created MOK key must be an RSA 2048-bit. The private part is used for signing operations and must be kept safe. The public X.509 key certificate in DER format must be enrolled within the shim MOK list.

Once the public key certificate is enrolled within the shim, the MOK key is accepted as a valid signing key.

Note that kernel module signing requires a special configuration. For example, the `extendedKeyUsage` field must show an OID of 1.3.6.1.4.1.2312.16.1.2. That OID informs shim that this is meant to be a module signing certificate.

The following is an example of OpenSSL configuration file for illustration purposes:

```
HOME                        = .
RANDFILE                    = $ENV::HOME/.rnd
[ req ]
distinguished_name          = req_distinguished_name
x509_extensions             = v3
string_mask                 = utf8only
prompt                      = no

[ req_distinguished_name ]
countryName                 = US
stateOrProvinceName         = Westborough
localityName                = Massachusetts
0.organizationName          = CampanyX
commonName                  = Secure Boot Signing
emailAddress                = example@example.com

[ v3 ]
subjectKeyIdentifier        = hash
authorityKeyIdentifier      = keyid:always,issuer
basicConstraints            = critical,CA:FALSE
extendedKeyUsage            = codeSigning,1.3.6.1.4.1.311.10.3.6,1.3.6.1.4.1.2312.16.1.2
nsComment                   = "OpenSSL Generated Certificate"
```

To enroll the MOK key certificate, download the associated key certificate to the BlueField file system and run the following command:

```
ubuntu@localhost:~$ sudo mokutil --import mok.der
input password:
input password again:
```

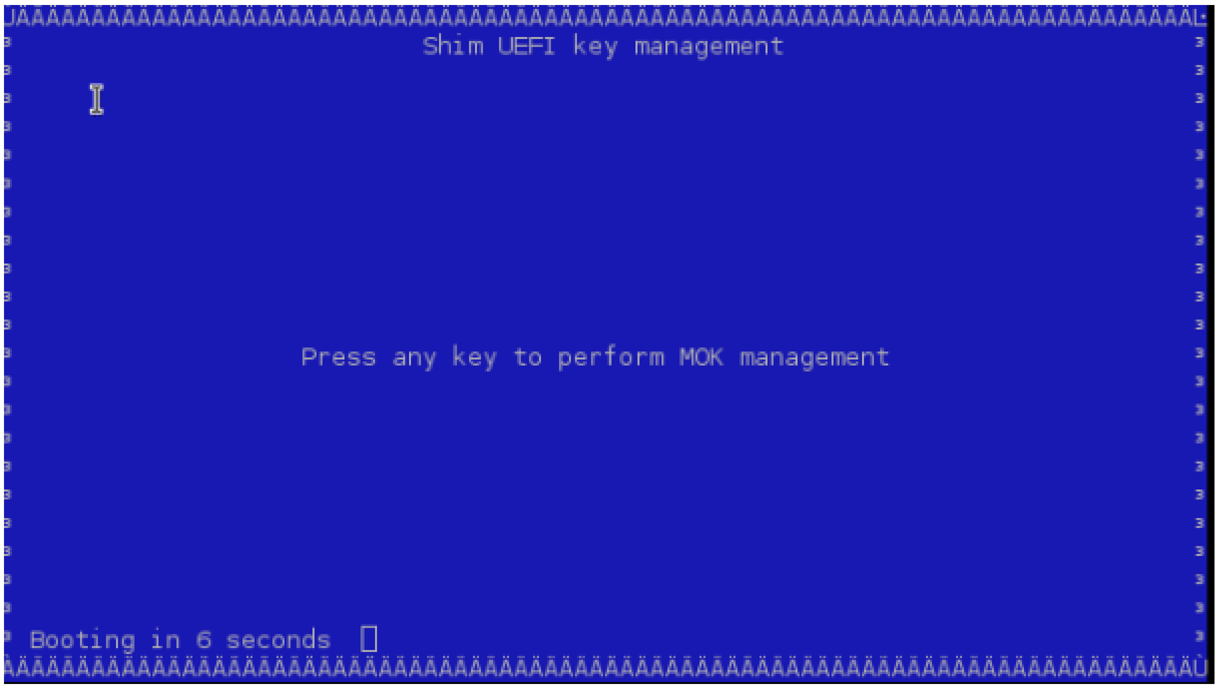
You must follow the prompts to enter a password to be used to make sure you really do want to enroll the key certificate.

Note that the key certificate is not enrolled yet. It will be enrolled by the shim upon the next reboot. To list the imported certificate file to enroll:

```
ubuntu@localhost:~$ sudo mokutil --list-new
```

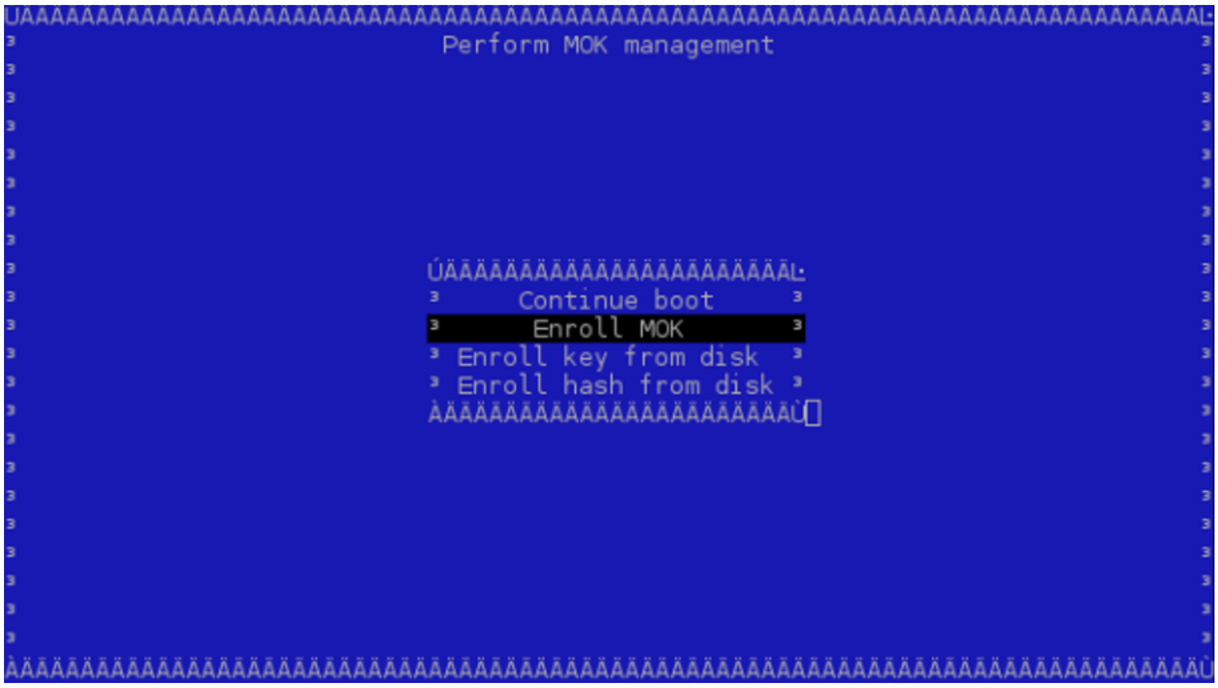
A reboot must be performed.

Just before loading GRUB, shim displays a blue screen which is actually another piece of the shim project called "MokManager". You may ignore the blue screen showing the error message. Press "OK" to enter the "Shim UEFI key management" screen.



```
Shim UEFI key management

Press any key to perform MOK management

Booting in 6 seconds
```

Select "Enroll MOK" and follow the menus to finish the enrolling process.



```
Perform MOK management

Continue boot
Enroll MOK
Enroll key from disk
Enroll hash from disk
```

You may look at the properties of the key you are adding to make sure it is indeed correct using "View key". MokManager will ask for the same password you typed in earlier when running mokutil before reboot. MokManager will save the key and you will need to reboot again.

To list the enrolled certificate files, run the following command:

```
ubuntu@localhost:~$ sudo mokutil --list-enrolled
```

2.4.2.4.4 Generation of Custom Keys and Certificates

To boot binaries not signed with the existing public keys and certificates in the UEFI database (like the Microsoft certificate and key described in "Signing OS Loader by Microsoft"), create an X.509 certificate (which includes the public key part of the public-private key pair) that can be imported either directly though the UEFI or, more easily, via shim.

Creating a certificate and public key for use in the UEFI secure boot is relatively simple. OpenSSL can do it by running the command `req`.

For illustration purposes only, this example shows how to create a 2048-bit RSA MOK key and its associated certificate file in DER format:

```
$ openssl req -new -x509 -newkey rsa:2048 -nodes -days 36500 -outform DER -keyout "mok.priv" -out "mok.der"
```

An OpenSSL configuration file may be used for key generation. It may be specified using `--config path/to/openssl.cnf`.

> Detailed key and certificate generation are beyond the scope of this document. Any organization should choose the proper way to generate keys and certificates based on their security policy.

The following sections refer to the db private key as `key.priv` and its DER certificate as `cert.der`. Similarly, the MOK private key is referred to as `mok.priv` and its DER certificate as `mok.der`.

2.4.2.4.5 Enrolling Your Own Key to UEFI DB

Some users may need to generate their own keys. For convenience, the processes used to enroll keys into UEFI db as well as to sign UEFI binaries are provided in this document.

To execute your binaries while UEFI secure boot is enabled, you need your own pair of private and public key certificates. The supported keys are RSA 2048-bit and ECDSA 384-bit.

The private part is used for signing operations and must be kept safe. The public part X.509 key certificate in DER format must be enrolled within the UEFI db.

A prerequisite for the following steps is having UEFI secure boot temporarily disabled on the DPU. After temporarily disabling UEFI secure boot per device as in section "Existing DPU Certificates", it is possible to override all the key certificate files of the UEFI database. This allows you to enroll your PK key certificate, KEK key certificate, and db certificates.

The following subsections detail how enrolling can be done.

2.4.2.4.5.1 Using a Capsule

To enroll your key certificates, create a capsule file by way of tools and scripts provided along with the BlueField software.

To create the capsule files, execute the `mlx-mkcap` script. After BlueField software installation, the script can be found under `/lib/firmware/mellanox/boot/capsule/scripts`. This script generates a capsule file to supply the key certificates to UEFI and enables UEFI secure boot:

```
$ ./mlx-mkcap --pk-key pk.cer --kek-key kek.cer --db-key db.cer EnrollYourKeysCap
```

Note that you may specify as many db certificates as needed using the `--db-key` flag. In this example, only a single db certificate is specified.

To set the UEFI password, you may specify the `--uefi-passwd` flag. For example, to set the UEFI password to `bluefield`, run:

```
$ ./mlx-mkcap --pk-key pk.cer --kek-key kek.cer --db-key db.cer --uefi-passwd "bluefield" EnrollYourKeysCap
```

The resulting capsule file, `EnrollYourKeysCap`, can be downloaded to the BlueField file system to initiate the key enrollment process. From the the BlueField console execute the following command then reboot:

```
ubuntu@localhost:~$ bfrec --capsule EnrollYourKeysCap
```

On the next reboot, the capsule file is processed and the UEFI database is populated with the keys extracted from the capsule file.

Enrolling the PK key certificate file enables the UEFI secure boot.

2.4.2.4.5.2 Enroll Certificate into UEFI DB

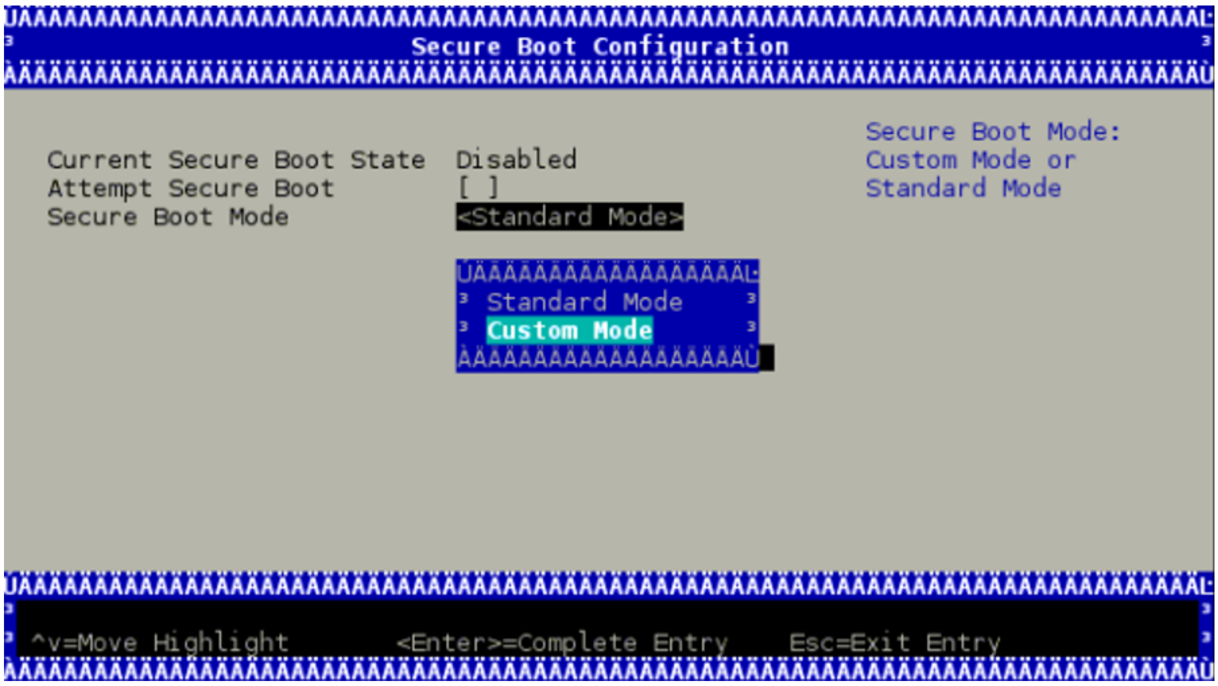
As mentioned, the public part of the X.509 key certificate in DER format must be enrolled within the UEFI db. The X.509 DER certificate file must be installed into the EFI system partition (ESP).

Download the certificate file to BlueField file system and place it into the ESP:

```
ubuntu@localhost:~$ sudo cp path/to/cert.der /boot/efi/
```

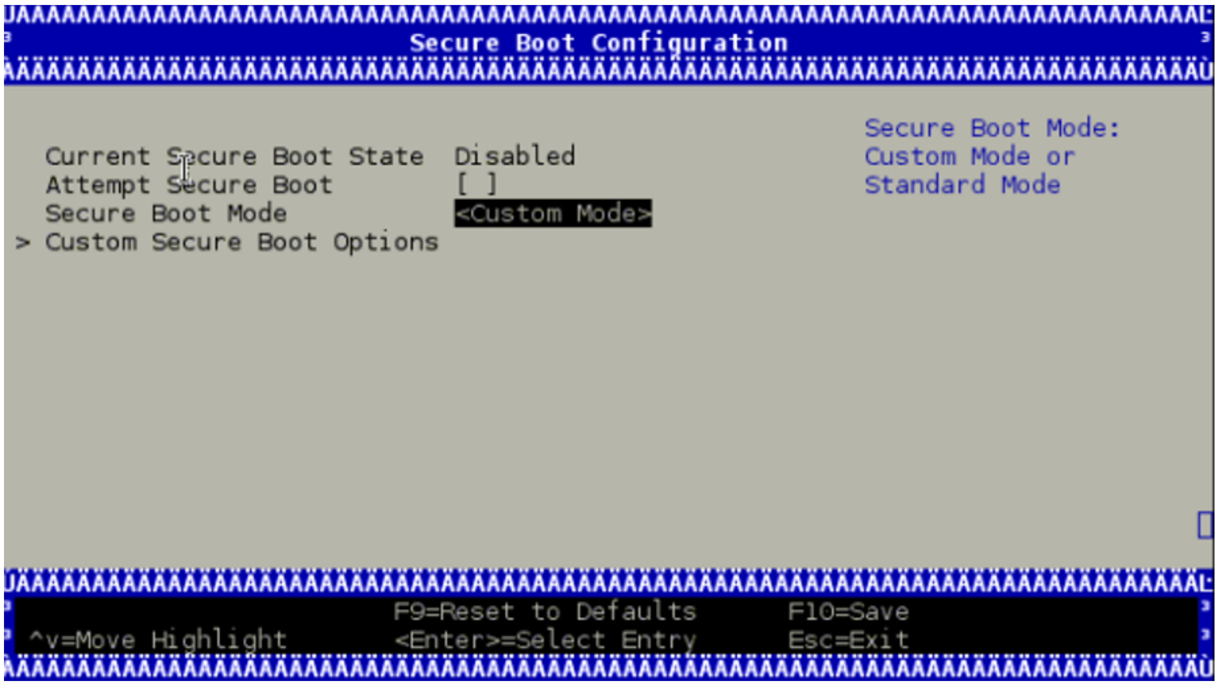
To enroll the certificate into the UEFI db, you must to reboot and log in again into the UEFI menu. From the "UEFI menu", select "Device Manager" entry then "Secure Boot Configuration". Navigate to "Secure Boot Mode" and select "Custom Mode" setup.

The secure boot "Custom Mode" setup feature allows a physically present user to modify the UEFI database.



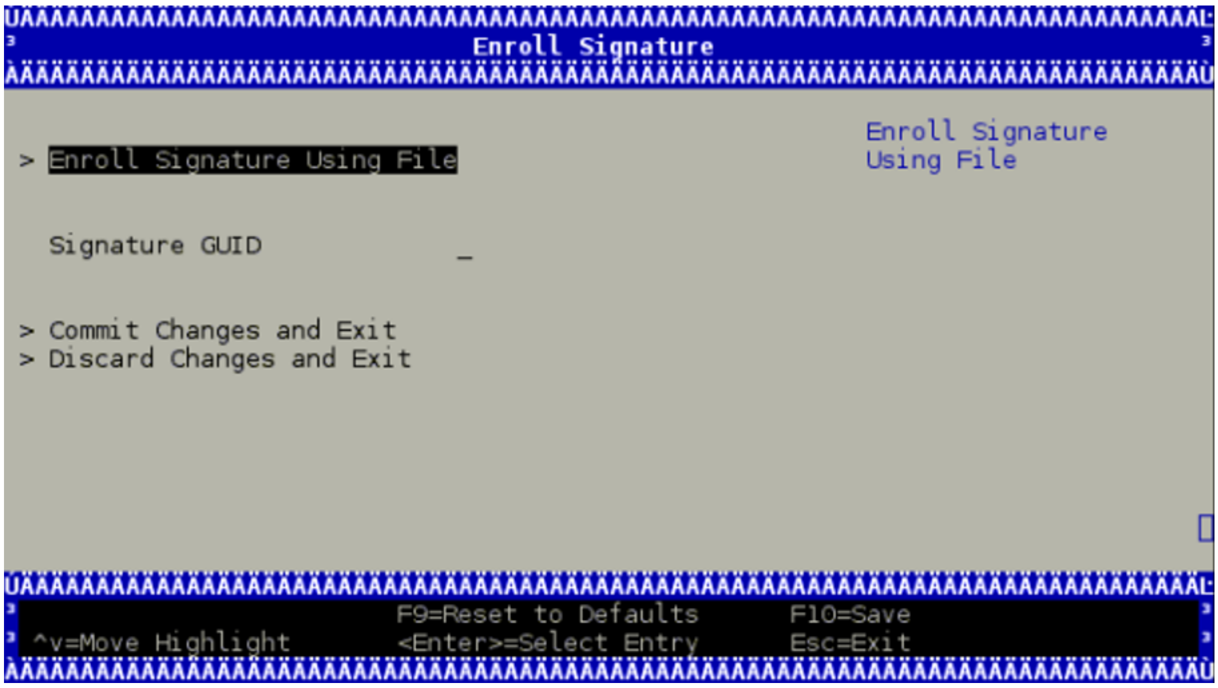
```
                    Secure Boot Configuration

  Current Secure Boot State  Disabled              Secure Boot Mode:
  Attempt Secure Boot        [ ]                   Custom Mode or
  Secure Boot Mode           <Standard Mode>       Standard Mode

                               Standard Mode
                               Custom Mode

 ^v=Move Highlight      <Enter>=Complete Entry     Esc=Exit Entry
```

Once the platform is in "Custom Mode", a "Custom Secure Boot Options" menu entry appears which allows you to manipulate the UEFI database keys and certificates.



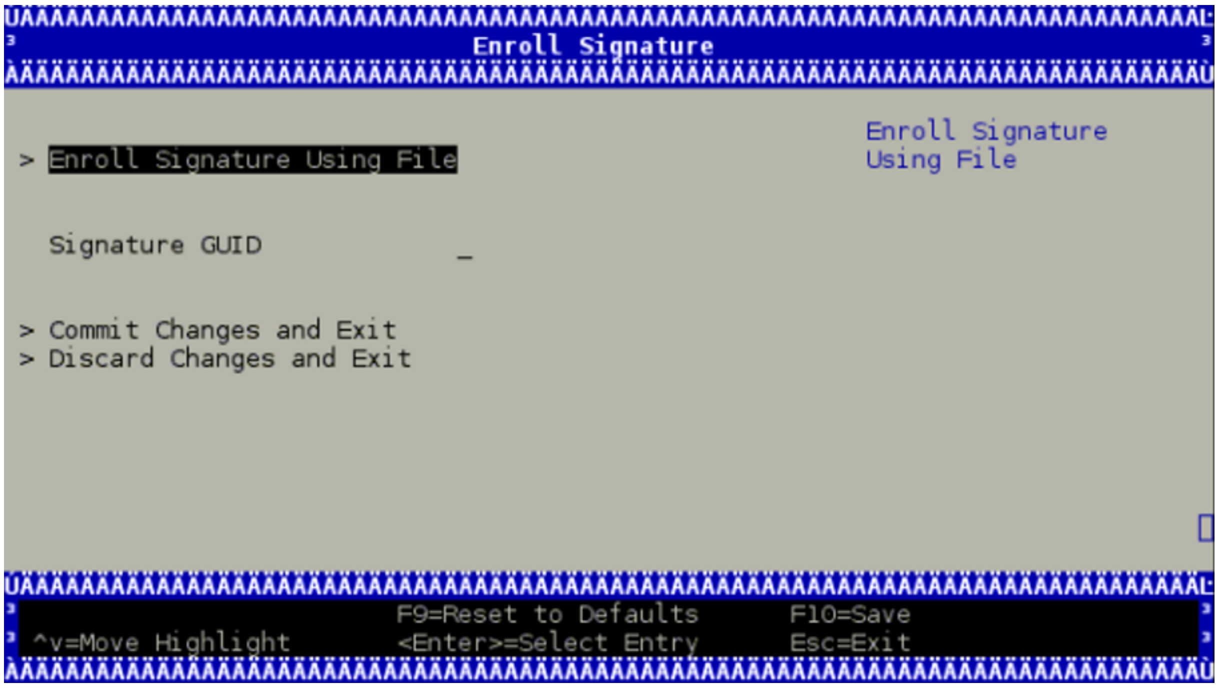
```
                    Secure Boot Configuration

  Current Secure Boot State  Disabled              Secure Boot Mode:
  Attempt Secure Boot        [ ]                   Custom Mode or
  Secure Boot Mode           <Custom Mode>         Standard Mode
> Custom Secure Boot Options

                        F9=Reset to Defaults       F10=Save
 ^v=Move Highlight      <Enter>=Select Entry       Esc=Exit
```

To enroll your DER certificate file, select "DB Options" and enter the "Enroll Signature" menu. Select "Enroll Signature Using File" and navigate within the EFI System Partition (ESP) to the db DER certificate file.



```
                        Enroll Signature

> Enroll Signature Using File                      Enroll Signature
                                                   Using File

  Signature GUID              _

> Commit Changes and Exit
> Discard Changes and Exit

                   F9=Reset to Defaults     F10=Save
 ^v=Move Highlight <Enter>=Select Entry     Esc=Exit
```

The ESP path is shown below as "system-boot, [VenHw(*)/HD(*)]".



```
                        Enroll Signature

> Enroll Signature Using File                      Enroll Signature
                                                   Using File

  Signature GUID              _

> Commit Changes and Exit
> Discard Changes and Exit

                   F9=Reset to Defaults     F10=Save
 ^v=Move Highlight <Enter>=Select Entry     Esc=Exit
```

While enrolling the certificate file, you may enter a GUID along with the key certificate file. The GUID is the platform's way of identifying the key. It serves no purpose other than for you to tell which key is which when you delete it (it is not used at all in signature verification).

This value must be in the following format: 11111111-2222-3333-4444-1234567890ab. If nothing is entered, a GUID of 00000000-0000-0000-0000-000000000000 is created.

Finally, commit the changes and exit. You may be asked to reboot.

2.4.2.5 Signing Binaries

2.4.2.5.1 Signing Custom Kernel and UEFI Binaries

To sign a custom kernel or any other EFI binary (UEFI application, UEFI driver or OS loader) you want to have loaded by shim, you need the private part of the key and the certificate in PEM format.

To convert the certificate into PEM, run:

```
$ openssl x509 -in mok.der -inform DER -outform PEM -out mok.pem
```

Now, to sign your EFI binary, run:

```
$ sbsign --key mok.priv --cert mok.pem binary.efi --output binary.efi.signed
```

If you are using your db key, use the private part of the key and its associated certificate converted into PEM format for binary signing.

If the X.509 key certificate is enrolled in UEFI db or by way of shim, the binary should be loaded without an issue.

2.4.2.5.2 Signing Kernel Modules

The X.509 certificate you added must be visible to the kernel. To verify the keys visible to the kernel, run:

```
ubuntu@localhost:~$ sudo cat /proc/keys
```

For a straightforward result, run:

```
ubuntu@localhost:~$ dmesg | grep -i "X.509"
[    1.869521] Loading compiled-in X.509 certificates
[    1.875441] Loaded X.509 cert 'Build time autogenerated kernel key: b1a3fbd0178bdb7190387a4187e8e4b0eb476cdc'
[    1.941752] integrity: Loading X.509 certificate: UEFI:db
[    1.947636] integrity: Loaded X.509 cert 'YourSigningDbKey: a109f01707ba6769c4d546530ba1592c7daedc3b'
[    1.958736] integrity: Loading X.509 certificate: UEFI:db
[    1.964170] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011:
13adbf4309bd82709c8cd54f316ed522988a1bd4'
[    2.023740] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.030090] integrity: Loaded X.509 cert 'YourSingingMokKey: 2012e5122669ffc0cc28827c6134329a6bec0b88'
[    2.040796] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.046830] integrity: Loaded X.509 cert 'SomeOrg: shim: 331c1c8963538e327d6e39346f4f53b200987015'
[    2.055796] integrity: Loading X.509 certificate: UEFI:MokListRT
[    2.062114] integrity: Loaded X.509 cert 'Canonical Ltd. Master Certificate Authority:
ad91990bc22ab1f517048c23b6655a268e345a63'
```

If the X.509 certificate attributes (`commonName` , etc.) are configured properly, you should see your key certificate information in the result output. In this example, two custom keys are visible to the kernel:

- `YourSigningMokKey` - registered with the shim as a MOK
- `YourSigningDbKey` - registered with UEFI as db

This example is for illustration purposes only. The actual output might differ from the output shown in this example depending on what key was previously enrolled and how it was enrolled.

You may sign kernel modules using either of these approaches:

- `kmodsign` command
- Linux kernel script sign-file

2.4.2.5.2.1 Signing Kernel Modules Using kmodsign

If you are using the `kmodsign` command to sign kernel modules, run:

```
ubuntu@localhost:~$ sudo cat /proc/keys
```

The signature is appended to the kernel module by `kmodsign`.

But if you rather keep the original kernel module unchanged, run:

```
ubuntu@localhost:~$ kmodsign sha512 mok.priv mok.der module.ko module-signed.ko
```

Refer to `kmosign --help` for more information.

2.4.2.5.2.2 Signing Kernel Modules Using Sign File

To sign the kernel module using the Linux kernel script sign-file, please refer to Linux kernel documentation.

If you are using your db key, use the private part of the key and its associated certificate for binary signing.

To validate that the module is signed, check that it includes the string `~Module signature appended~`:

```
ubuntu@localhost:~$ hexdump -Cv module.ko | tail -n 5
00002c20  10 14 08 cd eb 67 a8 3d  ac 82 e1 1d 46 b5 5c 91  |.....g.=....F.\.|
00002c30  9c cb 47 f7 c9 77 00 00  02 00 00 00 00 00 00 00  |..G..w..........|
00002c40  02 9e 7e 4d 6f 64 75 6c  65 20 73 69 67 6e 61 74  |..~Module signat|
00002c50  75 72 65 20 61 70 70 65  6e 64 65 64 7e 0a        |ure appended~.|
00002c5e
```

2.4.2.5.3 Ongoing Updates

2.4.2.5.3.1 Update Key Certificates

This requires UEFI secure boot to have been enabled using your own keys, which means that you own the signing keys.

While UEFI secure boot is enabled, it is possible to update your keys using a capsule file.

To create a capsule intended to update the UEFI secure boot keys, generate a new set of keys and then run:

```
$ ./mlx-mkcap --pk-key new_pk.cer --kek-key new_kek.cer --db-key new_db1.cer --db-key new_db2.cer --db-key
new_db3.cer --signer-key db.key --signer-cert db.pem EnrollYourNewKeysCap
```

Note that `--signer-key` and `--signer-cert` are set so the capsule is signed. When UEFI secure boot is enabled, the capsule is verified using the key certificates previously enrolled in the UEFI database. It is important to use the old signing keys associated with the certificates in the UEFI database to sign the capsule. The new key certificates are intended to replace the existing key certificates after capsule processing. Once the UEFI database is updated, the new keys must be used to sign the newly created capsule files.

To enroll the new set of keys, download the capsule file to the BlueField console and use `bfrec` to initiate the capsule update.

2.4.2.5.3.2 Disable UEFI Secure Boot Using a Capsule

> This requires UEFI secure boot to have been enabled using your own keys, which means that you own the signing keys.

It is possible to disable UEFI secure boot through a capsule update. This requires an empty PK key when creating the capsule file.

To create a capsule intended to disable UEFI secure boot:

1. Create a dummy empty PK certificate:

   ```
   $ touch null_pk.cer
   ```

2. Create the capsule file:

   ```
   $ ./mlx-mkcap --pk-key null_pk.cer --signer-key db.key --signer-cert db.pem DeletePkCap
   ```

`--signer-key` and `--signer-cert` must be specified with the appropriate private keys and certificates associated with the actual key certificates in the UEFI database.

To enroll the empty PK certificate, download the capsule file to the BlueField console and use `bfrec` to initiate the capsule update.

> Deleting the PK certificate will result in UEFI secure boot to be disabled which is not recommended in a production environment.

2.4.3 Updating Platform Firmware

To update the platform firmware on secured devices, download the latest NVIDIA® BlueField® software images from NVIDIA.com.

2.4.3.1 Updating eMMC Boot Partitions Image

The capsule file `/lib/firmware/mellanox/boot/capsule/MmcBootCap` is used to update the eMMC boot partition and update the Arm pre-boot code (i.e., Arm trusted firmware and UEFI).

The capsule file is signed with NVIDIA keys. If UEFI secure boot is enabled, make sure the NVIDIA certificate files are enrolled into the UEFI database. Please refer to "UEFI Secure Boot" for more information on how to update the UEFI database key certificates.

To initiate the update of the eMMC boot partitions, run the following command:

```
ubuntu@localhost:~$ sudo bfrec --capsule /lib/firmware/mellanox/boot/capsule/MmcBootCap
```

After the command completes, reboot the system to process the capsule file. On the next reboot, UEFI will verify the capsule signature. If verified, UEFI will process the capsule file, extract the pre-boot image and burn it into the eMMC boot partitions.

Note that the pre-boot code is signed with the NVIDIA key. The bootloader images are installed into the eMMC with their associated certificate files. The public key is derived from the certificate file and its integrity is verified by the ROM code against an on-chip public key hash value stored in E-FUSEs. If the verification fails, then the pre-boot code will not be allowed to execute.

2.4.3.1.1 Recovering eMMC Boot Partition

If the system cannot boot from the eMMC boot partitions for any reason, it is recommended to download a valid BFB image and boot it over the BlueField platform.

The recovery path relies on the platform to be configured to boot solely from the RShim interface (either RShim USB or RShim PCIe). With this configuration there must not be a way to interrupt or bypass the RoT when secure booting.

You will need to append a capsule file to the BFB prior to booting. Run:

```
$ mlx-mkbfb --capsule MmcBootCap install.bfb recovery_install.bfb
```

Then boot the `recovery_install.bfb` using the RShim interface. Run:

```
$ cat recovery_install.bfb > /dev/rshim0/boot
```

The capsule file will be processed by UEFI upon boot.

2.4.3.2 Updating SPI Flash FS4 Image

The SPI flash contains the firmware image of the DPU firmware in FS4 format. The firmware image is provided along with the software.

There are two different ways to install the firmware image:

- From the BlueField console, using the following command:

  ```
  ubuntu@localhost:~$ /opt/mellanox/mlnx-fw-updater/firmware/mlxfwmanager_sriov_dis_aarch64_<bf-dev>
  ```

- From the PCIe host console, using the following command:

  ```
  # flint -d /dev/mst/mt<bf-dev>_pciconf0 -i firmware.bin b
  ```

`bf-dev` is 41686 for BlueField-2 or 41692 for BlueField-3.

2.5 Default Passwords and Policies

2.5.1 BMC Passwords

The BMC password must comply with the following policy parameters:

- Using ASCII and Unicode characters is permitted
- Minimum length: 12
- Maximum length: 20
- Maximum number of consecutive character pairs: 4

> Two characters are consecutive if `|hex(char_1)-hex(char_2)|=1`.
>
> Examples of passwords with 5 consecutive character pairs (invalid): `DcB` `a123456AbCd!`; `ab1XbcYcdZdeGef!`; `Testing_123abcgh!`.

The following is a valid example password:

- `HelloNvidia3D!`

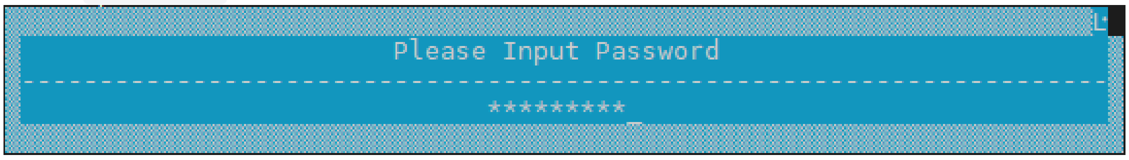
> A user account is locked for 10 minutes after 10 consecutive failed attempts.

2.5.2 UEFI Menu Password

A password is required to enter the UEFI menu during BlueField bootup. The UEFI menu contains various settings which impact BlueField behavior. Therefore, it is very important to keep that password secure.

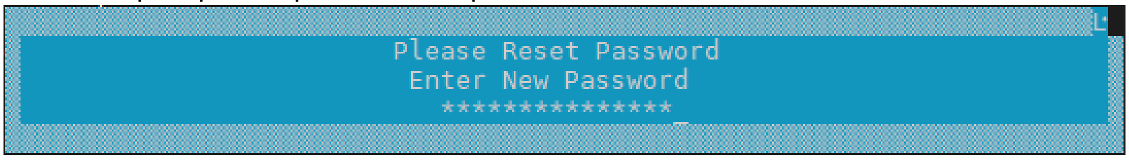
2.5.2.1 Default Password

1. A first-time user accessing the UEFI menu must enter the default password for the UEFI menu, `bluefield`:



```
Please Input Password
----------------------------------------------------------------------
*********_
```

2. The user is prompted to provide a new password:

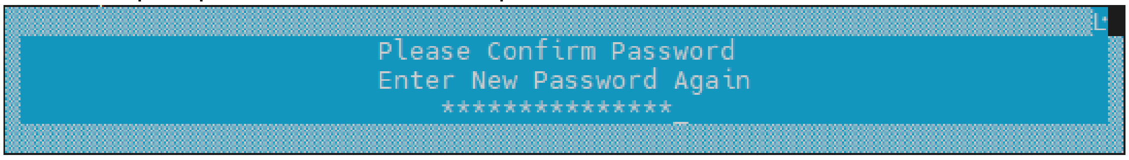


```
Please Reset Password
Enter New Password
***************_
```

The new password entered above must be in compliance with the password policy:

- The password must be between 12 and 64 characters (inclusive)
- There are no requirements for upper/lower case, or special characters. Spaces are allowed.

3. The user is prompted to confirm the new password:



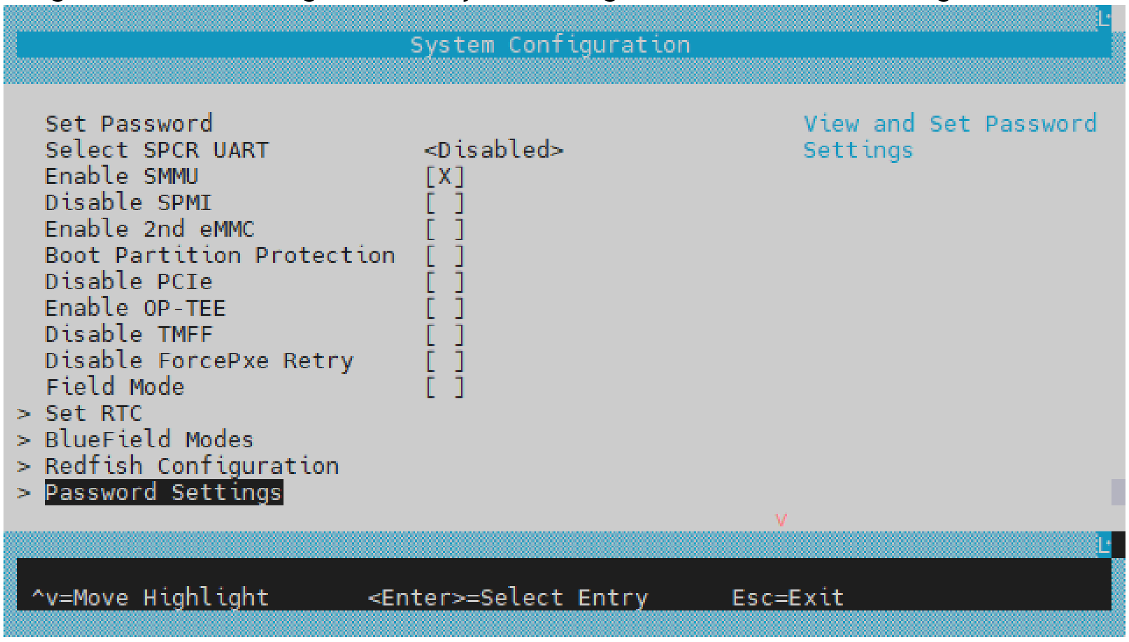
```
Please Confirm Password
Enter New Password Again
****************
```

2.5.2.2 Default Password Policy

The user can enable/disable the UEFI password policy. The default password policy is enabled by default using a checkbox in the UEFI menu.

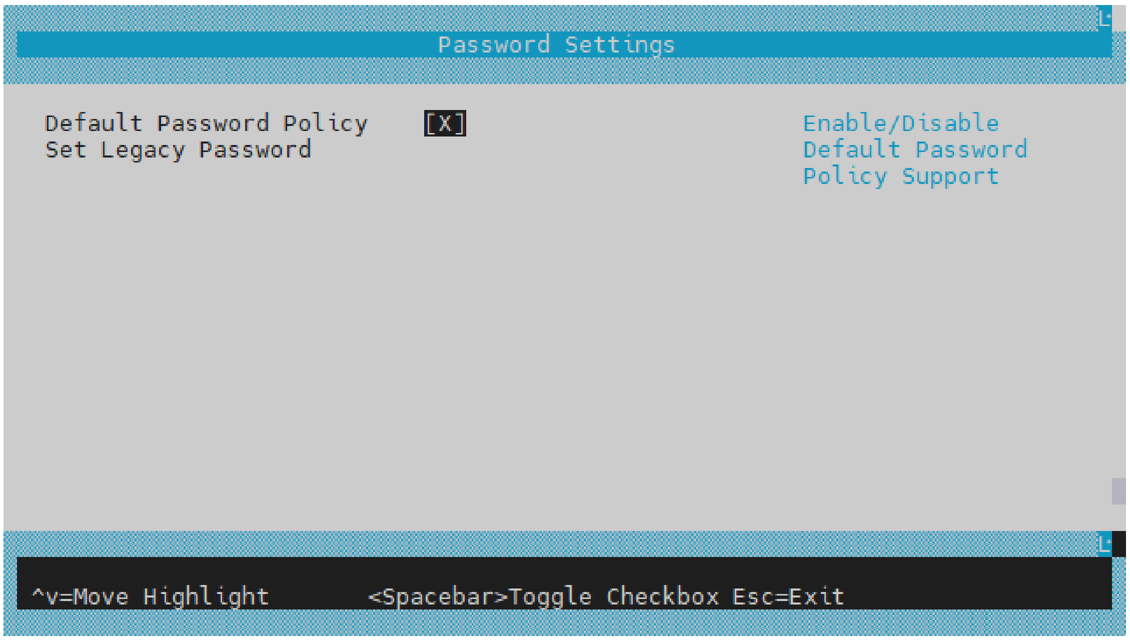
The user can browse the UEFI menu and disable as follows:

1. Navigate to "Device Configuration" > "System Configuration" > "Password Settings":



```
                         System Configuration

  Set Password                                         View and Set Password
  Select SPCR UART              <Disabled>             Settings
  Enable SMMU                   [X]
  Disable SPMI                  [ ]
  Enable 2nd eMMC               [ ]
  Boot Partition Protection     [ ]
  Disable PCIe                  [ ]
  Enable OP-TEE                 [ ]
  Disable TMFF                  [ ]
  Disable ForcePxe Retry        [ ]
  Field Mode                    [ ]
> Set RTC
> BlueField Modes
> Redfish Configuration
> Password Settings

 ^v=Move Highlight       <Enter>=Select Entry       Esc=Exit
```

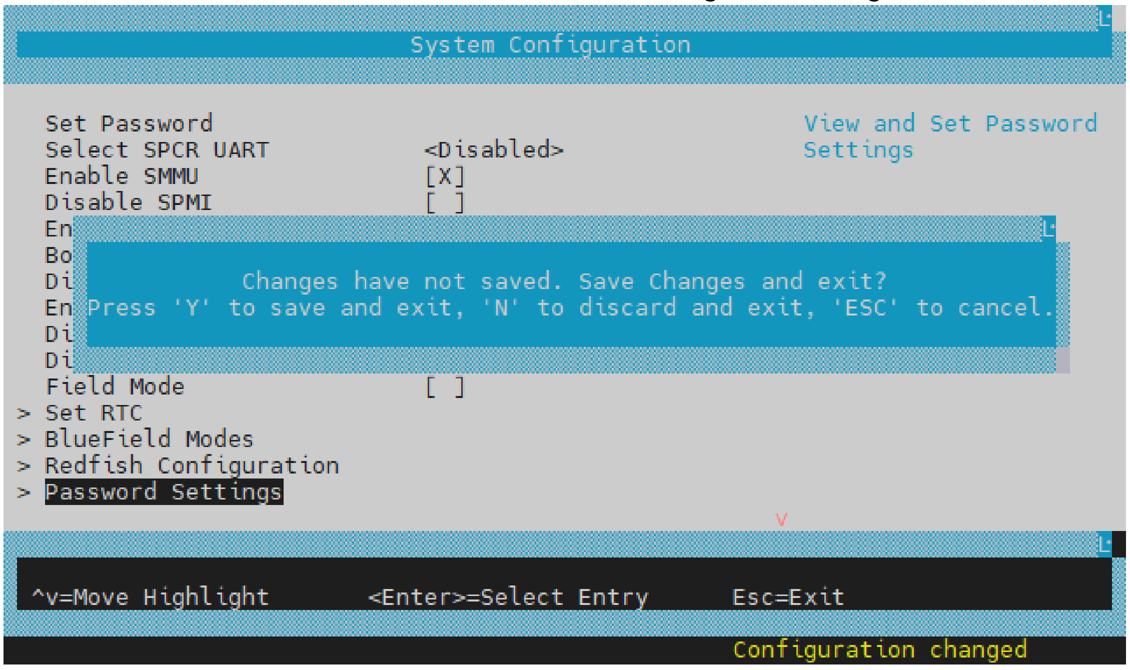
2. The "Default Password Policy" checkbox controls whether the more secure password policy is enabled:



```
                         Password Settings

  Default Password Policy     [X]                        Enable/Disable
  Set Legacy Password                                    Default Password
                                                         Policy Support

^v=Move Highlight      <Spacebar>Toggle Checkbox Esc=Exit
```

To disable the Default Password Policy, hit the spacebar to clear the checkbox.

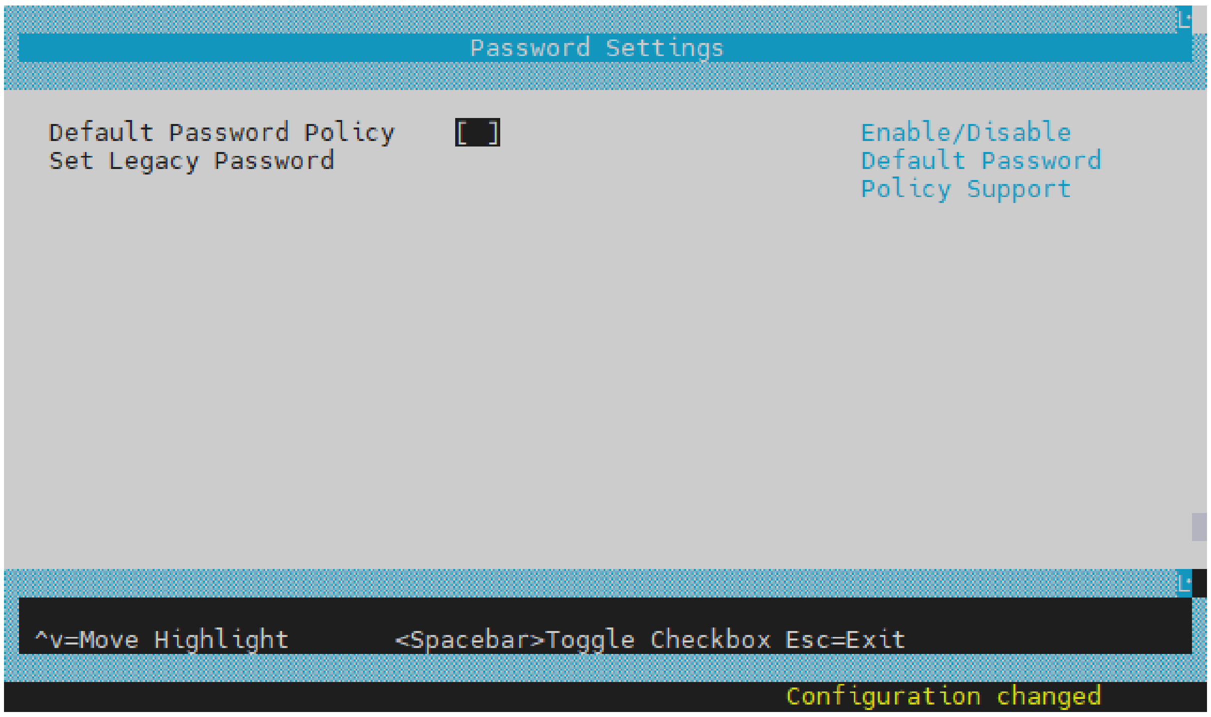
3. The user must hit ESC ESC and answer "Y" to save the configuration change.



```
                       System Configuration

  Set Password                                           View and Set Password
  Select SPCR UART           <Disabled>                  Settings
  Enable SMMU                [X]
  Disable SPMI               [ ]
  En
  Bo
  Di           Changes have not saved. Save Changes and exit?
  En Press 'Y' to save and exit, 'N' to discard and exit, 'ESC' to cancel.
  Di
  Di
  Field Mode                 [ ]
> Set RTC
> BlueField Modes
> Redfish Configuration
> Password Settings

^v=Move Highlight      <Enter>=Select Entry      Esc=Exit
                                                  Configuration changed
```

2.5.2.2.1 Disabling Default Password Policy

To disable the Default Password Policy, hit the spacebar to clear the checkbox.


```
Password Settings

Default Password Policy     [ ]          Enable/Disable
Set Legacy Password                      Default Password
                                         Policy Support

^v=Move Highlight       <Spacebar>Toggle Checkbox Esc=Exit
                                  Configuration changed
```

If the Default Password Policy is disabled, the password entered must be between 1 and 64 characters.

2.5.2.3 Software Downgrade

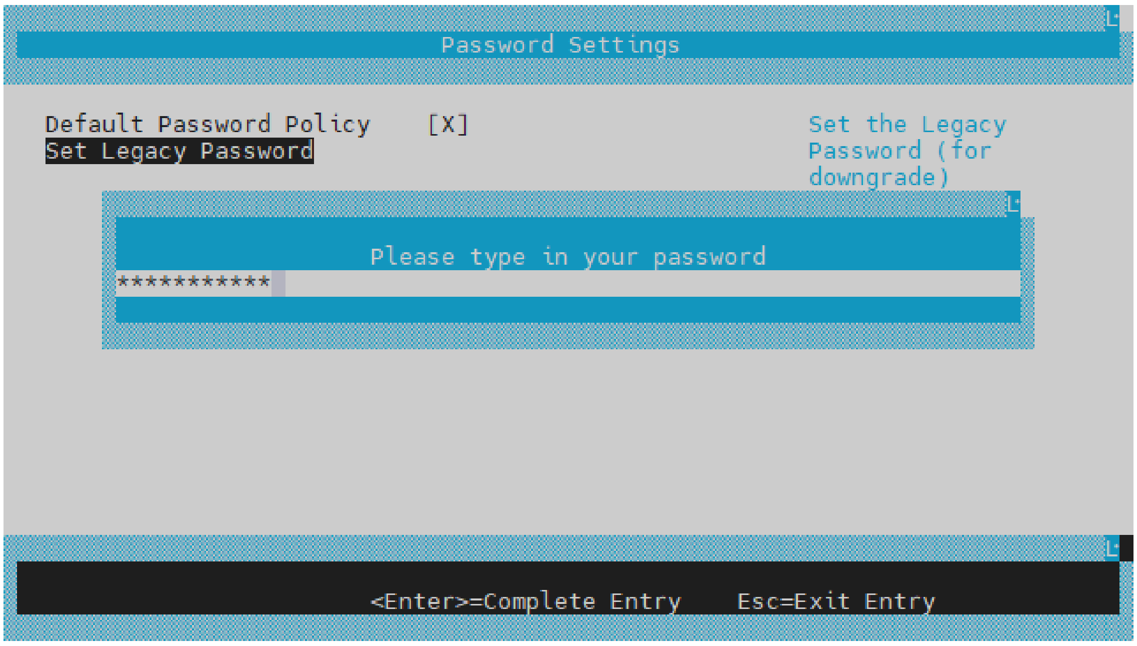
The UEFI's password policy is not backward compatible. Although downgrade is not recommended, users are allowed to downgrade their software while their password is set. But, if and only if the password is set, users must configure the legacy password prior to performing any downgrade.

For BSP 4.6.0 (DOCA 2.6.0) or higher, users must change the UEFI password saved to the older "Legacy" format.

If this procedure is not followed before performing a software downgrade, users would not be able to enter the UEFI menu.

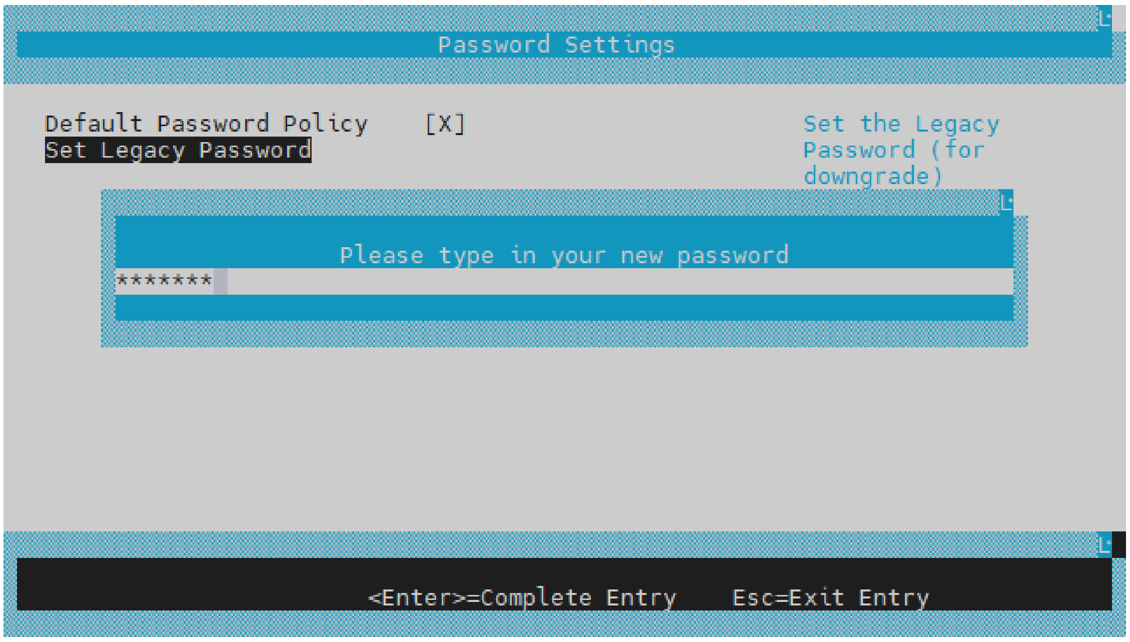
In the UEFI menu:

1. Navigate to "Device Manager" > "System Configuration" >"Password Settings" >"Set Legacy Password".
2. Select "Set Legacy Password".
3. Enter your current password:

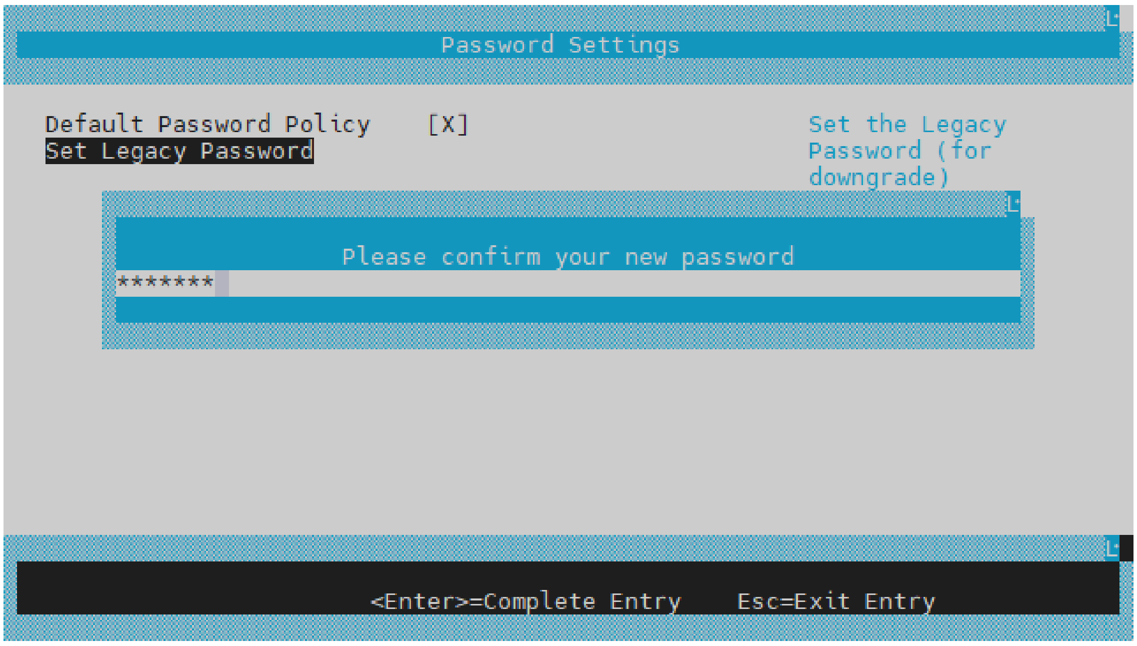

```
                         Password Settings

Default Password Policy     [X]                     Set the Legacy
Set Legacy Password                                 Password (for
                                                    downgrade)

                    Please type in your password
***********

              <Enter>=Complete Entry    Esc=Exit Entry
```

4. Type in a new legacy password between 1 and 20 characters:

> The password format allows up to 64 characters but anything greater than 20 characters is not backward compatible.


```
                         Password Settings

Default Password Policy     [X]                     Set the Legacy
Set Legacy Password                                 Password (for
                                                    downgrade)

                  Please type in your new password
*******

              <Enter>=Complete Entry    Esc=Exit Entry
```

5. Confirm the new password:



```
Password Settings

Default Password Policy     [X]               Set the Legacy
Set Legacy Password                           Password (for
                                              downgrade)

        Please confirm your new password
*******

        <Enter>=Complete Entry     Esc=Exit Entry
```

Now, you may downgrade your BlueField image.

2.5.2.4 Password Reset

To reset the UEFI menu password, users may use the ready to use capsule file `EnrollKeysCap` insta lled under `/lib/firmware/mellanox/boot/capsule/EnrollKeysCap` on the BlueField DPU file system. From the BlueField console, execute the following command, then reboot:

```
ubuntu@localhost:~$ bfrec --capsule /lib/firmware/mellanox/boot/capsule/EnrollKeysCap
```

On the next reboot, the capsule file is processed, and the UEFI password is reset to `bluefield`.

2.5.3 GRUB Password

GRUB menu entries are protected by a username and password to prevent unwanted changes to the default boot options or parameters.

The default credentials are as follows:

Username	`admin`
Password	`BlueField`

The password can be changed during BFB installation by providing a new `grub_admin_PASSWORD` parameter in `bf.cfg`:

```
# vim bf.cfg
grub_admin_PASSWORD='
grub.pbkdf2.sha512.10000.5EB1FF92FDD89BDAF3395174282C77430656A6DBEC1F9289D5F5DAD17811AD0E2196D0E49B49EF31C21972669D
180713E265BB2D1D4452B2EA9C7413C3471C53.F533423479EE7465785CC2C79B637BDF77004B5CC16C1DDE806BCEA50BF411DE04DFCCE42279
E2E1F605459F1ABA3A0928CE9271F2C84E7FE7BF575DC22935B1'
```

To get a new encrypted password value use the command `grub-mkpasswd-pbkdf2`.

After the installation, the password can be updated by editing the file `/etc/grub.d/40_custom` and then running the command `update-grub` which updates the file `/boot/grub/grub.cfg`.

3 Release Notes

The release note pages provide information for NVIDIA® BlueField® DPU family software such as changes and new features, supported platforms, and reports on software known issues as well as bug fixes.

- Changes and New Features
- Supported Platforms and Interoperability
- Bug Fixes In This Version
- Known Issues
- Validated and Supported Cables and Modules
- Release Notes Change Log History
- Bug Fixes History

3.1 Changes and New Features

For an archive of changes and features from previous releases, refer to Release Notes Change Log History.

NVIDIA® BlueField® DPUs support configuring network ports as either Ethernet only or InfiniBand only.

3.1.1 Changes and New Features in 4.7.0

- Added support for new BlueField reset and reboot procedures for loading new firmware and firmware configuration changes which replace previous need for server power cycle
- Updated the default operation mode of SuperNICs to NIC mode (from DPU mode). This is relevant to the following SKUs:
  - 900-9D3B4-00CC-EA0
  - 900-9D3B4-00SC-EA0
  - 900-9D3B4-00CV-EA0
  - 900-9D3B4-00SV-EA0
  - 900-9D3B4-00EN-EA0
  - 900-9D3B4-00PN-EA0
  - 900-9D3D4-00EN-HA0
  - 900-9D3D4-00NN-HA0

  When upgrading one of these SuperNICs to 2.7.0, if its mode of operation was changed at any point in the past, then the last configured mode of operation will remain unchanged. Otherwise, the SuperNIC will rise in NIC operation mode.
- Installing the BFB Bundle now performs NIC firmware update by default
- Added ability to install NIC firmware and BMC software in NIC mode in NVIDIA® BlueField®-3.

It is important to note the following:

- During BFB Bundle installation, Linux is expected to boot to upgrade NIC firmware and BMC software
- As Linux is booting during BFB Bundle installation, it is expected for the mlx5 core driver to timeout on the BlueField Arm
- During the BFB Bundle installation, it is expected for the mlx5 driver to error messages on the x86 host. These prints may be ignored as they are resolved by a mandatory, post-installation power cycle.
- It is mandatory to power cycle the host after the installation is complete for the changes to take effect

- Software packaging - new BlueField firmware bundle package (`bf-fwbundle-<version>.prod.bfb`), a smaller image for Day 2 upgrades, without the OS and DOCA runtime. Includes ATF, UEFI, nic-fw, bmc-fw, and eROT only.
- Improved BlueField BMC robustness -
  - Report LLDP for L2 discovery via Redfish
  - Improved BlueField DPU debuggability
- Increased support for virtio-net VF devices on BlueField-3 networking platforms to 2K
- Reduced power consumption for BlueField NIC mode
- RAS
  - Report DDR Error to OS, including both single-bit ECC error and UCE error
  - Support error injection in processors, memory, and PCIe devices

3.2 Supported Platforms and Interoperability

3.2.1 Supported NVIDIA BlueField-3 DPU Platforms

SKU	PSID	Description
900-9D3D4-00NN-HA0	MT_0000001070	NVIDIA BlueField-3 B3140H E-series HHHL DPU; 400GbE(default mode)/NDR IB; Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on board DDR; integrated BMC; Crypto Disabled
900-9D3B4-00CV-EA0	MT_0000001093	NVIDIA BlueField-3 B3220L E-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SC-EA0	MT_0000001117	NVIDIA BlueField-3 B3210E E-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B6-00SN-AB0	MT_0000000964	NVIDIA BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE / NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B4-00CC-EA0	MT_0000000966	NVIDIA BlueField-3 B3210L E-series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual port QSFP112; PCIe Gen4.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00PN-EA0	MT_0000001011	NVIDIA BlueField-3 B3140L E-Series FHHL DPU; 400GbE / NDR IB (default mode); Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled

SKU	PSID	Description
900-9D3B6-00CC-AA0	MT_0000001024	NVIDIA BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SC-AA0	MT_0000001025	NVIDIA BlueField-3 B3210 P-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3D4-00EN-HA0 / 900-9D3D4-00EN-HAQ	MT_0000001069	Nvidia BlueField-3 B3140H E-series HHHL DPU; 400GbE(default mode)/NDR IB; Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00SC-EA0	MT_0000000967	NVIDIA BlueField-3 B3210L E-series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual port QSFP112; PCIe Gen4.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3D4-03EN-HA0	MT_0000001125	HPE Data Processing Unit InfiniBand NDR/Ethernet 400Gb 1-port QSFP112 HHHL B3140H Adapter
699-21014-0230	NVD0000000038	NVIDIA A800T WITH BLUEFIELD-3; P1014 SKU 230; GENERIC; GA100 80GB HBM2E; PASSIVE DUAL SLOT 350W GEN5; DPU CRYPTO ON
900-9D3B4-00EN-EA0	MT_0000001010	NVIDIA BlueField-3 B3140L E-Series FHHL DPU; 400GbE / NDR IB (default mode); Single-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00SV-EA0	MT_0000001094	NVIDIA BlueField-3 B3220L E-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16; 8 Arm cores; 16GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3C6-00SV-GA0	MT_0000001101	NVIDIA BlueField-3 B3220SH E-Series No Heatsink FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3C6-00SV-DA0	MT_0000001102	NVIDIA BlueField-3 B3220SH E-Series FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Disabled;
900-9D3B6-00CV-AA0	MT_0000000884	NVIDIA BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B6-00SV-AA0	MT_0000000965	NVIDIA BlueField-3 B3220 P-Series FHHL DPU; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Disabled
900-9D3B6-H1CN-AB0	MT_0000000883	NVIDIA BlueField-3 B3240 P-Series Dual-slot FHHL DPU; 400GbE / NDR IB (default mode); Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3C6-00CV-DA0	MT_0000001075	NVIDIA BlueField-3 B3220SH E-Series FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Enabled; Secure Boot
900-9D3C6-00CV-GA0	MT_0000001083	NVIDIA BlueField-3 B3220SH E-Series No heatsink FHHL Storage Controller; 200GbE (default mode) / NDR200 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 48GB on-board DDR; integrated BMC; Crypto Enabled

SKU	PSID	Description
900-9D3B6-00CC-EA0	MT_0000001115	NVIDIA BlueField-3 B3210E E-Series FHHL DPU; 100GbE (default mode) / HDR100 IB; Dual-port QSFP112; PCIe Gen5.0 x16 with x16 PCIe extension option; 16 Arm cores; 32GB on-board DDR; integrated BMC; Crypto Enabled
900-9D3B4-00EN-EAS	MT_0000001029	NVIDIA BlueField-3 BF3140L E-series SuperNIC NDR/400GbE single port QSFP112; PCIe Gen5.0 x16 FHHL; Crypto Enabled; 16GB on board DDR; integrated BMC; Tall Bracket; IPN QP

3.2.2 Supported NVIDIA BlueField-2 DPU Platforms

NVIDIA SKU	Legacy OPNs	PSID	Description
900-9D219-0056-SN1	MBF2M516A-CENOT	MT_0000000560	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0073-ST0	MBF2H532C-AESOT	MT_0000000766	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0086-ST1	MBF2M516A-CECOT	MT_0000000375	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-ST2	MBF2H516C-EESOT	MT_0000000737	BlueField-2 P-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-STA	MBF2H516C-CEUOT	MT_0000000973	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
900-9D208-0086-ST4	MBF2M516C-EECOT	MT_0000000728	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0086-ST2	MBF2H536C-CECOT	MT_0000000768	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D206-0063-ST1	MBF2H322A-AEEOT	MT_0000000543	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 8GB on-board DDR; 1GbE OOB management; HHHL
900-9D250-0048-ST1	MBF2M345A-HECOT	MT_0000000716	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D218-0083-ST2	MBF2H512C-AECOT	MT_0000000724	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0086-SQ0	MBF2H516C-CECOT	MT_0000000729	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0076-ST5	MBF2M516C-CESOT	MT_0000000731	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL

NVIDIA SKU	Legacy OPNs	PSID	Description
900-9D208-0076-ST3	MBF2H536C-CESOT	MT_0000000767	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
699140280000	N/A	NVD0000000020	ZAM/NAS
900-9D219-0066-ST2	MBF2M516A-CEEOT	MT_0000000561	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0086-ST0	MBF2M516A-EECOT	MT_0000000376	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D206-0053-SQ0	MBF2H332A-AENOT	MT_0000000539	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D219-0006-ST0	MBF2H516A-CEEOT	MT_0000000702	BlueField-2 DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0056-ST2	MBF2H516A-CENOT	MT_0000000703	BlueField-2 DPU 100GbE Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D219-0066-ST3	MBF2H516A-EEEOT	MT_0000000704	BlueField-2 DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D218-0073-ST1	MBF2H512C-AESOT	MT_0000000723	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-ST6	MBF2M516C-EESOT	MT_0000000732	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D208-0086-ST3	MBF2M516C-CECOT	MT_0000000733	BlueField-2 E-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D218-0083-ST4	MBF2H532C-AECOT	MT_0000000765	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled; Crypto Enabled; 32GB on-board DDR; 1GbE OOB management; FHHL
P1004 / 699210040230	N/A	NVD0000000015	ROY BlueField-2 + GA100 PCIe Gen4 x8; two 100Gbe/EDR QSFP28 ports; FHFL
900-9D219-0056-ST1	MBF2M516A-EENOT	MT_0000000377	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D206-0063-ST2	MBF2H332A-AEEOT	MT_0000000540	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D206-0053-ST2	MBF2H322A-AENOT	MT_0000000544	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Disabled; 8GB on-board DDR; 1GbE OOB management; HHHL

NVIDIA SKU	Legacy OPNs	PSID	Description
900-9D219-0066-ST0	MBF2M516A-EEEOT	MT_0000000559	BlueField-2 E-Series DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Enabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D208-0076-ST1	MBF2H516C-CESOT	MT_0000000738	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; Tall Bracket; FHHL
900-9D206-0083-ST3	MBF2H332A-AECOT	MT_0000000541	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto and Secure Boot Enabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D219-0056-SQ0	MBF2H516A-EENOT	MT_0000000705	BlueField-2 DPU 100GbE/EDR/HDR100 VPI Dual-Port QSFP56; PCIe Gen4 x16; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; FHHL
900-9D250-0038-ST1	MBF2M345A-HESOT	MT_0000000715	BlueField-2 E-Series DPU; 200GbE/HDR single-port QSFP56; PCIe Gen4 x16; Secure Boot Enabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management; HHHL
900-9D218-0073-ST4	MBF2H512C-AEUOT	MT_0000000972	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; integrated BMC; PCIe Gen4 x8; Secure Boot Enabled with UEFI disabled; Crypto Disabled; 16GB on-board DDR; 1GbE OOB management
900-9D208-0076-STB	MBF2H536C-CEUOT	MT_0000001008	BlueField-2 P-Series DPU 100GbE Dual-Port QSFP56; integrated BMC; PCIe Gen4 x16; Secure Boot Enabled with UEFI Disabled; Crypto Disabled; 32GB on-board DDR; 1GbE OOB management; FHHL
900-9D206-0083-ST1	MBF2H322A-AECOT	MT_0000000542	BlueField-2 P-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto and Secure Boot Enabled; 8GB on-board DDR; 1GbE OOB management; HHHL
900-9D206-0063-ST4	MBF2M322A-AEEOT	MT_0000000490	BlueField-2 E-Series DPU 25GbE Dual-Port SFP56; PCIe Gen4 x8; Crypto Enabled; 8GB on-board DDR; 1GbE OOB management; HHHL

3.2.3 Embedded Software

The BlueField DPU installation DOCA local repo package for DPU for this release is `DOCA_2.7.0_BSP_4.7.0_Ubuntu_22.04-2.23-07.prod.bfb`.

The following software components are embedded in it:

Unable to render include or excerpt-include. Could not retrieve page.

For more information about embedded software components and drivers, refer to the DOCA Release Notes.

3.2.4 Supported DPU Linux Distributions (aarch64)

- Ubuntu 22.04

3.2.4.1 Supported Host OS per DOCA-Host Installation Profile

Unable to render include or excerpt-include. Could not retrieve page.

3.2.5 Supported Open vSwitch

- 2.15.1

3.3 Bug Fixes In This Version

For an archive of bug fixes from previous releases, please see "Bug Fixes History".

Ref #	Issue Description
3814526	Description: Kubernetes official repository changed location and it causes apt/yum failures on the BlueField OSes older than BSP 4.7.0 (DOCA 2.7.0).
	Keywords: Kubernetes; OS
	Reported in version: 4.6.0
3814526	Description: The location of the official Kubernetes repository changed, causing apt/yum failures.
	Keywords: Kubernetes
	Reported in version: 4.6.0
3820661	Description: Virtio-net may see TX timeout on specific queues.
	Keywords: Emulated devices
	Reported in version: 4.6.0
3850459	Description: BMC components update fails while using default BMC root password.
	Keywords: BMC; update
	Reported in version: 4.6.0
3774088	Description: When enrolling a certificate to the UEFI DB, the failure message "`ERROR: Unsupported file type!`" is displayed when the DB was full.
	Keywords: SNAP; UEFI; error
	Reported in version: 4.5.0
3787003	Description: Host PCIe driver hangs when hot plugging a device due to SF creation and error flow handling failure.
	Keywords: Subfunction; hot-plug
	Reported in version: 4.5.0
3663398	Description: On rare occasions, OP-TEE may panic upon boot.
	Keywords: fTPM over OP-TEE
	Reported in version: 4.5.0

Ref #	Issue Description
3677366	Description: On rare occasions, the devices `/dev/tpm0` and `/dev/tpmrm0` are not created triggering an fTPM panic during boot. This message indicates that the fTPM over OP-TEE feature is not functional.
	Keywords: fTPM over OP-TEE
	Reported in version: 4.5.0
3712916	Description: The following fTMP over OP-TEE error appears when booting BlueField: `ftpm-tee PRP0001:01: ftpm_tee_probe: tee_client_open_session failed, err=ffff3024`
	Keywords: fTPM over OP-TEE
	Reported in version: 4.5.0
3830034	Description: The following bfscripts have been deprecated and may no longer work as expected: bfinst, bfdracut, bfacpievt. These scripts are no longer supported and will eventually be removed. Warning logs have been added to notify users.
	Keywords: Deprecated bfscripts
	Reported in version: 4.5.0
3618936	Description: When moving to DPU mode from NIC mode, it is necessary to reinstall the BFB and perform a graceful reboot to the DPU by shutting down the Arm cores before rebooting the host system.
	Keywords: NIC mode
	Reported in version: 4.2.2
3603146	Description: Running `mlxfwreset` on BlueField-3 may cause the external host to crash when the RShim driver is running on that host.
	Keywords: RShim; mlxfwreset
	Reported in version: 4.2.1
3444073	Description: `mlxfwreset` is not supported in this release.
	Keywords: mlxfwreset; support
	Reported in version: 4.0.2

3.4 Known Issues

Ref #	Issue
3239320	Description: Resetting hugepage size to 0 on Rocky Linux 8.6 using the `sysctl` tool fails.
	Workaround: Use the following command instead: `echo 0 > /sys/kernel/mm/hugepages/hugepages-<Size>/nr_hugepages`
	Keyword: Hugepage; sysctl

Ref #	Issue
	Reported in version: 4.7.0
3859113	Description: Reloading MLNX_OFED drivers with the command `/etc/init.d/openibd restart` fails when the NVMe driver is installed and in use.
	Workaround: Reboot the machine to load all the MLNX_OFED drivers.
	Keyword: NVMe; driver
	Reported in version: 4.7.0
3748649	Description: With the numbering of CPUs in an 8-core configuration, the kernel is expected to assign virtual CPU ID numbers from 0-7, where N is the number of cores enabled. With CTyunOS, however, the numbering is unexpected.
	Workaround: N/A
	Keyword: CTyunOS; CPU numbering
	Reported in version: 4.7.0
3756748	Description: When performing BFB push repeatedly, BlueField-3 may in rare instances fail to boot with the message "`PSC error -60`" appearing in the RShim log sometimes.
	Workaround: Reset the card or repeat the operation (bfb push).
	Keyword: BFB Push; FW Reset
	Reported in version: 4.7.0
3831230	Description: In OpenEuler 20.03, the Linux Kernel version 4.19.90 is affected by an issue that impacts the discard/trim functionality for the BlueField's eMMC which may cause degraded performance of the eMMC over time.
	Workaround: Upgrade to Linux Kernel version 5.10 or later.
	Keyword: eMMC discard; trim functionality
	Reported in version: 4.7.0
3665070	Description: Virtio-net controller fails to load if `DPA_AUTHENTICATION` is enabled.
	Workaround: N/A
	Keyword: Virtio-net; DPA
	Reported in version: 4.7.0
3862683	Description: Creating VFs and hotplug PFs in parallel can lead to controller crash.
	Workaround: Create VFs followed by hotplug PF or vice versa.
	Keyword: Virtio-net emulation
	Reported in version: 4.7.0
3844066	Description: On CentOS 7.6 with kernel 4.19, bringing up OVS bridge interface causes call traces: `WARNING: CPU: 5 PID: 14339 at kernel/rcu/tree_plugin.h:342` `rcu_note_context_switch+0x48/0x538`
	Workaround: Do not bring UP OVS bridge interfaces.
	Keyword: CentOS; kernel; rcu_note_context_switch
	Reported in version: 4.7.0

Ref #	Issue
3886315	Description: To reset or shut down the BlueField Arm, it is mandatory to specify the `--sync 0` argument. For example: `mlxfwreset -d <device> -l 1 -t 4 --sync 0  r`
	Workaround: N/A
	Keyword: Arm; shutdown
	Reported in version: 4.7.0
3881941	Description: When working with RShim 2.0.28, PCIe host crash may rarely occur at the beginning of BFB push after the Arm reset.
	Workaround: Downgrade to RShim 2.0.27 or upgrade to RShim 2.0.29.
	Keyword: RShim; driver
	Reported in version: 4.7.0
3844705	Description: In OpenEuler 20.03, the Linux Kernel version 4.19.90 is affected by an issue that impacts the discard/trim functionality for the DPU eMMC device which may cause degraded performance of the DPU eMMC over time.
	Workaround: Upgrade to Linux Kernel version 5.10 or later.
	Keyword: eMMC discard; trim functionality
	Reported in version: 4.7.0

3877725	Description: During BFB installation in NIC mode on BlueField-3, too much information is added into RShim log which fills it, causing the Linux installation progress log to not appear in the RShim log.
	`echo "DISPLAY_LEVEL 2" > /dev/rshim0/misc`
	`cat /dev/rshim0/misc`

	Workaround: Monitor the BlueField-3 Arm's UART console to check whether BFB installation has completed or not for NIC mode.
	`[13:58:39] INFO: Installation finished`
	`...`
	`[14:01:53] INFO: Rebooting...`

	Keyword: NIC mode; BFB install
	Reported in version: 4.7.0
3875394	Description: After the BFB installation, the root partition UUID in `/etc/fstab` does not match current partition UUID. As a result root partition is mounted as read-only: `/dev/nvme0n1p2 on / type xfs (ro,relatime,attr2,inode64,logbufs=8,logbsize=32k,noquota)`

Ref #	Issue
	Workaround: 1. Fix root partition UUID in the `/etc/fstab`: `mount -o remount,rw /dev/nvme0n1p2 /` 2. Get UUID: `# lsblk -o UUID /dev/nvme0n1p2` `UUID` `ae1d5e37-7aee-4234-984b-9a9203bfd182` 3. Update UUID in the `/etc/fstab` to match the one printed by `lsblk`: `UUID=ae1d5e37-7aee-4234-984b-9a9203bfd182  /           xfs     defaults                    0 1`
	Keyword: Read-only; OL; UUID
	Reported in version: 4.7.0
3855702	Description: Trying to jump from a steering level in the hardware to a lower level using software steering is not supported on `rdma-core` lower than 48.x.
	Workaround: N/A
	Keyword: RDMA; SWS
	Reported in version: 4.7.0
3844705	Description: In OpenEuler 20.03, the Linux Kernel version 4.19.90 is affected by a bug that impacts the discard/trim functionality for the DPU eMMC device which may cause degraded performance of the DPU eMMC over time.
	Workaround: Upgrade to Linux Kernel version 5.10 or later.
	Keyword: eMMC discard; trim functionality
	Reported in version: 4.7.0
3743879	Description: `mlxfwreset` could timeout on servers where the RShim driver is running and INTx is not supported. The following error message is printed: `BF reset flow encountered a failure due to a reset state error of negotiation timeout`.
	Workaround: Set `PCIE_HAS_VFIO=0` and `PCIE_HAS_UIO=0` in `/etc/rshim.conf` and restart the RShim driver. Then re-run the `mlxfwreset` command. If host Linux kernel lockdown is enabled, then manually unbind the RShim driver before `mlxfwreset` and bind it back after `mlxfwreset`: `echo "DROP_MODE 1" > /dev/rshim0/misc` `mlxfwreset <arguments>` `echo "DROP_MODE 0" > /dev/rshim0/misc`
	Keyword: Timeout; mlxfwreset; INTx
	Reported in version: 4.7.0
3670361	Description: Rarely, the driver takes more than several minutes to load.
	Workaround: Re-run `/sbin/mlnx_bf_configure`.
	Keywords: Driver; boot

Ref #	Issue
	Reported in version: 4.6.0
3746866	Description: The error message `IANA PEN registry open failed: No such file or directory` may appear when using ipmitool version 1.8.19-7. This message can be safely ignored.
	Workaround: N/A
	Keywords: IPMI; Debian
	Reported in version: 4.6.0

3755143	Description: UEFI synchronous exception is observed at address 0x479B7xxxx where the UEFI module names are not printed. See the following example:
	`ERR[UEFI]: PC=0x479B78480(B4000040 3900001F A94153F3 F94013F5)`
	`ERR[UEFI]: PC=0x479B78480`
	`ERR[UEFI]: PC=0x479B7E684`
	`ERR[UEFI]: PC=0x47A0E93F4`
	`ERR[UEFI]: PC=0x47A0E9608`

	Workaround: Run software reset or reinstall the BFB.
	Keywords: UEFI synchronous exception
	Reported in version: 4.6.0
3772177	Description: SHHing to the DPU with Debian 12 can print the following warning: `-bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)`.
	Workaround: Run: `sudo dpkg-reconfigure locales`
	Keywords: Debian 12; locale; LC_ALL
	Reported in version: 4.6.0
3704985	Description: When the RShim driver is not running on the external host or when the `tmfifo_net0` interface is down on the DPU side, the following kernel warning may appear on the DPU side: `virtio_net virtio1 tmfifo_net0: TX timeout`.
	Workaround: N/A
	Keywords: RShim; log
	Reported in version: 4.6.0
3767580	Description: On Debian 12, the first boot after BFB installation may fail with the following kernel panic: `[ end Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000100 ]`
	Workaround: Reset the DPU using the RShim interface: `echo "SW_RESET 1" > /dev/rshim0/misc`
	Keywords: Debian 12; Kernel panic; kill init
	Reported in version: 4.6.0

Ref #	Issue
3771601	Description: On Debian 12, `/etc/init.d/openibd restart` fails with the following error: `rmmod: ERROR: Module rdma_cm is in use by: nvme_rdma`
	Workaround: Run: `modprobe -r nvme_rdma; /etc/init.d/openibd restart`
	Keywords: Debian 12; openibd; nvme_rdma
	Reported in version: 4.6.0
3686053	Description: BlueField-2 supports a total of 120GB of PCIe memory space. When the GPU is configured to be exposed to the BlueField, it requests 32GB of space for its BAR0. The Linux 5.15 kernel also attempts to reserve space for the total number of VFs, even if they are not enabled. By default, the A100 allows 20 VFs which each need 4GB of memory space. Because of PCIe memory alignment requirements and other small devices on the bus, this additional 80GB causes PCIe resource allocation to fail.
	Workaround: Add "pci=realloc=off" to the Linux command line. This will force Linux to accept the resource allocation done by UEFI and allow enumeration to succeed.
	Keyword: VF; kernel; resources
	Reported in version: 4.6.0
3678069	Description: If using DPUs with NVMe and mmcbld and configured to boot from mmcblk, users must create a `bf.cfg` file with `device=/dev/mmcblk0` before installing the `*.bfb`.
	Workaround: N/A
	Keyword: NVMe
	Reported in version: 4.6.0
3747285	Description: The ipmitool command to force PXE in BMC modifies both the IPMI and Redfish request settings. When Redfish is enabled in UEFI, Redfish takes priority, so all PXE boot entries are attempted and before regular boot continues.
	Workaround: Redfish must be disabled if IPMI force PXE retry behavior is expected.
	Keyword: PXE; retry; fail
	Reported in version: 4.6.0
3745529	Description: When rebooting the DPU while the host side is running traffic over bond, TX timeout is likely to occur. This generates a TX timeout recovery flow that may conflict with host recovery attempts from the DPU reboot.
	Workaround: N/A
	Keyword: Bond; timeout
	Reported in version: 4.6.0
3733713	Description: CA certificates in the UEFI are stored in the database provided by the user. It is user responsibility to enroll the correct certificate. The user is the owner of the certificate and should make sure of its validity.
	Workaround: N/A
	Keyword: CA certificates; UEFI

Ref #	Issue
	Reported in version: 4.6.0
3733740	Description: CA certificates in the BMC are owned by the user who is required to enroll valid and correct certificates. If incorrect BMC CA certificates are enrolled, then DPU-BMC redfish communication will be invalid.
	Workaround: N/A
	Keyword: CA certificates; BMC
	Reported in version: 4.6.0
3666574	Description: Running `systemctl restart openibd` on the DPU can result in openvswitch service crash.
	Workaround: Run `/etc/init.d/openvswitch-switch start`.
	Keyword: OVS fail; openibd
	Reported in version: 4.6.0
3204153	Description: On BlueField-2, the OOB may not get an IP address due to the interface being down.
	Workaround: restart auto-negotiation using the command `ethtool -r oob_net0`.
	Keyword: OOB; IP
	Reported in version: 4.5.0
3601491	Description: Symmetric pause must be enabled in the DHCP server for the OOB to be able to reliably get an IP address assigned.
	Workaround: N/A
	Keyword: OOB; IP
	Reported in version: 4.5.0
3673330	Description: On Debian 12, Arm ports remain in Legacy mode after multiple Arm reboot iterations. The following error message appears in `/var/log/syslog`: `mlnx_bf_configure[2601]: ERR: Failed to configure switchdev mode for 0000:03:00.0 after 61 retries`

	Workaround: Run:
	`$ echo SET_MODE_RETRY_NUM=300 >> /etc/mellanox/mlnx-bf.conf`
	`$ reboot`

	Keyword: Debian; Arm
	Reported in version: 4.5.0
3695543	Description: PXE boot may fail after a firmware upgrade from 32.36.xxxx, 32.37.xxxx, to 32.38.xxxx and above.

Ref #	Issue
	Workaround: Create `/etc/bf.cfg` with the following lines, then run `bfcfg` to recreate the PXE boot entries:

```
BOOT0=DISK
BOOT1=NET-NIC_P0-IPV4
BOOT2=NET-NIC_P0-IPV6
BOOT3=NET-NIC_P1-IPV4
BOOT4=NET-NIC_P1-IPV6
BOOT5=NET-OOB-IPV4
BOOT6=NET-OOB-IPV6
```

Ref #	Issue
	Keyword: MAC allocation; PXE boot
	Reported in version: 4.5.0
3647476	Description: Debian 12 OS does not support CT tunnel offload.
	Workaround: Recompile the kernel with `CONFIG_NET_TC_SKB_EXT` set.
	Keyword: Connection tracking; Linux
	Reported in version: 4.5.0
3007696	Description: When configuring a static IP address for `tmfifo_net0` interface in `/etc/network/interfaces`, the IP address is lost after restarting the RShim driver on Debian Linux.
	Workaround: Use netplan configuration. For example

```
# cat /etc/netplan/tmfifo_net0.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    tmfifo_net0:
      addresses:
      - 192.168.100.1/30
      dhcp4: false
```

Ref #	Issue
	Then run "netplan apply".
	Keyword: IP address; tmfifo_net0; host
	Reported in version: 4.5.0
3670628	Description: When NIC subsystem is in recovery mode, the interface towards to NVMe is not accessible. Thus, the SSD boot device would not be available.
	Workaround: The admin must configure the Arm subsystem boot device to boot from the eMMC, for example.
	Keyword: mlxfwreset; RShim
	Reported in version: 4.5.0
3702393	Description: On rare occasions, the boot process part of SWRESET (via RShim) or FWRESET (via mlxfwreset) may result in a device hanging on the boot flow or cause the host server to reboot.
	Workaround: Perform graceful shutdown and then a power cycle.
	Keyword: mlxfwreset; RShim
	Reported in version: 4.5.0
3665724	Description: If the UEFI password is an empty string (""), then it cannot be changed via Redfish.
	Workaround: UEFI; password; Redfish
	Keyword: UEFI; password; Redfish

Ref #	Issue
	Reported in version: 4.5.0
3671185	Description: XFRM rules must be deleted before driver restart or warm reboot are performed.
	Workaround: N/A
	Keyword: IPsec
	Reported in version: 4.5.0
3666160	Description: Installing BFB using `bfb-install` when `mlxconfig` `PF_TOTAL_SF` >1700, triggers server reboot immediately.
	Workaround: Change `PF_TOTAL_SF` to 0, perform graceful shutdown, then power cycle, and then install the BFB.
	Keyword: SF; PF_TOTAL_SF; BFB installation
	Reported in version: 4.2.2
3605254	Description: Following a system power cycle, both the DPU and BMC boot independently which may lead to the DPU's UEFI boot process to complete before the BMC's. As a result, when attempting to establish Redfish communication, the BMC may not yet be prepared to respond.
	Workaround: Wait until the BMC is done booting before issuing a reset command to the DPU.
	Keyword: Power cycle; Redfish; boot
	Reported in version: 4.2.1
3602044	Description: When the public key is deleted while Redfish is enabled, UEFI secure boot is disabled and UEFI reverts to Setup Mode (i.e., the `SecureBootEnable` Redfish property is reset to `false`). If later, the public key is re-enrolled, the platform does not implement UEFI secure boot until the `SecureBootEnable` Redfish property is explicitly changed to `true`.
	Workaround: Set `SecureBootEnable` to true using the Redfish API.
	Keyword: Redfish; UEFI secure boot
	Reported in version: 4.2.1
3592080	Description: When using UEK8 on the host in DPU mode, creating a VF on the host consumes about 100MB memory on the DPU.
	Workaround: N/A
	Keyword: UEK; VF
	Reported in version: 4.2.1
3568341	Description: Downgrading BSP software from 4.2.0 fails if UEFI secure boot is enabled.
	Workaround: Disable UEFI secure boot before downgrading.
	Keyword: Software; downgrade
	Reported in version: 4.2.0
3566042	Description: Virtio hotplug is not supported in GPU-HOST mode on the NVIDIA Converged Accelerator.
	Workaround: N/A
	Keyword: Virtio; Converged Accelerator
	Reported in version: 4.2.0

Ref #	Issue
3546474	Description: PXE boot over ConnectX interface might not work due to an invalid MAC address in the UEFI boot entry.
	Workaround: On the DPU, create `/etc/bf.cfg` file with the relevant PXE boot entries, then run the command `bfcfg`.
	Keyword: PXE; boot; MAC
	Reported in version: 4.2.0
3546202	Description: After rebooting a BlueField-3 DPU running Rocky Linux 8.6 BFB, the kernel log shows the following error: `[    3.787135] mlxbf_gige MLNXBF17:00: Error getting PHY irq. Use polling instead` This message indicates that the Ethernet driver will function normally in all aspects, except that PHY polling is enabled.
	Workaround: N/A
	Keyword: Linux; PHY; kernel
	Reported in version: 4.2.0
3306489	Description: When performing longevity tests (e.g., mlxfwreset, DPU reboot, burning of new BFBs), a host running an Intel CPU may observer errors related to "CPU 0: Machine Check Exception".
	Workaround: Add `intel_idle.max_cstate=1` entry to the kernel command line.
	Keyword: Longevity; mlxfwreset; DPU reboot
	Reported in version: 4.2.0
3538486	Description: When removing LAG configuration from the DPU, a kernel warning for `uverbs_destroy_ufile_hw` is observed if virtio-net-controller is still running.
	Workaround: Stop virtio-net-controller service before cleaning up bond configuration.
	Keyword: Virtio-net; LAG
	Reported in version: 4.2.0
3462630	Description: When trying to perform a PXE installation when UEFI Secure Boot is enabled, the following error messages may be observed: `error: shim_lock protocol not found.` `error: you need to load the kernel first.`
	Workaround: Download a Grub EFI binary from the Ubuntu website. For further information on Ubuntu UEFI Secure Boot PXE Boot, please visit Ubuntu's official website.
	Keyword: PXE; UEFI Secure Boot
	Reported in version: 4.0.2
3412847	Description: Socket-Direct is currently not supported on BlueField-3 devices.
	Workaround: N/A
	Keyword: Socket-Direct; support
	Reported in version: 4.0.2
3448841	Description: While running CentOS 8.2, switchdev Ethernet DPU runs in "shared" RDMA net namespace mode instead of "exclusive".

Ref #	Issue
	Workaround: Use `ib_core` module parameter `netns_mode=0`. For example: `echo "options ib_core netns_mode=0" >> /etc/modprobe.d/mlnx-bf.conf`
	Keywords: RDMA; isolation; Net NS
	Reported in version: 4.0.2
3413938	Description: Using `mlnx-sf` script, creating and deleting an SF with same ID number in a stressful manner may cause the setup to hang due to a race between create and delete commands.
	Workaround: N/A
	Keywords: Hang; `mlnx-sf`
	Reported in version: 4.0.2
3452740	Description: Ovs-pki is not working due to two versions of OpenSSL being installed, causing the PKA engine to not load properly.
	Workaround: N/A
	Keywords: PKA; OpenSSL
	Reported in version: 4.0.2
3273435	Description: Changing the mode of operation between NIC and DPU modes results in different capabilities for the host driver which might cause unexpected behavior.
	Workaround: Reload the host driver or reboot the host.
	Keywords: Modes of operation; driver
	Reported in version: 4.0.2
2706803	Description: When an NVMe controller, SoC management controller, and DMA controller are configured, the maximum number of VFs is limited to 124.
	Workaround: N/A
	Keywords: VF; limitation
	Reported in version: 4.0.2
3264224	Description: When trying to change boot order using efibootmgr, BlueField fails to attempt PXE boot from `p0` even though efibootmgr returns a successful result.
	Workaround: Drop into the UEFI menu and regenerate all the EFI entries.
	Keywords: PXE; efibootmgr
	Reported in version: 3.9.3.1
3188415	Description: An Arm firmware update to the same version that is installed will fail and is not supported.
	Workaround: N/A
	Keywords: Arm; firmware; update
	Reported in version: 3.9.2
N/A	Description: The `BootOptionEnabled` attribute changes back to true after DPU-force reset.
	Workaround: N/A

Ref #	Issue
	Keywords: Redfish; `BootOptionEnabled`
	Reported in version: 3.9.2
3012182	Description: The command `ethtool -I --show-fec` is not supported by the DPU with kernel 5.4.
	Workaround: N/A
	Keywords: Kernel; show-fec
	Reported in version: 3.9.0
2855986	Description: After disabling SR-IOV VF on a virtio device, removing virtio-net/PCIe driver from guest OS may render the virtio controller unusable.
	Workaround: Restart the virtio-net controller to recover it. To avoid this issue, monitor the log from controller and make sure VF resources are destroyed before unloading virtio-net/PCIe drivers.
	Keywords: Virtio-net; VF
	Reported in version: 3.9.0
2863456	Description: SA limit by packet count (hard and soft) are supported only on traffic originated from the ECPF. Trying to configure them on VF traffic removes the SA when hard limit is hit. However, traffic could still pass as plain text due to the tunnel offload used in such configuration.
	Workaround: N/A
	Keywords: ASAP2; IPsec Full Offload
	Reported in version: 3.9.0
2982184	Description: When multiple BlueField resets are issued within 10 seconds of each other, EEPROM error messages are displayed on the console and, as a result, the BlueField may not boot from the eMMC and may halt at the UEFI menu.
	Workaround: Power-cycle the BlueField to fix the EEPROM issue. Manual recovery of the boot options and/or SW installation may be needed.
	Keywords: Reset; EEPROM
	Reported in version: 3.9.0
2853408	Description: Some pre-OS environments may fail when sensing a hot plug operation during their boot stage.
	Workaround: Run "`mlxconfig -d <mst dev> set PF_LOG_BAR_SIZE=0`".
	Keywords: BIOS; hot-plug; Virtio-net
	Reported in version: 3.9.0
2934833	Description: Running I/O traffic and toggling both physical ports status in a stressful manner on the receiving-end machine may cause traffic loss.
	Workaround: N/A
	Keywords: MLNX_OFED; RDMA; port toggle
	Reported in version: 3.8.5
2911425	Description: ProLiant DL385 Gen10 Plus server with BIOS version 1.3 hangs when large number of SFs (`PF_TOTAL_SF=252`) are configured.
	Workaround: Update the BIOS version to 2.4 which should correctly detect the PCIe device with the bigger BAR size.

Ref #	Issue
	Keywords: Scalable functions; BIOS
	Reported in version: 3.8.5
N/A	Description: Only QP queues are supported for GGA accelerators from this version onward.
	Workaround: N/A
	Keywords: Firmware; SQ; QP
	Reported in version: 3.8.0
2846108	Description: Setting `VHCA_TRUST_LEVEL` does not work when there are active SFs or VFs.
	Workaround: N/A
	Keywords: Firmware; SF; VF
	Reported in version: 3.8.0
2750499	Description: Some devlink commands are only supported by mlnx devlink (`/opt/mellanox/iproute2/sbin/devlink`). The default devlink from the OS may produce failure (e.g., `devlink port show -j`).
	Workaround: N/A
	Keywords: Devlink
	Reported in version: 3.7.1
2730157	Description: Kernel upgrade is not currently supported on BlueField as there are out of tree kernel modules (e.g., ConnectX drivers that will stop working after kernel upgrade).
	Workaround: Kernel can be upgraded if there is a matching DOCA repository that includes all the drivers compiled with the new kernel or as a part of the new BFB package.
	Keywords: Kernel; upgrade
	Reported in version: 3.7.0
2706710	Description: Call traces are seen on the host when recreating VFs before the controller side finishes the deletion procedure.
	Workaround: N/A
	Keywords: Virtio-net controller
	Reported in version: 3.7.0
2685478	Description: 3rd party (netkvm.sys) Virtio-net drivers for Windows do not support SR-IOV.
	Workaround: N/A
	Keywords: Virtio-net; SR-IOV; WinOF-2
	Reported in version: 3.7.0
2684501	Description: Once the contiguous memory pool, a limited resource, is exhausted, fallback allocation to other methods occurs. This process triggers `cma_alloc` failures in the dmesg log.
	Workaround: N/A
	Keywords: Log; cma_alloc; memory
	Reported in version: 3.7.0
2590016	Description: ibdev2netdev tool is not supported for PCIe PF operating in switchdev mode or on SFs.

Ref #	Issue
	Workaround: N/A
	Keywords: ibdev2netdev
	Reported in version: 3.6.0.11699
2590016	Description: A "double free" error is seen when using the "curl" utility. This error is from libcrypto.so library which is part of the OpenSSL package. This happens only when OpenSSL is configured to use a dynamic engine (e.g. Bluefield PKA engine).
	Workaround: Set `OPENSSL_CONF=/etc/ssl/openssl.cnf.orig` before using the curl utility. For example: `# OPENSSL_CONF=/etc/ssl/openssl.cnf.orig curl -O https://tpo.pe/pathogen.vim` OPENSSL_CONF is aimed at using a custom config file for applications. In this case, it is used to point to a config file where dynamic engine (PKA engine) is not enabled.
	Keywords: OpenSSL; curl
	Reported in version: 3.6.0.11699
2407897	Description: The host may crash when the number of PCIe devices overflows the PCIe device address. According to the PCIe spec, the device address space is 8 bits in total—device (5 bits) and function (3 bits)—which means that the total number of devices cannot be more than 256. The second PF maximum number of VFs is limited by the total number of additional PCIe devices that precedes it. By default, the preceding PCIe devices are 2 PFs + RShim DMA + 127 VFs of the first PF. This means that the maximum valid number of VFs for the second port will be 126.
	Workaround: Use the maximum allowed VFs on the 2nd PCIe PF of BlueField instead of the maximum of 127 VFs.
	Keywords: Emulated devices; VirtIO-net; VirtIO-blk; VFs; RShim
	Reported in version: 3.6.0.11699
2445289	Description: If secure boot is enabled, MFT cannot be installed on the BlueField DPU independently from BlueField drivers (MLNX_OFED).
	Workaround: N/A
	Keywords: MFT; secure boot
	Reported in version: 3.5.1.11601
2377021	Description: Executing `sudo poweroff` on the Arm side causes the system to hang.
	Workaround: Perform graceful shutdown, then reboot your BlueField device or power cycle the server.
	Keywords: Hang; reboot
	Reported in version: 3.5.0.11563
2350132	Description: Boot process hangs at BIOS (version 1.2.11) stage when power cycling a server (model Dell PowerEdge R7525) after configuring "PCI_SWITCH_EMULATION_NUM_PORT" > 27.
	Workaround: N/A
	Keywords: Server; hang; power cycle
	Reported in version: 3.5.0.11563

Ref #	Issue
2581408	Description: On a BlueField device operating in Embedded CPU mode, PXE driver will fail to boot if the Arm side is not fully loaded and the OVS bridge is not configured.
	Workaround: Run warm reboot on the host side and boot again via the device when Arm is up and the OVS bridge is configured.
	Keywords: Embedded CPU; PXE; UEFI; Arm
	Reported in version: 2.5.0.11176
1859322	Description: On some setups, DPU does not power on following server cold boot when UART cable is attached to the same server.
	Workaround: As long as the RShim driver is loaded on the server and the RShim interface is visible, the RShim driver will detect this and auto-reset the card into normal state.
	Keywords: DPU; Arm; Cold Boot
	Reported in version: 2.4.0.11082
1899921	Description: Driver restart fails when SNAP service is running.
	Workaround: Stop the SNAP services nvme_sf and nvme_snap@nvme0, then restart the driver. After the driver loads restart the services.
	Keywords: SNAP
	Reported in version: 2.2.0.11000
1911618	Description: Defining namespaces with certain Micron disks (Micron_9300_MTFDHAL3T8TDP) using consecutive attach-ns commands can cause errors.
	Workaround: Add delay between attach-ns commands.
	Keywords: Micron; disk; namespace; attach-ns
	Reported in version: 2.2.0.11000

3.5 Validated and Supported Cables and Modules

3.5.1 Cables Lifecycle Legend

Lifecycle Phase	Definition
EOL	End of Life
LTB	Last Time Buy
HVM	GA level
MP	GA level
P-Rel	GA level
Preliminary	Engineering Sample
Prototype	Engineering Sample

3.5.2 Supported Cables and Modules for BlueField-3

Unable to render include or excerpt-include. Could not retrieve page.

3.5.3 Supported Cables and Modules for BlueField-2

Unable to render include or excerpt-include. Could not retrieve page.

3.6 Release Notes Change Log History

3.6.1 Changes and New Features in 4.6.0

- Updated minimum UEFI password requirements
- Included DPU BMC firmware as part of the BFB image
- Added virtio-net support for plugging/unplugging parallel devices
- Implemented virtio debug enhancements

3.6.2 Changes and New Features in 4.5.0

- Added Redfish support for configuring all UEFI secure boot settings (disable, enable, enroll user keys, etc.) at scale, remotely, and securely
- For FHHL DPUs, added support for performing PCIe bifurcation configuration via MFT tool

  > Only a subset of configurations are supported.

- Updated the print of the manufacturing (MFG) setting, `MFG_OOB_MAC`, displayed by the command `bfcfg -d` to appear in lower-case to align with standard Linux tools

3.6.3 Changes and New Features in 4.2.0

> **Important!**
>
> Upgrading to this BSP version installs a new version of Ubuntu GRUB. This version of GRUB revokes the old UEFI secure boot certificates and install new ones. The new certificates will not validate older images and boot will fail. Therefore, to roll back to older software versions, users must disable UEFI secure boot.

- BFB installation chooses the on-chip NVMe (`/dev/nvme0n1`) by default for the EFI system partition and Linux rootfs installation and can be overloaded with `device=/dev/mmcblk0` in `bf.cfg` to push together with the BFB.

Installing on NVMe causes DPU booting to stay at the UEFI shell when changing to Livefish mode.

A previously installed OS on the eMMC device stays intact. Only the EFI boot entry is updated to boot from the SSD device.

3.6.4 Changes and New Features in 4.0.3

- BlueField-3 tuning update for power and performance

3.6.5 Changes and New Features in 4.0.2

- BlueField-3 power-capping and thermal-throttling
- Added Linux `fsck` to boot flow
- Log PCIe errors (to RShim log)
- Halt uncorrectable double-bit ECC error on DDR

3.6.6 Changes and New Features in 3.9.3

- Added support for live migration of VirtIO-net and VirtIO-blk VFs from one VM to another. Requires working with the new vDPA driver.
- OS configuration - enabled tmpfs in `/tmp`

3.6.7 Changes and New Features in 3.9.2

- Added support for Arm host
- Enroll new NVIDIA certificates to DPU UEFI database

Important: User action required! See known issue #3077361 for details.

3.6.8 Changes and New Features in 3.9.0

This is the last release to offer GA support for first-generation NVIDIA® BlueField® DPUs.

- Added support for NIC mode of operation
- Added password protection to change boot parameters in GRUB menu
- Added IB support for DOCA runtime and dev environment
- Implemented RShim PF interrupts
- Virtio-net-controller is split to 2 processes for fast recovery after service restart
- Added support for live virtio-net controller upgrade instead of performing a full restart
- Expanded BlueField-2 PCIe bus number range to 254 (0-253)

- Added a new CAP field, `log_max_queue_depth` (value can be set to `2K` / `4K`), to indicate the maximal NVMe SQ and CQ sizes supported by firmware. This can be used by NVMe controllers or by non-NVMe drivers which do not rely on NVMe CAP field.
- Added ability for the RShim driver to still work when the host is in secure boot mode
- Added `bfb-info` command which provides the breakdown of the software components bundled in the BFB package
- Added support for rate limiting VF groups

3.6.9 Changes and New Features in 3.8.5

- PXE boot option is enabled automatically and is available for the ConnectX and OOB network interfaces
- Added Vendor Class option "BF2Client" in DHCP request for PXE boot to identify card
- Updated the "force PXE" functionality to continue to retry PXE boot entries until successful. A configuration called "boot override retry" has been added. With this configured, UEFI does not rebuild the boot entries after all boot options are attempted but loops through the PXE boot options until booting is successful. Once successful, the boot override entry configuration is disabled and would need to be reenabled for future boots.
- Added ability to change the CPU clock dynamically according to the temperature and other sensors of the DPU. If the power consumption reaches close to the maximum allowed, the software module decreases the CPU clock rate to ensure that the power consumption does not cross the system limit.

  This feature is relevant only for OPNs MBF2H516C-CESOT, MBF2M516C-EECOT, MBF2H516C-EESOT, and MBF2H516C-CECOT.

- Bug fixes

3.6.10 Changes and New Features in 3.8.0

- Added ability to perform warm reboot on BlueField-2 based devices
- Added support for DPU BMC with OpenBMC
- Added support for NVIDIA Converged Accelerator (900-21004-0030-000)

3.7 Bug Fixes History

Ref #	Issue Description
3660460	Description: Ubuntu kernel 5.15.0-88-generic backports a bug from the upstream kernel which results in virtio-net full emulation not functioning.
	Keywords: Kernel
	Fixed in version: 4.6.0
3695367	Description: For BlueField-2, although an option to configure "large ICM size" appears in the UEFI menu it is not functional as large ICM size is not supported on it.
	Keywords: UEFI
	Fixed in version: 4.6.0

Ref #	Issue Description
3571285	Description: Intermittent UEFI/grub exception after many power-cycles:

```
Call Stack: Synchronous Exception at 0xF4B72E0C

ERR[UEFI]: PC=0xF4B72E0C
ERR[UEFI]: PC=0xF4B72E70
ERR[UEFI]: PC=0xF4B73570
ERR[UEFI]: PC=0xF4B74904
ERR[UEFI]: PC=0xF4F04444
ERR[UEFI]: PC=0xF4F044F8
ERR[UEFI]: PC=0xF4F05160
ERR[UEFI]: PC=0xF4F02030
ERR[UEFI]: PC=0xFDFC3A38 (0xFDFB0000+0x13A38) [ 1] DxeCore.dll
ERR[UEFI]: PC=0xF56E3594 (0xF56D4000+0xF594) [ 2] BdsDxe.dll
ERR[UEFI]: PC=0xF56F1FFC (0xF56D4000+0x1DFFC) [ 2] BdsDxe.dll
ERR[UEFI]: PC=0xF56F40D4 (0xF56D4000+0x200D4) [ 2] BdsDxe.dll
ERR[UEFI]: PC=0xFDFC6E50 (0xFDFB0000+0x16E50) [ 3] DxeCore.dll
ERR[UEFI]: PC=0x880092E0
ERR[UEFI]: PC=0x8800947C
ERR[UEFI]: X0=0x0 X1=0xF4B78FC3 X2=0xE X3=0x0
ERR[UEFI]: X4=0x0 X5=0xFFFFFFFFFFFFFFF8 X6=0x0 X7=0xFFFFFFF5
ERR[UEFI]: X8=0xF4B79480 X9=0x2 X10=0xFFFFFFFFFFFFFFFF X11=0xFFFFDC00
```

Ref #	Issue Description
	Keyword: Security
	Fixed in version: 4.5.0
3599839	Description: On a reboot following BFB install, the error message "Boot Image update completed, Status: Volume Corrupt" is observed. The error is non-functional and may be safely ignored.
	Keyword: Software provisioning; EFI capsule update; eMMC boot partitions
	Fixed in version: 4.5.0
3556795	Description: The first uplink representor interface may not be renamed to `p0` from `ethX` .
	Keyword: Representors
	Fixed in version: 4.5.0
3629875	Description: Fixed base address of static ICM.
	Keyword: ICM
	Fixed in version: 4.5.0
3365363	Description: On BlueField-3, when booting virtio-net emulation device using a GRUB2 bootloader, the bootloader may attempt to close and re-open the virtio-net device. This can result in unexpected behavior and possible system failure to boot.
	Keywords: BlueField-3; virtio-net; UEFI
	Fixed in version: 4.5.0
3373849	Description: Different OVS-based packages can include their own systemd services which prevents `/sbin/mlnx_bf_configure` from identifying the right one.
	Keywords: OVS; systemd
	Fixed in version: 4.5.0
3605332	Description: A dmseg is printed due to the OVS bridge interface being configured DOWN by default.
	Keyword: OVS
	Fixed in version: 4.2.1
3479040	Description: For non-LSO data, a max chain of 4 descriptors is posted onto the send queue resulting in a partial packet going out on the wire.
	Keyword: Send; LSO
	Fixed in version: 4.2.1

Ref #	Issue Description
3549785	Description: NVMe and mlx5_core drivers fail during BFB installation. As a result, Anolis OS cannot be installed on the SSD and the `mlxfwreset` command does not work during Anolis BFB installation.
	Keyword: Linux; NVMe; BFB installation
	Fixed in version: 4.2.1
3393316	Description: When LSO is enabled, if the header and data appear in the same fragment, the following warning is given from tcpdump: `truncated-ip - 9 bytes missing`
	Keyword: Virtio-net; large send offload
	Fixed in version: 4.2.1
3554128	Description: "`dmidecode`" output does not match "`ipmitool fru print`" output.
	Keywords: IPMI; print
	Fixed in version: 4.2.1
3508018	Description: Failure to ssh to Arm via 1GbE OOB interface is experienced after performing warm reboot on the DPU.
	Keywords: SSH; reboot
	Fixed in version: 4.2.0
3451539	Description: BSP build number (fourth digit in version number) does not appear in UEFI menu.
	Keywords: UEFI; software
	Fixed in version: 4.2.0
3259805	Description: Following many power cycles on the BlueField DPU, the virtio-net controller may fail to start with the error `failed to register epoll` in the log.
	Keywords: Virtio-net; power cycle; epoll
	Fixed in version: 4.2.0
3266180	Description: Enabled reset on MMC to enhance recovery on error.
	Keywords: MMC; reset
	Fixed in version: 4.2.0
3448217	Description: The PKA engine is not working on CentOS 7.6 due to multiple OpenSSL versions (1.0.2k 1.1.1k) being installed and the library loader not selecting the correct version of the openssl library.
	Keywords: PKA; OpenSSL
	Fixed in version: 4.2.0
3448228	Description: On virtio-net devices with LSO (large send offload) enabled, bogus packets may be captured on the SF representor when running heavy `iperf` traffic.
	Keywords: Virtio-net; iperf
	Fixed in version: 4.2.0
3452583	Description: OpenSSL is not working with PKA engine on CentOS 7.6 with 4.23 5.4 5.10 kernels due to multiple versions of OpenSSL(1.0.2k and 1.1.1k) are installed.
	Keywords: OpenSSL; PKA

Ref #	Issue Description
	Fixed in version: 4.2.0
3455873	Description: 699140280000 OPN is not supported.
	Keywords: SKU; support
	Fixed in version: 4.2.0
3519341	Description: Populate the vGIC maintenance interrupt number in MADT to avoid harmless.
	Keywords: Error
	Fixed in version: 4.2.0
3522652	Description: The timer frequency is measured using the c0 fmon feature causing new kernels to complain if CNTFRQ_EL0 has a different value on different cores.
	Keywords: Timer frequency
	Fixed in version: 4.2.0
3531965	Description: Memory info displayed via `dmidecode` is not correct for memory sizes 32G and above.
	Keywords: Memory; dmidecode
	Fixed in version: 4.2.0
3362181	Description: A customized BFB with an older kernel does not support bond speed above 200Gb/s.
	Keywords: Bond; LAG; speed
	Fixed in version: 4.2.0
3177569	Description: DCBX configuration may not take effect.
	Keywords: DCBX; QoS; lldpad
	Fixed in version: 4.2.0
2824859	Description: Hotplug/unplug of virtio-net devices during host shutdown/bootup may result in failure to do plug/unplug.
	Keywords: Virtio-net, hotplug
	Fixed in version: 4.2.0
3252083	Description: Assert errors may be observed in the RShim log after reset/reboot. These errors are harmless and may be ignored.
	Keywords: RShim; log; error
	Fixed in version: 4.0.3
3240060	Description: Hotplug of a modern virtio-net device is not supported when `VIRTIO_EMULATION_HOTPLUG_TRANS` is `TRUE` from mlxconfig.
	Keywords: Virtio-net; hotplug; legacy
	Fixed in version: 4.0.3
3240182	Description: Virtio-net full emulation is not supported in CentOS 8.2 with inbox-kernel 4.18.0-193.el8.aarch64.
	Keywords: Virtio-net; CentOS
	Fixed in version: 4.0.3

Ref #	Issue Description
3151884	Description: If secure boot is enabled, the following error message is observed while installing Ubuntu on the DPU: `ERROR: need to use capsule in secure boot mode`. This message is harmless and may be safely ignored.
	Keywords: Error message; installation
	Fixed in version: 3.9.3
2793005	Description: When Arm reboots or crashes after sending a virtio-net unplug request, the hotplugged devices may still be present after Arm recovers. The host, however, will not see those devices.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.9.3
3107227	Description: BlueField with secured BFB fails to boot up if the `PART_SCHEME` field is set in `bf.cfg` during installation.
	Keywords: Installation; bf.cfg
	Fixed in version: 3.9.2
3109270	Description: If the RShim service is running on an external host over the PCIe interface then, in very rare cases, a soft reset of the BlueField can cause a poisoned completion to be returned to the host. The host may treat this as a fatal error and crash.
	Keywords: RShim; ATF
	Fixed in version: 3.9.2
2790928	Description: Virtio-net-controller recovery may not work for a hot-plugged device because the system assigns a BDF (string identifier) of 0 for the hot-plugged device, which is an invalid value.
	Keywords: Virtio-net; hotplug; recovery
	Fixed in version: 3.9.0
2780819	Description: Eye-opening is not supported on 25GbE integrated-BMC BlueField-2 DPU.
	Keywords: Firmware, eye-opening
	Fixed in version: 3.9.0
2876447	Description: Virtio full emulation is not supported by NVIDIA® BlueField®-2 multi-host cards.
	Keywords: Virtio full emulation; multi-host
	Fixed in version: 3.9.0
2855485	Description: After BFB installation, Linux crash may occur with `efi_call_rts` messages in the call trace which can be seen from the UART console.
	Keywords: Linux crash; `efi_call_rts`
	Fixed in version: 3.9.0
2901514	Description: Relaxed ordering is not working properly on virtual functions.
	Keywords: MLNX_OFED; relaxed ordering; VF
	Fixed in version: 3.9.0
2852086	Description: On rare occasions, the UEFI variables in UVPS EEPROM are wiped out which hangs the boot process at the UEFI menu.
	Keywords: UEFI; hang
	Fixed in version: 3.9.0

Ref #	Issue Description
2934828	Description: PCIe device address to RDMA device name mapping on x86 host may change after the driver restarts in Arm.
	Keywords: RDMA; Arm; driver
	Fixed in version: 3.9.0
-	Description: RShim driver does not work when the host is in secure boot mode.
	Keywords: RShim; Secure Boot
	Fixed in version: 3.9.0
2787308	Description: At rare occasions during Arm reset on BMC-integrated DPUs, the DPU will send "PCIe Completion" marked as poisoned. Some servers treat that as fatal and may hang.
	Keywords: Arm reset; BMC integrated
	Fixed in version: 3.9.0
2585607	Description: Pushing the BFB image fails occasionally with a "bad magic number" error message showing up in the console.
	Keywords: BFB push; installation
	Fixed in version: 3.9.0
2802943	Description: SLD detection may not function properly.
	Keywords: Firmware
	Fixed in version: 3.9.0
2580945	Description: External host reboot may also reboot the Arm cores if the DPU was configured using mlxconfig.
	Keywords: Non-volatile configuration; Arm; reboot
	Fixed in version: 3.9.0
2899740	Description: BlueField-2 may sometimes go to PXE boot instead of Linux after installation.
	Keywords: Installation; PXE
	Fixed in version: 3.8.5
2870143	Description: Some DPUs may get stuck at GRUB menu when booting due to the GRUB configuration getting corrupted when board is powered down before the configuration is synced to memory.
	Keywords: GRUB; memory
	Fixed in version: 3.8.5
2873700	Description: The available RShim logging buffer may not have enough space to hold the whole register dump which may cause buffer wraparound.
	Keywords: RShim; logging
	Fixed in version: 3.8.5
2801891	Description: IPMI EMU service reports cable link as down when it is actually up.
	Keywords: IPMI EMU
	Fixed in version: 3.8.0
2779861	Description: Virtio-net controller does not work with devices other than `mlx5_0/1`.
	Keywords: Virtio-net controller

Ref #	Issue Description
	Fixed in version: 3.8.0
2801378	Description: No parameter validation is done for feature bits when performing hotplug.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.8.0
2802917	Description: When secure boot is enabled, PXE boot may not work.
	Keywords: Secure boot; PXE
	Fixed in version: 3.8.0
2827413	Description: Updating a BFB could fail due to congestion.
	Keywords: Installation; congestion
	Fixed in version: 3.8.0
2829876	Description: For virtio-net device, modifying the number of queues does not update the number of MSIX.
	Keywords: Virtio-net; queues
	Fixed in version: 3.8.0
2597790	Description: A "double free" error is seen when using the "curl" utility. This happens only when OpenSSL is configured to use a dynamic engine (e.g. Bluefield PKA engine).
	Keywords: OpenSSL; curl
	Fixed in version: 3.8.0
2853295	Description: UEFI secure boot enables the kernel lockdown feature which blocks access by mstmcra.
	Keywords: Secure boot
	Fixed in version: 3.8.0
2854472	Description: Virtio-net controller may fail to start after power cycle.
	Keywords: Virtio-net controller
	Fixed in version: 3.8.0
2854995	Description: Memory consumed for a representor exceeds what is necessary making scaling to 504 SF's not possible.
	Keywords: Memory
	Fixed in version: 3.8.0
2856652	Description: Modifying VF bits yields an error.
	Keywords: Virtio-net controller
	Fixed in version: 3.8.0
2859066	Description: Arm hangs when user is thrown to livefish by FW (e.g. secure boot).
	Keywords: Arm; livefish
	Fixed in version: 3.8.0
2866082	Description: The current installation flow requires multiple resets after booting the self-install BFB due to the watchdog being armed after capsule update.
	Keywords: Reset; installation
	Fixed in version: 3.8.0

Ref #	Issue Description
2866537	Description: Power-off of BlueField shows up as a panic which is then stored in the RShim log and carried into the BERT table in the next boot which is misleading to the user.
	Keywords: RShim; log; panic
	Fixed in version: 3.8.0
2868944	Description: Various errors related to the UPVS store running out of space are observed.
	Keywords: UPVS; errors
	Fixed in version: 3.8.0
2754798	Description: `oob_net0` cannot receive traffic after a network restart.
	Keywords: `oob_net0`
	Fixed in version: 3.8.0
2691175	Description: Up to 31 hot-plugged virtio-net devices are supported even if `PCI_SWITCH_EMULATION_NUM_PORT=32`. Host may hang if it hot plugs 32 devices.
	Keywords: Virtio-net; hotplug
	Fixed in version: 3.8.0

2597973	Description: Working with CentOS 7.6, if SF network interfaces are statically configured, the following parameters should be set.
	`NM_CONTROLLED="no"`
	`DEVTIMEOUT=30`
	For example:

```
# cat /etc/sysconfig/network-scripts/ifcfg-p0m0
NAME=p0m0
DEVICE=p0m0
NM_CONTROLLED="no"
PEERDNS="yes"
ONBOOT="yes"
BOOTPROTO="static"
IPADDR=12.212.10.29
BROADCAST=12.212.255.255
NETMASK=255.255.0.0
NETWORK=12.212.0.0
TYPE=Ethernet
DEVTIMEOUT=30
```

Ref #	Issue Description
	Keywords: CentOS; subfunctions; static configuration
	Fixed in version: 3.7.0
2581534	Description: When shared RQ mode is enabled and offloads are disabled, running multiple UDP connections from multiple interfaces can lead to packet drops.
	Keywords: Offload; shared RQ
	Fixed in version: 3.7.0
2581621	Description: When OVS-DPDK and LAG are configured, the kernel driver drops the LACP packet when working in shared RQ mode.
	Keywords: OVS-DPDK; LAG; LACP; shared RQ
	Fixed in version: 3.7.0
2601094	Description: The gpio-mlxbf2 and mlxbf-gige drivers are not supported on 4.14 kernel.
	Keywords: Drivers; kernel
	Fixed in version: 3.7.0

Ref #	Issue Description
2584427	Description: Virtio-net-controller does not function properly after changing uplink representor MTU.
	Keywords: Virtio-net controller; MTU
	Fixed in version: 3.7.0
2438392	Description: VXLAN with IPsec crypto offload does not work.
	Keywords: VXLAN; IPsec crypto
	Fixed in version: 3.7.0
2406401	Description: Address Translation Services is not supported in BlueField-2 step A1 devices. Enabling ATS can cause server hang.
	Keywords: ATS
	Fixed in version: 3.7.0
2402531	Description: PHYless reset on BlueField-2 devices may cause the device to disappear.
	Keywords: PHY; firmware reset
	Fixed in version: 3.7.0
2400381	Description: When working with strongSwan 5.9.0bf, running `ip xfrm state show` returns partial information as to the offload parameters, not showing "`mode full`".
	Keywords: strongSwan; ip xfrm; IPsec
	Fixed in version: 3.7.0
2392604	Description: Server crashes after configuring PCI_SWITCH_EMULATION_NUM_PORT to a value higher than the number of PCIe lanes the server supports.
	Keywords: Server; hang
	Fixed in version: 3.7.0
2293791	Description: Loading/reloading NVMe after enabling VirtIO fails with a PCI bar memory mapping error.
	Keywords: VirtIO; NVMe
	Fixed in version: 3.7.0
2245983	Description: When working with OVS in the kernel and using Connection Tracking, up to 500,000 flows may be offloaded.
	Keywords: DPU; Connection Tracking
	Fixed in version: 3.7.0
1945513	Description: If the Linux OS running on the host connected to the BlueField DPU has a kernel version lower then 4.14, MLNX_OFED package should be installed on the host.
	Keywords: Host OS
	Fixed in version: 3.7.0
1900203	Description: During heavy traffic, ARP reply from the other tunnel endpoint may be dropped. If no ARP entry exists when flows are offloaded, they remain stuck on the slow path.
	Workaround: Set a static ARP entry at the BlueField Arm to VXLAN tunnel endpoints.
	Keywords: ARP; Static; VXLAN; Tunnel; Endpoint
	Fixed in version: 3.7.0

Ref #	Issue Description
2082985	Description: During boot, the system enters systemctl emergency mode due a corrupt root file system.
	Keywords: Boot
	Fixed in version: 3.6.0.11699
2278833	Description: Creating a bond via NetworkManager and restarting the driver (openibd restart) results in no pf0hpf and bond creation failure.
	Keywords: Bond; LAG; network manager; driver reload
	Fixed in version: 3.6.0.11699
2286596	Description: Only up to 62 host virtual functions are currently supported.
	Keywords: DPU; SR-IOV
	Fixed in version: 3.6.0.11699
2397932	Description: Before changing SR-IOV mode or reloading the mlx5 drivers on IPsec-enabled systems, make sure all IPsec configurations are cleared by issuing the command `ip x s f && ip x p f`.
	Keywords: IPsec; SR-IOV; driver
	Fixed in version: 3.6.0.11699
2405039	Description: In Ubuntu, during or after a reboot of the Arm, manually, or as part of a firmware reset, the network devices may not transition to switchdev mode. No device representors would be created (pf0hpf, pf1hpf, etc). Driver loading on the host will timeout after 120 seconds.
	Keywords: Ubuntu; reboot; representors; switchdev
	Fixed in version: 3.6.0.11699
2403019	Description: EEPROM storage for UEFI variables may run out of space and cause various issues such as an inability to push new BFB (due to timeout) or exception when trying to enter UEFI boot menu.
	Keywords: BFB install; timeout; EEPROM UEFI Variable; UVPS
	Fixed in version: 3.6.0.11699

2458040	Description: When using OpenSSL on BlueField platforms where Crypto support is disabled, the following errors may be encountered:
	`PKA_ENGINE: PKA instance is invalid`
	`PKA_ENGINE: failed to retrieve valid instance`
	This happens due to OpenSSL configuration being linked to use PKA hardware, but that hardware is not available since crypto support is disabled on these platforms.

	Keywords: PKA; Crypto
	Fixed in version: 3.6.0.11699
2456947	Description: All NVMe emulation counters (Ctrl, SQ, Namespace) return "0" when queried.
	Keywords: Emulated devices; NVMe
	Fixed in version: 3.6.0.11699
2411542	Description: Multi-APP QoS is not supported when LAG is configured.
	Keywords: Multi-APP QoS; LAG
	Fixed in version: 3.6.0.11699
2394130	Description: When creating a large number of VirtIO VFs, hung task call traces may be seen in the dmesg.

Ref #	Issue Description
	Keywords: VirtIO; call traces; hang
	Fixed in version: 3.5.1.11601
2398050	Description: Only up to 60 virtio-net emulated virtual functions are supported if LAG is enabled.
	Keywords: Virtio-net; LAG
	Fixed in version: 3.5.1.11601
2256134	Description: On rare occasions, rebooting the BlueField DPU may result in traffic failure from the x86 host.
	Keywords: Host; Arm
	Fixed in version: 3.5.1.11601
2400121	Description: When emulated PCIe switch is enabled, and more than 8 PFs are enabled, the BIOS boot process might halt.
	Keywords: Emulated PCIe switch
	Fixed in version: 3.5.0.11563
2082985	Description: During boot, the system enters systemctl emergency mode due a corrupt root file system.
	Keywords: Boot
	Fixed in version: 3.5.0.11563
2249187	Description: With the OCP card connecting to multiple hosts, one of the hosts could have the RShim PF exposed and probed by the RShim driver.
	Keywords: RShim; multi-host
	Fixed in version: 3.5.0.11563
2363650	Description: When moving to separate mode on the DPU, the OVS bridge remains and no ping is transmitted between the Arm cores and the remote server.
	Keywords: SmartNIC; operation modes
	Fixed in version: 3.5.0.11563
2394226	Description: Pushing the BFB image v3.5 with a WinOF-2 version older than 2.60 can cause a crash on the host side.
	Keywords: Windows; RShim
	Fixed in version: 3.5.0.11563

4 BlueField Software Overview

NVIDIA provides software which enables users to fully utilize the NVIDIA® BlueField® DPU and enjoy the rich feature-set it provides. Using the BlueField software packages, users can:

- Quickly and easily boot an initial Linux image on your development board
- Port existing applications to and develop new applications for BlueField
- Patch, configure, rebuild, update or otherwise customize your image
- Debug, profile, and tune their development system using open-source development tools taking advantage of the diverse and vibrant Arm ecosystem

Coupled with the NVIDIA® ConnectX® interconnect, the BlueField family of DPU devices includes an array of Arm cores according to the following:

- 64-bit Armv8 A72 for BlueField-2 DPUs
- 64-bit Armv8 A78 for BlueField-3 DPUs

Standard Linux distributions run on the Arm cores allowing common open-source development tools to be used. Developers should find the programming environment familiar and intuitive which in turn allows them to design, implement, and verify their control-plane and data-plane applications quickly and efficiently.

BlueField SW ships with the NVIDIA® BlueField® Reference Platform. BlueField SW is a reference Linux distribution based on the Ubuntu Server distribution extended to include the MLNX_OFED stack for Arm and a Linux kernel which supports NVMe-oF. This software distribution can run all customer-based Linux applications seamlessly.

The following are other software elements delivered with BlueField DPU:

- Arm Trusted Firmware (ATF) for BlueField
- UEFI for BlueField
- OpenBMC for BMC (ASPEED 2500) found on development board
- MLNX_OFED stack
- Mellanox MFT

4.1 Debug Tools

BlueField DPU includes hardware support for the Arm DS5 suite as well as CoreSight™ debug. As such, a wide range of commercial off-the-shelf Arm debug tools should work seamlessly with BlueField. The CoreSight debugger interface can be accessed via RShim interface (USB or PCIe if using DPU) as well which could be used for debugging with open-source tools like OpenOCD.

The BlueField DPU also supports the ubiquitous GDB.

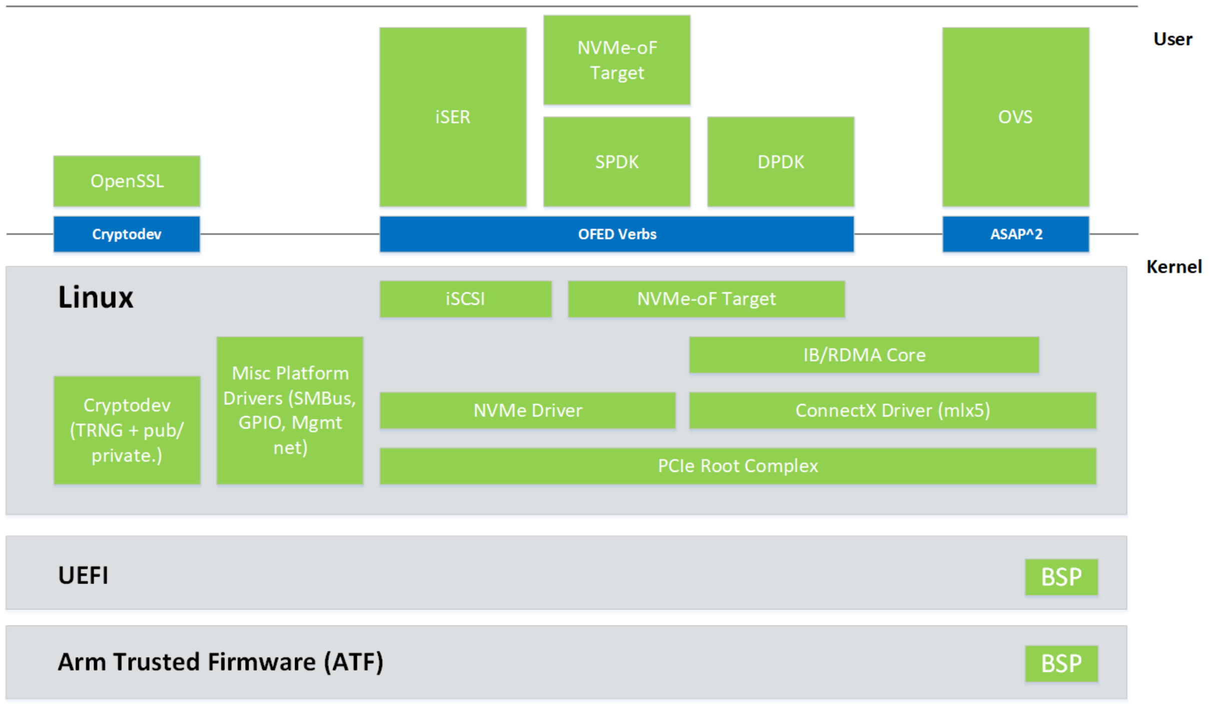
4.2 BlueField-based Storage Appliance

BlueField software provides the foundation for building a JBOF (Just a Bunch of Flash) storage system including NVMe-oF target software, PCIe switch support, NVDIMM-N support, and NVMe disk hot-swap support.

BlueField SW allows enabling ConnectX offload such as RDMA/RoCE, T10 DIF signature offload, erasure coding offload, iSER, Storage Spaces Direct, and more.

4.3 BlueField Architecture

The BlueField architecture is a combination of two preexisting standard off-the-shelf components, Arm AArch64 processors, and ConnectX-6 Dx (for BlueField-2), ConnectX-7 (for BlueField-3), or network controller, each with its own rich software ecosystem. As such, almost any of the programmer-visible software interfaces in BlueField come from existing standard interfaces for the respective components.

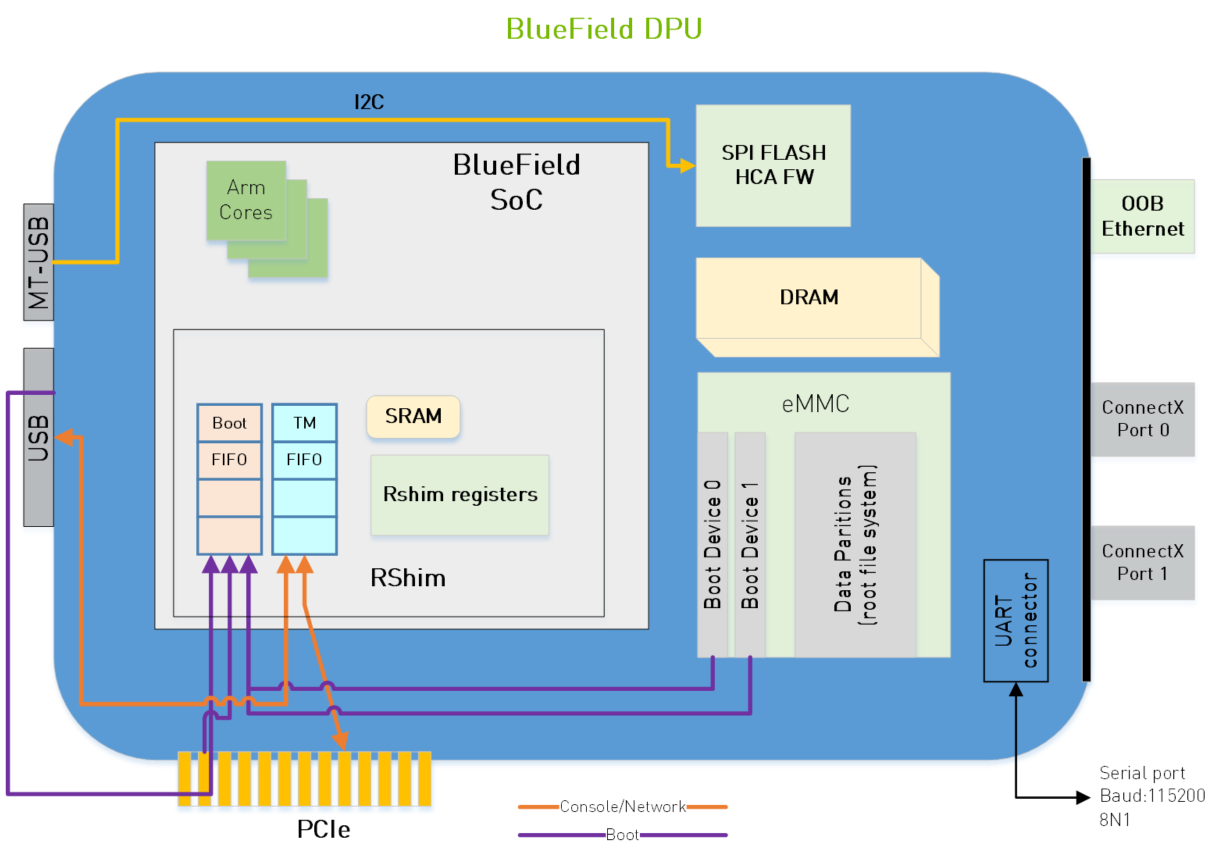


The Arm related interfaces (including those related to the boot process, PCIe connectivity, and cryptographic operation acceleration) are standard Linux on Arm interfaces. These interfaces are enabled by drivers and low-level code provided by NVIDIA as part of the BlueField software delivered and upstreamed to respective open-source projects, such as Linux.

The ConnectX network controller-related interfaces (including those for Ethernet and InfiniBand connectivity, RDMA and RoCE, and storage and network operation acceleration) are identical to the interfaces that support ConnectX standalone network controller cards. These interfaces take advantage of the MLNX_OFED software stack and InfiniBand verbs-based interfaces to support software.

4.4 System Connections

The BlueField DPU has multiple connections (see diagram below). Users can connect to the system via different consoles, network connections, and a JTAG connector.



4.4.1 System Consoles

The BlueField DPU has multiple console interfaces:

- Serial console 0 (`/dev/ttyAMA0` on the Arm cores)
  - Requires cable to NC-SI connector on DPU 25G
  - Requires serial cable to 3-pin connector on DPU 100G
  - Connected to BMC serial port on BF1200 platforms
- Serial console 1 (`/dev/ttyAMA1` on the Arm cores but only for BF1200 reference platform)
  - ttyAMA1 is the console connection on the front panel of the BF1200
- Virtual RShim console (`/dev/hvc0` on the Arm cores) is driven by
  - The RShim PCIe driver (does not require a cable but the system cannot be in isolation mode as isolation mode disables the PCIe device needed)
  - The RShim USB driver (requires USB cable)
  - It is not possible to use both the PCIe and USB RShim interfaces at the same time

4.4.2 Network Interfaces

The DPU has multiple network interfaces.

- ConnectX Ethernet/InfiniBand interfaces
- RShim virtual Ethernet interface (via USB or PCIe)
  The virtual Ethernet interface can be very useful for debugging, installation, or basic management. The name of the interface on the host DPU server depends on the host

operating system. The interface name on the Arm cores is normally "tmfifo_net0". The virtual network interface is only capable of roughly 10MB/s operation and should not be considered for production network traffic.

- OOB Ethernet interface
  BlueField-2 based platforms feature an OOB 1GbE management port. This interface provides a 1Gb/s full duplex connection to the Arm cores. The interface name is normally "oob_net0". The interface enables TCP/IP network connectivity to the Arm cores (e.g., for file transfer protocols, SSH, and PXE boot). The OOB port is not a path for the BlueField-2 boot stream (i.e., any attempt to push a BFB to this port will not work).

5 Software Installation and Upgrade

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

The NVIDIA® BlueField® DPU is shipped with the BlueField software based on Ubuntu 22.04 pre-installed. The DPU's Arm execution environment has the capability of being functionally isolated from the host server and uses a dedicated network management interface (separate from the host server's management interface). The Arm cores can run the Open vSwitch Database (OVSDB) or other virtual switches to create a secure solution for bare metal provisioning.

The software package also includes support for DPDK as well as applications for accelerated encryption.

The BlueField DPU supports several methods for OS deployment and upgrade:

- Full OS image deployment using a BlueField boot stream file (BFB) via RShim interface

  This installation method is compatible with SuperNICs.

- Full OS deployment using PXE which can be used over different network interfaces available on the BlueField DPU (1GbE mgmt, tmfifo or NVIDIA® ConnectX®)
- Individual packages can be installed or upgraded using standard Linux package management tools (e.g., apt, dpkg, etc.)

The DPU's BMC software (i.e., BMC firmware, ERoT firmware, DPU golden image, and NIC firmware golden image) is included in the BFB. The BFB installation updates BMC software components automatically if BMC credentials (i.e., `BMC_USER` and `BMC_PASSWORD`) are provided in bf.cfg.

The minimum BMC Firmware version that supports this method of upgrade from the BlueField is 23.07. If your BMC firmware version is lower, follow the NVIDIA BlueField BMC Software documentation to upgrade BMC firmware.

Upgrading BlueField software using BFB Bundle now performs NIC firmware update by default.

A reduced size BFB `bf-fwbundle-<version>.prod.bfb` is available for BlueField devices running a customized OS that should not be changed by the BFB installation process. This BFB does not include BlueField OS and can use the same set of bf.cfg parameters as a standard BFB with the exception of BlueField OS related flags (e.g., `UPDATE_DPU_OS`).

5.1 Deploying BlueField Software Using BFB from Host

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

This procedure assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following table lists an overview of the steps required to install Ubuntu BFB on your DPU:

Step	Procedure	Link to Section
1	Uninstall previous DOCA on host (if exists)	Uninstall Previous Software from Host
2	Install RShim on the host	Install RShim on Host
3	Verify that RShim is running on the host	Ensure RShim Running on Host
4	Change the default credentials using `bf.cfg` file (optional)	Changing Default Credentials Using bf.cfg
5	Install the Ubuntu BFB image	BFB Installation
6	Verify installation completed successfully	Verify BFB is Installed
7	Upgrade the firmware on your DPU	Firmware Upgrade

5.1.1 Uninstall Previous Software from Host

If an older DOCA software version is installed on your host, make sure to uninstall it before proceeding with the installation of the new version:

Ubuntu	
	```
	host# for f in $( dpkg --list | grep doca | awk '{print $2}' ); do echo
	$f ; apt remove --purge $f -y ; done
	host# sudo apt-get autoremove
	```

CentOS/RHEL	
	```
	host# for f in $(rpm -qa |grep -i doca ) ; do yum -y remove $f; done
	host# yum autoremove
	host# yum makecache
	```

5.1.2 Install RShim on Host

Before installing the RShim driver, verify that the RShim devices, which will be probed by the driver, are listed under `lsusb` or `lspci`.

```
lspci | grep -i nox
```

Output example:

```
27:00.0 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller
27:00.1 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller
27:00.2 Non-Volatile memory controller: Mellanox Technologies NVMe SNAP Controller
27:00.3 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface // This is the RShim PF
```

RShim is compiled as part of the `doca-runtime` package in the `doca-host-repo-ubuntu<version>_amd64` file (`.deb` or `.rpm`).

To install `doca-runtime` :

OS	Procedure

Ubuntu/Debian	1. Download the DOCA Runtime host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the deb repo. Run:
	`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`
	3. Perform apt update. Run:
	`host# sudo apt-get update`
	4. Run `apt install` for DOCA runtime package.
	`host# sudo apt install doca-runtime`

CentOS/RHEL 7.x	1. Download the DOCA runtime host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	3. Enable new yum repos. Run:
	`host# sudo yum makecache`
	4. Run `yum install` to install DOCA runtime package.
	`host# sudo yum install doca-runtime`

CentOS/RHEL 8.x or Rocky 8.6	1. Download the DOCA runtime host package from the "Installation Files" section in the *NVIDIA DOCA Installation Guide for Linux*.
	2. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	3. Enable new dnf repos. Run:
	`host# sudo dnf makecache`
	4. Run `dnf install` to install DOCA runtime.
	`host# sudo dnf install doca-runtime`

5.1.3 Ensure RShim Running on Host

1. Verify RShim status. Run:

   ```
   sudo systemctl status rshim
   ```

   Expected output:

   ```
   active (running)
   ...
   Probing pcie-0000:<BlueField's PCIe Bus address on host>
   create rshim pcie-0000:<BlueField's PCIe Bus address on host>
   rshim<N> attached
   ```

   Where `<N>` denotes RShim enumeration starting with 0 (then 1, 2, etc.) for every additional DPU installed on the server.
   If the text "`another backend already attached`" is displayed, users will not be able to use RShim on the host. Please refer to "RShim Troubleshooting and How-Tos" to troubleshoot RShim issues.

   a. If the previous command displays `inactive` or another error, restart RShim service. Run:

      ```
      sudo systemctl restart rshim
      ```

   b. Verify RShim status again. Run:

      ```
      sudo systemctl status rshim
      ```

      If this command does not display "`active (running)`", then refer to "RShim Troubleshooting and How-Tos".

2. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        pcie-0000:04:00.2
   ```

   This output indicates that the RShim service is ready to use.

5.1.4 Installing Ubuntu on BlueField

5.1.4.1 BFB Installation

For BlueField-2 DPUs Only

Check the BFB version installed on your BlueField-2 DPU. If the version is 1.5.0 or lower, please see Known Issue Reference #3600716 under Known Issues section.

To upgrade the BMC firmware using BFB, the user must provide the current BMC credentials in the bf.cfg.

Upgrading the BlueField networking platform using BFB Bundle updates the NIC firmware by default. NIC firmware upgrade triggers a NIC reset flow via `mlxfwreset` in the BlueField Arm.

If this reset flow cannot complete or is not supported on your setup, `bfb-install` alerts about it at the end of the installation. In this case, perform a BlueField system reboot for the `mlxconfig` settings to take effect.

To skip NIC firmware upgrade during BFB Bundle installation, provide the parameter `WITH_NIC_FW_UPDATE=no` in the `bf.cfg` text file when running `bfb-install`.

A pre-built BFB of Ubuntu 22.04 with DOCA Runtime and DOCA packages installed is available on the NVIDIA DOCA SDK developer zone page.

All new BlueField-2 devices and all BlueField-3 are secure boot enabled, hence all the relevant SW images (ATF/UEFI, Linux Kernel and Drivers) must be signed in order to boot. All formally published SW images are signed.

When installing the BFB bundle in NIC mode, users must perform the following:

1. Prior to installing the BFB bundle, users must unbind each NIC port, using its PCIe function address. For example:

   ```
   host]# lspci -d 15b3:
   21:00.0 Ethernet controller: Mellanox Technologies MT43244 BlueField-3 integrated ConnectX-7
   network controller (rev 01)
   21:00.1 Ethernet controller: Mellanox Technologies MT43244 BlueField-3 integrated ConnectX-7
   network controller (rev 01)
   21:00.2 DMA controller: Mellanox Technologies MT43244 BlueField-3 SoC Management Interface (rev
   01)

   host]# echo 0000:21:00.0 > /sys/bus/pci/drivers/mlx5_core/unbind
   host]# echo 0000:21:00.1 > /sys/bus/pci/drivers/mlx5_core/unbind
   ```

   If there are multiple BlueField devices to be updated in the server, repeat this step on all of them, before starting BFB bundle installations.
2. After the BFB bundle installation is done, users must perform a warm reboot on the host.

To install Ubuntu BFB, run on the host side:

```
# bfb-install -h
syntax: bfb-install --bfb|-b <BFBFILE> [--config|-c <bf.cfg>] \
  [--rootfs|-f <rootfs.tar.xz>] --rshim|-r <rshimN> [--help|-h]
```

The `bfb-install` utility is installed by the RShim package.

This utility script pushes the BFB image and optional configuration (`bf.cfg` file) to the BlueField side and checks and prints the BFB installation progress. To see the BFB installation progress, please install the `pv` Linux tool.

BFB image installation must complete before restarting the system/BlueField. Doing so may result in the BlueField DPU not operating as expected (e.g., it may not be accessible using SSH). If this happens, re-initiate the update process with `bfb-install` to recover the DPU.

The following is an output example of Ubuntu 20.04 installation with the `bfb-install` script assuming `pv` has been installed.

```
# bfb-install --bfb <BlueField-BSP>.bfb --config bf.cfg --rshim rshim0 Pushing bfb + cfg
1.46GiB 0:01:11 [20.9MiB/s]
[                                                              <=>                        ]
Collecting BlueField booting status. Press Ctrl+C to stop…
 INFO[PSC]: PSC BL1 START
 INFO[BL2]: start
 INFO[BL2]: boot mode (rshim)
 INFO[BL2]: VDDQ: 1120 mV
 INFO[BL2]: DDR POST passed
 INFO[BL2]: UEFI loaded
 INFO[BL31]: start
 INFO[BL31]: lifecycle Production
 INFO[BL31]: MB8: VDD adjustment complete
 INFO[BL31]: VDD: 743 mV
 INFO[BL31]: power capping disabled
 INFO[BL31]: runtime
 INFO[UEFI]: eMMC init
 INFO[UEFI]: eMMC probed
 INFO[UEFI]: UPVS valid
 INFO[UEFI]: PMI: updates started
 INFO[UEFI]: PMI: total updates: 1
 INFO[UEFI]: PMI: updates completed, status 0
 INFO[UEFI]: PCIe enum start
 INFO[UEFI]: PCIe enum end
 INFO[UEFI]: UEFI Secure Boot (disabled)
 INFO[UEFI]: exit Boot Service
 INFO[MISC]: : Found bf.cfg
 INFO[MISC]: : Ubuntu installation started
 INFO[MISC]: bfb_pre_install
 INFO[MISC]: Installing OS image
 INFO[MISC]: : Changing the default password for user ubuntu
 INFO[MISC]: : Running bfb_modify_os from bf.cfg
 INFO[MISC]: : Ubuntu installation finished
```

5.1.4.2 Verify BFB is Installed

After installation of the Ubuntu OS is complete, the following note appears in `/dev/rshim0/misc` on first boot:

```
...
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
```

"DPU is ready" indicates that all the relevant services are up and users can login the system.

After the installation of the Ubuntu 20.04 BFB, the configuration detailed in the following sections is generated.

Make sure all the services (including cloud-init) are started on BlueField and to perform a graceful shutdown before power cycling the host server.

BlueField OS image version is stored under `/etc/mlnx-release` in the BlueField:

```
# cat /etc/mlnx-release
bf-bundle-2.7.0-<version>_ubuntu-22.04_prod
```

5.1.4.3 Changing Default Credentials Using bf.cfg

For a comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

Ubuntu users are prompted to change the default password (ubuntu) for the default user (ubuntu) upon first login. Logging in will not be possible even if the login prompt appears until all services are up ("`DPU is ready`" message appears in `/dev/rshim0/misc`).

Attempting to log in before all services are up prints the following message: `Permission denied, please try again.`

Alternatively, Ubuntu users can provide a unique password that will be applied at the end of the BFB installation. This password must be defined in a `bf.cfg` configuration file. To set the password for the `ubuntu` user:

1. Create password hash. Run:

   ```
   # openssl passwd -1
   Password:
   Verifying - Password:
   $1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1
   ```

2. Add the password hash in quotes to the `bf.cfg` file:

   ```
   # vim bf.cfg
   ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'
   ```

   The `bf.cfg` file is used with the `bfb-install` script in the steps that follow.

5.1.4.4 Password Policy

The following table provides the password policy parameters.

Config File Path	Parameter	Value	Description
`/etc/security/pwquality.conf`	`minlen`	12	Minimum password length
`/etc/pam.d/common-password`	`remember`	3	The number of previous passwords which cannot be reused
`/etc/security/faillock.conf`	`silent`	Uncommented	Prevents printing informative messages to the user
	`deny`	10	The number of authentication attempts permitted before the user is locked out
	`unlock_time`	600	The duration, in seconds, of the lockout period

Each of these parameters is configurable in its respective config file indicated in the "Config File Path" column.

Please refer to the "Default Passwords and Policies" section for more password policy information.

5.1.4.5 GRUB Password Protection

GRUB menu entries are protected by a username and password to prevent unwanted changes to the default boot options or parameters.

The default credentials are as follows:

Username	`admin`
Password	`BlueField`

The password can be changed during BFB installation by providing a new `grub_admin_PASSWORD` parameter in `bf.cfg`:

```
# vim bf.cfg
grub_admin_PASSWORD='
grub.pbkdf2.sha512.10000.5EB1FF92FDD89BDAF3395174282C77430656A6DBEC1F9289D5F5DAD17811AD0E2196D0E49B49EF31C21972669D
180713E265BB2D1D4452B2EA9C7413C3471C53.F533423479EE7465785CC2C79B637BDF77004B5CC16C1DDE806BCEA50BF411DE04DFCCE42279
E2E1F605459F1ABA3A0928CE9271F2C84E7FE7BF575DC22935B1'
```

To get a new encrypted password value use the command `grub-mkpasswd-pbkdf2`.

After the installation, the password can be updated by editing the file `/etc/grub.d/40_custom` and then running the command `update-grub` which updates the file `/boot/grub/grub.cfg`.

5.1.4.6 Firmware Upgrade

To upgrade firmware:

1. Access the BlueField using one of the available interfaces (RShim console, BMC console, SSH via `oob_net0` or `tmfifo_net0` interfaces).
2. Upgrade the firmware on the DPU. Run:

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

   Example output:

   ```
   Device #1:
   ----------

     Device Type:      BlueField-2
     [...]
     Versions:         Current        Available
        FW             <Old_FW>       <New_FW>
   ```

Important! To apply NVConfig changes, stop here and follow the steps in section "Updating NVConfig Params". In this case, the following step #3 is redundant.

3. Perform a BlueField system reboot for the upgrade to take effect.

5.1.4.7 Updating NVConfig Params from Host

1. Optional. To reset the BlueField NIC firmware configuration (aka Nvconfig params) to their factory default values, run the following from the BlueField ARM OS or from the host OS:

```
# sudo mlxconfig -d /dev/mst/<MST device> -y reset

Reset configuration for device /dev/mst/<MST device>? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

For now, please ignore tool's instruction to reboot

To learn what MST device the BlueField DPU has on your setup, run:

```
mst start
mst status
```

Example output taken on a multiple DPU host:

```
// The MST device corresponds with PCI Bus address.

MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded

MST devices:
------------
/dev/mst/mt41692_pciconf0         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:03:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
/dev/mst/mt41692_pciconf1         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:83:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
/dev/mst/mt41686_pciconf0         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:a3:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
```

The MST device IDs for the BlueField-2 and BlueField-3 DPUs in this example are `/dev/mst/mt41686_pciconf0` and `/dev/mst/mt41692_pciconf0` respectively.

2. (Optional) Enable NVMe emulation. Run:

```
sudo mlxconfig -d <MST device> -y s NVME_EMULATION_ENABLE=1
```

3. Skip this step if your BlueField DPU is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your DPU type.
If you have a VPI DPU, the default link type of the ports will be configured to IB. If you want to change the link type to Ethernet, please run the following configuration:

```
sudo mlxconfig -d <MST device> -y s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

4. Perform a BlueField system-level reset for the new settings to take effect.

5.1.4.8 Customizations During BFB Installation

Using special purpose configuration parameters in the `bf.cfg` file, the BlueField's boot options and OS can be further customized. For a full list of the supported parameters to customize your DPU system during BFB installation, refer to section "bf.cfg Parameters". In addition, the `bf.cfg` file offers further control on customization of BlueField OS installation and software configuration through scripting.

Add any of the following functions to the `bf.cfg` file for them to be called by the `install.sh` script embedded in the BFB:

- `bfb_modify_os` - called after the file system is extracted on the target partitions. It can be used to modify files or create new files on the target file system mounted under `/mnt`. So the file path should look as follows: `/mnt/<expected_path_on_target_OS>`. This can be used to run a specific tool from the target OS (remember to add `/mnt` to the path for the tool).
- `bfb_pre_install` - called before eMMC/SSD partitions format and OS filesystem is extracted
- `bfb_post_install` - called as a last step before reboot. All eMMC/SSD partitions are unmounted at this stage.

For example, the `bf.cfg` script below disables OVS bridge creation upon boot:

```
# cat /root/bf.cfg

bfb_modify_os()
{
        log ===================== bfb_modify_os =====================
        log "Disable OVS bridges creation upon boot"
        sed -i -r -e 's/(CREATE_OVS_BRIDGES=).*/\1"no"/' /mnt/etc/mellanox/mlnx-ovs.conf
}

bfb_pre_install()
{
        log ===================== bfb_pre_install =====================
}

bfb_post_install()
{
        log ===================== bfb_post_install =====================
}
```

After modifying files on the BlueField, run the command `sync` to flush file system buffers to eMMC/SSD flash memory to avoid data loss during reboot or power cycle.

5.1.4.9 Default Ports and OVS Configuration

The `/sbin/mlnx_bf_configure` script runs automatically with `ib_umad` kernel module loaded (see `/etc/modprobe.d/mlnx-bf.conf`) and performs the following configurations:

1. Ports are configured with switchdev mode and software steering.
2. RDMA device isolation in network namespace is enabled.

3. Two scalable function (SF) interfaces are created (one per port) if BlueField is configured with Embedded CPU mode (default):

```
# mlnx-sf -a show

SF Index: pci/0000:03:00.0/229408
  Parent PCI dev: 0000:03:00.0
  Representor netdev: en3f0pf0sf0
  Function HWADDR: 02:a9:49:7e:34:29
  Function trust: off
  Function roce: true
  Function eswitch: NA
  Auxiliary device: mlx5_core.sf.2
    netdev: enp3s0f0s0
    RDMA dev: mlx5_2

SF Index: pci/0000:03:00.1/294944
  Parent PCI dev: 0000:03:00.1
  Representor netdev: en3f1pf1sf0
  Function HWADDR: 02:53:8f:2c:8a:76
  Function trust: off
  Function roce: true
  Function eswitch: NA
  Auxiliary device: mlx5_core.sf.3
    netdev: enp3s0f1s0
    RDMA dev: mlx5_3
```

The parameters for these SFs are defined in configuration file `/etc/mellanox/mlnx-sf.conf`.

```
/sbin/mlnx-sf --action create --device 0000:03:00.0 --sfnum 0 --hwaddr 02:61:f6:21:32:8c
/sbin/mlnx-sf --action create --device 0000:03:00.1 --sfnum 0 --hwaddr 02:30:13:6a:2d:2c
```

To avoid repeating a MAC address in the your network, the SF MAC address is set randomly upon BFB installation. You may choose to configure a different MAC address that better suit your network needs.

4. Two OVS bridges are created:

```
# ovs-vsctl show
f08652a8-92bf-4000-ba0b-7996c772aff6
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port p1
            Interface p1
        Port en3f1pf1sf0
            Interface en3f1pf1sf0
        Port pf1hpf
            Interface pf1hpf
    Bridge ovsbr1
        Port p0
            Interface p0
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
    ovs_version: "2.14.1"
```

The parameters for these bridges are defined in configuration file `/etc/mellanox/mlnx-ovs.conf`:

```
CREATE_OVS_BRIDGES="yes"
OVS_BRIDGE1="ovsbr1"
OVS_BRIDGE1_PORTS="p0 pf0hpf en3f0pf0sf0"
OVS_BRIDGE2="ovsbr2"
OVS_BRIDGE2_PORTS="p1 pf1hpf en3f1pf1sf0"
OVS_HW_OFFLOAD="yes"
OVS_START_TIMEOUT=30
```

If failures occur in `/sbin/mlnx_bf_configure` or configuration changes happen (e.g. switching to separated host mode) OVS bridges are not created even if `CREATE_OVS_BRIDGES="yes"` .

5. OVS HW offload is configured.

5.1.4.10 Customization of BFB Installation Using bf.cfg

The BFB installation process as well as the content and configuration of the target OS can be customized during BFB installation process using the `bf.cfg` file. The `bf.cfg` file is passed to the DPU via RShim or using PXE configuration and is sourced by BFB's installation script at the beginning of the BFB installation process.

Information is available under "bf.cfg Parameters".

A number of helper functions are available in the BFB's `install.sh` script to enable customization.

- `bfb_modify_os` - the shell function is called after the file system is extracted on the target partitions. It can be used to modify files or create new files on the target file system mounted under `/mnt` . So the file path should look something like the following: `/mnt/<expected_path_on_target_OS>` . This can be used to run a specific tool from the target OS (remember to add `/mnt` to the path for the tool).
- `bfb_pre_install` - the shell function is called before the partitions format and OS filesystem is extracted.
- `bfb_post_install` - the shell function is called as a last step before reboot. All partitions are unmounted at this stage.

The BFB installation process includes the following tasks:

1. Installing target OS if `UPDATE_DPU_OS="yes"` (default)
    a. Creating and formatting partitions on the SSD (default) or EMMC drive.
    b. Extracting target OS file system from the tarball file coming with the BFB.
    c. Configuring target OS depending on the underlying hardware and provided configuration.
    d. Building initramfs for the target OS to make sure all the requirements for boot drivers are included.
2. Updating ATF and UEFI if `UPDATE_ATF_UEFI="yes"` (default).

    This is relevant for PXE installation only as ATF and UEFI are updated automatically via RShim.
3. Updating BMC components:

    Requires BMC username and password to be provided.

a. Bringing up VLAN 4040 network interface on top of `oob_net0`. VLAN 4040 is configured with static IP `192.168.240.2/29`. The timeout for bringing up the connection with the DPU's BMC VLAN 4040 interface (192.168.240.1) is set to `BMC_IP_TIMEOUT` (default is 600 seconds).
b. Updating BMC firmware if a different version is available and `UPDATE_BMC_FW="yes"` (default). The timeout for the BMC firmware update task is `BMC_TASK_TIMEOUT` (default is 1800 seconds).
c. Updating CEC firmware if a different version is available and `UPDATE_CEC_FW="yes"` (default).
d. Updating the DPU golden image if a different version is available and `UPDATE_DPU_GOLDEN_IMAGE="yes"` (default).
e. Updating the NIC firmware golden image if a different version is available and `UPDATE_NIC_FW_GOLDEN_IMAGE="yes"` (default).
f. Rebooting BMC if its firmware was updated and `BMC_REBOOT="yes"` (disabled by default).

> BMC reboot is required to apply the new BMC firmware version, but BMC reboot resets the BMC console which is used to monitor the BFB installation process. This is why BMC reboot is disabled by default and should be done after the BFB installation process if using the BMC console.

4. NIC firmware update if `WITH_NIC_FW_UPDATE="yes"` (default).
5. Reboot.

A complete installation log becomes available on the target file system after the installation process is finished (e.g., `/root/Ubuntu.installation.log`).

5.1.4.11 bf.cfg Parameters

The following is a comprehensive list of the supported parameters to customize the `bf.cfg` file for BFB installation:

```
###############################################################################
# Configuration which can also be set in
#   UEFI->Device Manager->System Configuration
###############################################################################
# Enable SMMU in ACPI.
#SYS_ENABLE_SMMU = TRUE

# Enable I2C0 in ACPI.
#SYS_ENABLE_I2C0 = FALSE

# Disable SPMI in ACPI.
#SYS_DISABLE_SPMI = FALSE

# Enable the second eMMC card which is only available on the BlueField Reference Platform.
#SYS_ENABLE_2ND_EMMC = FALSE

# Enable eMMC boot partition protection.
#SYS_BOOT_PROTECT = FALSE

# Enable SPCR table in ACPI.
#SYS_ENABLE_SPCR = FALSE

# Disable PCIe in ACPI.
#SYS_DISABLE_PCIE = FALSE

# Enable OP-TEE in ACPI.
#SYS_ENABLE_OPTEE = FALSE

###############################################################################
# Boot Order configuration
# Each entry BOOT<N> could have the following format:
# PXE:
#   BOOT<N> = NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>-<IPV4 | IPV6>
# PXE over VLAN (vlan-id in decimal):
#   BOOT<N> = NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>[.<vlan-id>]-<IPV4 | IPV6>
# UEFI Shell:
#   BOOT<N> = UEFI_SHELL
# DISK: boot entries created during OS installation.
#   BOOT<N> = DISK
###############################################################################
# This example configures PXE boot over the 2nd ConnectX port.
# If fails, it continues to boot from disk with boot entries created during OS
# installation.
#BOOT0 = NET-NIC_P1-IPV4
#BOOT1 = DISK

# UPDATE_ATF_UEFI - Updated ATF/UEFI (Default: yes)
# Relevant for PXE installation only as while using RSHIM interface ATF/UEFI
# will always be updated using capsule method
UPDATE_ATF_UEFI="yes"

# UPDATE_DPU_OS - Update/Install DPU Operating System (Default: yes)
UPDATE_DPU_OS="yes"

# grub_admin_PASSWORD - Hashed password to be set for the "admin" user to enter Grub menu
# Relevant for Ubuntu BFB only. (Default: is not set)
# E.g.:
grub_admin_PASSWORD='grub.pbkdf2.sha512.10000.5EB1FF92FDD89BDAF3395174282C77430656A6DBEC1F9289D5F5DAD17811AD0E2196D
0E49B49EF31C21972669D180713E265BB2D1D4452B2EA9C7413C3471C53.F533423479EE7465785CC2C79B637BDF77004B5CC16C1DDE806BCEA
50BF411DE04DFCCE42279E2E1F605459F1ABA3A0928CE9271F2C84E7FE7BF575DC22935B1'
grub_admin_PASSWORD='grub.pbkdf2.sha512.10000.<hashed password>'

# ubuntu_PASSWORD - Hashed password to be set for "ubuntu" user during BFB installation process.
# Relevant for Ubuntu BFB only. (Default: is not set)
ubuntu_PASSWORD=<hashed password>

###############################################################################
# BMC Component Update
###############################################################################
# BMC_USER - User name to be used to access BMC (Default: root)
BMC_USER="root"

# BMC_PASSWORD - Password used by the BMC user to access BMC (Default: None)
BMC_PASSWORD=""

# BMC_IP_TIMEOUT - Maximum time in seconds to wait for the connection to the
# BMC to be established (Default: 600)
BMC_IP_TIMEOUT=600

# BMC_TASK_TIMEOUT - Maximum time in seconds to wait for BMC task (BMC/CEC
# Firmware update) to complete (Default: 1800)
BMC_TASK_TIMEOUT=1800

# UPDATE_BMC_FW - Update BMC firmware (Default: yes)
UPDATE_BMC_FW="yes"

# BMC_REBOOT - Reboot BMC after BMC firmware update to apply the new version
# (Default: no). Note that the BMC reboot will reset the BMC console.
BMC_REBOOT="no"

# UPDATE_CEC_FW - Update CEC firmware (Default: yes)
UPDATE_CEC_FW="yes"

# UPDATE_DPU_GOLDEN_IMAGE - Update DPU Golden Image (Default: yes)
UPDATE_DPU_GOLDEN_IMAGE="yes"

# UPDATE_NIC_FW_GOLDEN_IMAGE- Update NIC firmware Golden Image (Default: yes)
UPDATE_NIC_FW_GOLDEN_IMAGE="yes"

# pre_bmc_components_update - Shell function called by BFB's install.sh before
# updating BMC components (no communication to the BMC is established at this
# point)

# post_bmc_components_update - Shell function called by BFB's install.sh after
# updating BMC components

###############################################################################
# NIC Firmware update
###############################################################################
# WITH_NIC_FW_UPDATE - Update NIC Firmware (Default: yes)
WITH_NIC_FW_UPDATE="yes"

###############################################################################
# Other misc configuration
###############################################################################

# MAC address of the rshim network interface (tmfifo_net0).
#NET_RSHIM_MAC = 00:1a:ca:ff:ff:01

# DHCP class identifier for PXE (arbitrary string up to 32 characters)
#PXE_DHCP_CLASS_ID = NVIDIA/BF/PXE

# Create dual boot partition scheme (Ubuntu only)
# DUAL_BOOT=yes

# Upgrade NIC firmware
# WITH_NIC_FW_UPDATE=yes

# Target storage device for the DPU OS (Default SSD: /dev/nvme0n1)
device=/dev/nvme0n1

# bfb_modify_os - SHELL function called after the file system is extracted on the target partitions.
# It can be used to modify files or create new files on the target file system mounted under
# /mnt. So the file path should look as follows: /mnt/<expected_path_on_target_OS>. This
# can be used to run a specific tool from the target OS (remember to add /mnt to the path for
# the tool).

# bfb_pre_install - SHELL function called before partitions format
# and OS filesystem is extracted

# bfb_post_install - SHELL function called as a last step before reboot.
# All partitions are unmounted at this stage.
```

5.1.4.12 Default Network Interface Configuration

Network interfaces are configured using the `netplan` utility:

```
# cat /etc/netplan/50-cloud-init.yaml
# This file is generated from information provided by the datasource.  Changes
# to it will not persist across an instance reboot.  To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    ethernets:
        tmfifo_net0:
            addresses:
            - 192.168.100.2/30
            dhcp4: false
            nameservers:
                addresses:
                - 192.168.100.1
            routes:
            -   metric: 1025
                to: 0.0.0.0/0
                via: 192.168.100.1
        oob_net0:
            dhcp4: true
    renderer: NetworkManager
    version: 2

# cat /etc/netplan/60-mlnx.yaml
network:
  ethernets:
    enp3s0f0s0:
      dhcp4: 'true'
    enp3s0f1s0:
      dhcp4: 'true'
  renderer: networkd
  version: 2
```

BlueField DPUs also have a local IPv6 (LLv6) derived from the MAC address via the STD stack mechanism. For a default MAC, 00:1A:CA:FF:FF:01, the LLv6 address would be fe80::21a:caff:feff:ff01.

For multi-device support, the LLv6 address works with SSH for any number of DPUs in the same host by including the interface name in the SSH command:

```
host]# systemctl restart rshim
// wait 10 seconds
host]# ssh -6 ubuntu@fe80::21a:caff:feff:ff01%tmfifo_net<n>
```

If `tmfifo_net<n>` on the host does not have an LLv6 address, restart the RShim driver:

```
systemctl restart rshim
```

5.1.5 Ubuntu Boot Time Optimizations

To improve the boot time, the following optimizations were made to Ubuntu OS image:

```
# cat /etc/systemd/system/systemd-networkd-wait-online.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/bin/nm-online -s -q --timeout=5

# cat /etc/systemd/system/NetworkManager-wait-online.service.d/override.conf
```

```
[Service]
ExecStart=
ExecStart=/usr/lib/systemd/systemd-networkd-wait-online --timeout=5

# cat /etc/systemd/system/networking.service.d/override.conf
[Service]
TimeoutStartSec=5
ExecStop=
ExecStop=/sbin/ifdown -a --read-environment --exclude=lo --force --ignore-errors
```

This configuration may affect network interface configuration if DHCP is used. If a network device fails to get configuration from the DHCP server, then the timeout value in the two files above must be increased.

Grub Configuration:

Setting the Grub timeout at 2 seconds with `GRUB_TIMEOUT=2` under `/etc/default/grub`. In conjunction with the `GRUB_TIMEOUT_STYLE=countdown` parameter, Grub will show the countdown of 2 seconds in the console before booting Ubuntu. Please note that, with this short timeout, the standard Grub method for entering the Grub menu (i.e., SHIFT or Esc) does not work. Function key F4 can be used to enter the Grub menu.

System Services:

`docker.service` is disabled in the default Ubuntu OS image as it dramatically affects boot time.

The `kexec` utility can be used to reduce the reboot time. Script `/usr/sbin/kexec_reboot` is included in the default Ubuntu 20.04 OS image to run corresponding `kexec` commands.

```
# kexec_reboot
```

5.1.6 DHCP Client Configuration

```
/etc/dhcp/dhclient.conf:
send vendor-class-identifier "NVIDIA/BF/DP";
interface "oob_net0" {
  send vendor-class-identifier "NVIDIA/BF/OOB";
    }
```

5.1.7 Ubuntu Dual Boot Support

BlueField DPU may be installed with support for dual boot. That is, two identical images of the BlueField OS may be installed using BFB.

The following is a proposed SSD partitioning layout for 119.24 GB SSD:

```
Device              Start       End   Sectors  Size Type
/dev/nvme0n1p1       2048    104447    102400   50M EFI System
/dev/nvme0n1p2     104448 114550086 114445639 54.6G Linux filesystem
/dev/nvme0n1p3 114550087 114652486    102400   50M EFI System
/dev/nvme0n1p4 114652487 229098125 114445639 54.6G Linux filesystem
/dev/nvme0n1p5 229098126 250069645  20971520   10G Linux filesystem
```

Where:

- `/dev/nvme0n1p1` - boot EFI partition for the first OS image
- `/dev/nvme0n1p2` - root FS partition for the first OS image
- `/dev/nvme0n1p3` - boot EFI partition for the second OS image
- `/dev/nvme0n1p4` - root FS partition for the second OS image

- `/dev/nvme0n1p5` - common partition for both OS images

For example, the following is a proposed eMMC partitioning layout for a 64GB eMMC:

```
Device              Start       End  Sectors  Size Type
/dev/mmcblk0p1       2048    104447   102400   50M EFI System
/dev/mmcblk0p2     104448  50660334 50555887 24.1G Linux filesystem
/dev/mmcblk0p3   50660335  50762734   102400   50M EFI System
/dev/mmcblk0p4   50762735 101318621 50555887 24.1G Linux filesystem
/dev/mmcblk0p5 101318622 122290141 20971520   10G Linux filesystem
```

Where:

- `/dev/mmcblk0p1` - boot EFI partition for the first OS image
- `/dev/mmcblk0p2` - root FS partition for the first OS image
- `/dev/mmcblk0p3` - boot EFI partition for the second OS image
- `/dev/mmcblk0p4` - root FS partition for the second OS image
- `/dev/mmcblk0p5` - common partition for both OS images

  The common partition can be used to store BFB files that will be used for OS image update on the non-active OS partition.

5.1.7.1 Installing Ubuntu OS Image Using Dual Boot

For software upgrade procedure, please refer to section "Upgrading Ubuntu OS Image Using Dual Boot".

Add the values below to the `bf.cfg` configuration file (see section "bf.cfg Parameters" for more information).

```
DUAL_BOOT=yes
```

If EMMC size is ≤16GB, dual boot support is disabled by default, but it can be forced by setting the following parameter in `bf.cfg`:

```
FORCE_DUAL_BOOT=yes
```

To modify the default size of the `/common` partition, add the following parameter:

```
COMMON_SIZE_SECTORS=<number-of-sectors>
```

The number of sectors is the size in bytes divided by the block size (512). For example, for 10GB, the `COMMON_SIZE_SECTORS=$((10*2**30/512))`.

After assigning size for the `/common` partition, what remains is divided equally between the two OS images.

```
# bfb-install --bfb <BFB> --config bf.cfg --rshim rshim0
```

This will result in the Ubuntu OS image to be installed twice on the BlueField DPU.

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

5.1.7.2 Upgrading Ubuntu OS Image Using Dual Boot

1. Download the new BFB to the BlueField DPU into the `/common` partition. Use `bfb_tool.py` script to install the new BFB on the inactive BlueField DPU partition:

```
/opt/mellanox/mlnx_snap/exec_files/bfb_tool.py --op fw_activate_bfb --bfb <BFB>
```

2. Reset BlueField DPU to load the new OS image:

```
/sbin/shutdown -r 0
```

   BlueField DPU will now boot into the new OS image.

Use `efibootmgr` utility to manage the boot order if necessary.

- Change the boot order with:

```
# efibootmgr -o
```

- Remove stale boot entries with:

```
# efibootmgr -b <E> -B
```

  Where `<E>` is the last character of the boot entry (i.e., `Boot000<E>`). You can find that by running:

```
# efibootmgr
BootCurrent: 0040
Timeout: 3 seconds
BootOrder: 0040,0000,0001,0002,0003
Boot0000* NET-NIC_P0-IPV4
Boot0001* NET-NIC_P0-IPV6
Boot0002* NET-NIC_P1-IPV4
Boot0003* NET-NIC_P1-IPV6
Boot0040* focal0
....2
```

Modifying the boot order with `efibootmgr -o` does not remove unused boot options. For example, changing a boot order from 0001,0002, 0003 to just 0001 does not actually remove 0002 and 0003. 0002 and 0003 need to be explicitly removed using `efibootmgr -B`.

5.2 Deploying BlueField Software Using BFB from BMC

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

This section assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following table lists an overview of the steps required to install Ubuntu BFB on your DPU:

Step	Procedure	Direct Link
1	Verify that RShim is already running on BMC	Ensure RShim is Running on BMC
2	Change the default credentials using `bf.cfg` file (optional)	Changing Default Credentials Using bf.cfg
3	Install the Ubuntu BFB image	BFB Installation
4	Verify installation completed successfully	Verify BFB is Installed
5	Upgrade the firmware on your DPU	Firmware Upgrade

It is important to learn your BlueField's `device-id` to perform some of the software installations or upgrades in this guide.

To determine the device ID of the BlueField Platform on your setup, run:

```
host# mst start
host# mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA           NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1         net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0         net-ens1f0
0

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0.1   e2:00.1   mlx5_1         net-
ens7f1np1             4

BlueField3(rev:1)       /dev/mst/mt41692_pciconf0     e2:00.0   mlx5_0         net-
ens7f0np0             4
```

The device IDs for the BlueField-2 and BlueField-3 networking platforms in this example are `/dev/mst/mt41686_pciconf0` and `/dev/mst/mt41692_pciconf0` respectively.

5.2.1 Ensure RShim is Running on BMC

Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
```

```
DEV_NAME        usb-1.0
```

This output indicates that the RShim service is ready to use. If you do not receive this output:

1. Restart RShim service:

   ```
   sudo systemctl restart rshim
   ```

2. Verify the current setting again. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   ```

   If `DEV_NAME` does not appear, then proceed to "RShim driver not loading on DPU with integrated BMC".

5.2.2 Changing Default Credentials Using bf.cfg

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

Ubuntu users are prompted to change the default password (ubuntu) for the default user (ubuntu) upon first login. Logging in will not be possible even if the login prompt appears until all services are up ("`DPU is ready`" message appears in `/dev/rshim0/misc`).

Attempting to log in before all services are up prints the following message: `Permission denied, please try again.`

Alternatively, Ubuntu users can provide a unique password that will be applied at the end of the BFB installation. This password must be defined in a `bf.cfg` configuration file. To set the password for the `ubuntu` user:

1. Create password hash. Run:

   ```
   # openssl passwd -1
   Password:
   Verifying - Password:
   $1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1
   ```

2. Add the password hash in quotes to the `bf.cfg` file:

   ```
   # vim bf.cfg
   ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'
   ```

   The `bf.cfg` file is used with the `bfb-install` script in the steps that follow.

5.2.3 Password Policy

The following table provides the password policy parameters.

Config File Path	Parameter	Value	Description
`/etc/security/pwquality.conf`	`minlen`	12	Minimum password length
`/etc/pam.d/common-password`	`remember`	3	The number of previous passwords which cannot be reused
`/etc/security/faillock.conf`	`silent`	Uncommented	Prevents printing informative messages to the user
	`deny`	10	The number of authentication attempts permitted before the user is locked out
	`unlock_time`	600	The duration, in seconds, of the lockout period

Each of these parameters is configurable in its respective config file indicated in the "Config File Path" column.

Please refer to the "Default Passwords and Policies" section for more password policy information.

5.2.4 BFB Installation

To update the software on the BlueField DPU, the DPU must be booted up without mounting the eMMC flash device. This requires an external boot flow where a BFB (which includes ATF, UEFI, Arm OS, NIC firmware, and initramfs) is pushed from an external host via USB or PCIe. On BlueField DPUs with an integrated BMC, the USB interface is internally connected to the BMC and is enabled by default. Therefore, you must verify that the RShim driver is running on the BMC. This provides the ability to push a bootstream over the USB interface to perform an external boot.

The BFB installation procedure consists of the following main stages:

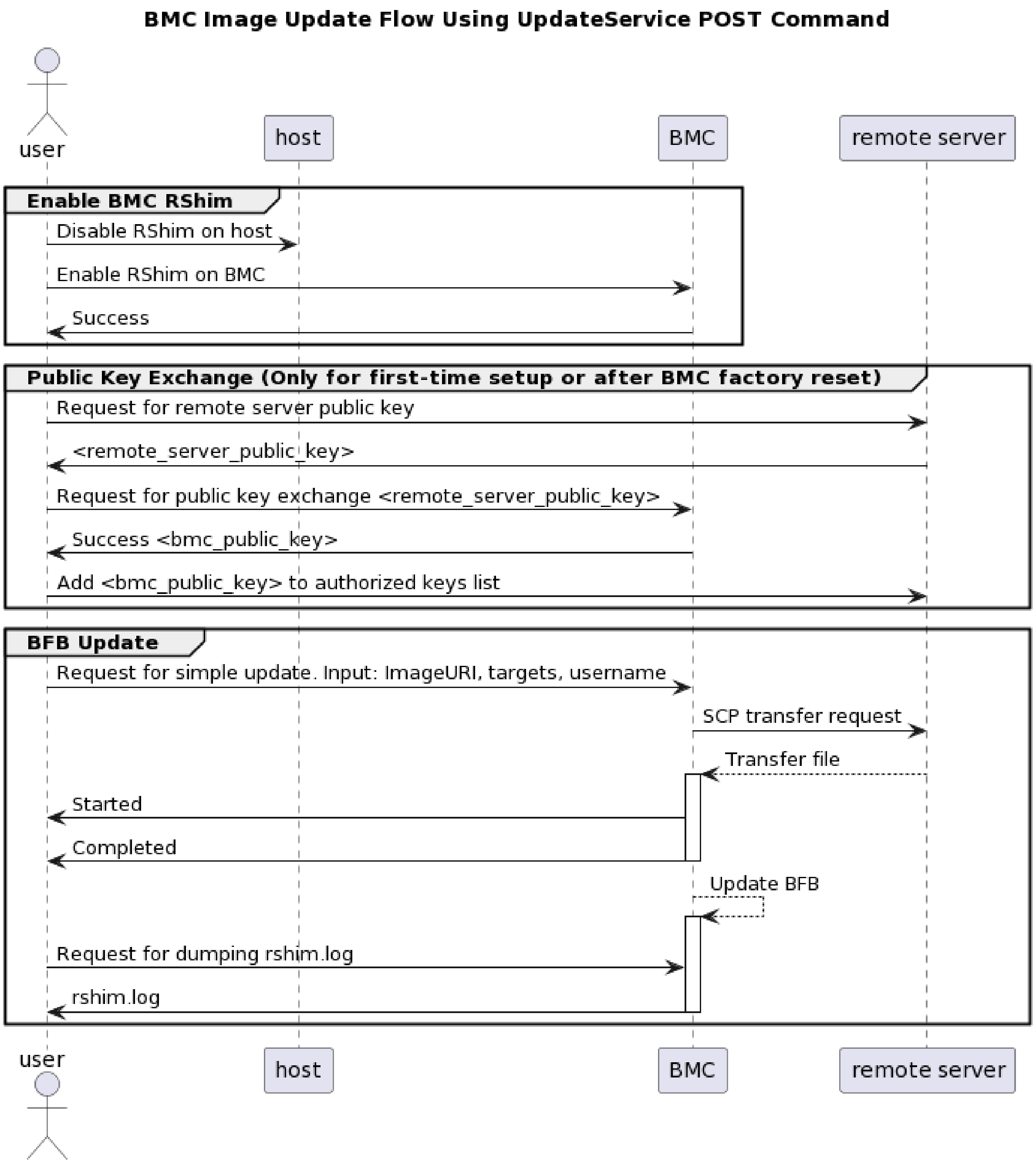
1. Enabling RShim on the BMC. See section "Enable RShim on DPU BMC" for instructions.
2. Initiating the BFB update procedure by transferring the BFB image using one of the following options:
    - Direct SCP
        i. Running an SCP command.
    - Redfish interface
        i. Confirming the identity of the host and BMC—required only during first-time setup or after BMC factory reset.
        ii. Sending a Simple-Update request.

5.2.4.1 Transferring BFB Image

Since the BFB is too large to store on the BMC flash or tmpfs, the image must be written to the RShim device. This can be done by either running SCP directly or using the Redfish interface.

5.2.4.1.1 Redfish Interface

The following is a simple sequence diagram illustrating the flow of the BFB installation process.


BMC Image Update Flow Using UpdateService POST Command
user
host
BMC
remote server
Enable BMC RShim
Disable RShim on host
Enable RShim on BMC
Success
Public Key Exchange (Only for first-time setup or after BMC factory reset)
Request for remote server public key
<remote_server_public_key>
Request for public key exchange <remote_server_public_key>
Success <bmc_public_key>
Add <bmc_public_key> to authorized keys list
BFB Update
Request for simple update. Input: ImageURI, targets, username
SCP transfer request
Transfer file
Started
Completed
Update BFB
Request for dumping rshim.log
rshim.log
user
host
BMC
remote server

The following are detailed instructions outlining each step in the diagram:

1. Confirm the identity of the remote server (i.e., host holding the BFB image) and BMC.

> Required only during first-time setup or after BMC factory reset.

a. Run the following on the remote server:

```
ssh-keyscan -t <key_type> <remote_server_ip>
```

Where:

- `key_type` - the type of key associated with the server storing the BFB file (e.g., ed25519)
- `remote_server_ip` - the IP address of the server hosting the BFB file

b. Retrieve the public key of the host holding the BFB image from the response and provide the remote server's credentials to the DPU using the following command:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d
'{"RemoteServerIP":"<remote_server_ip>", "RemoteServerKeyString":"<remote_server_public_key>"}'
https://<bmc_ip>/redfish/v1/UpdateService/Actions/Oem/NvidiaUpdateService.PublicKeyExchange
```

Where:

- `remote_server_ip` - the IP address of the server hosting the BFB file
- `remote_server_public_key` - remote server's public key from the `ssh-keyscan` response, which contains both the type and the public key with a space between the two fields (i.e., "`<type> <public_key>`").
- `bmc_ip` - BMC IP address

c. Extract the BMC public key information (i.e., "`<type> <bmc_public_key> <username>@<hostname>`") from the `PublicKeyExchange` response and append it to the `authorized_keys` file on the host holding the BFB image. This enables passwordless key-based authentication for users.

```
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "Please add the following public
       key info to ~/.ssh/authorized_keys on the
       remote server",
      "MessageArgs": [
        "<type> <bmc_public_key> root@dpu-bmc"
      ]
    },
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The request completed
       successfully.",
      "MessageArgs": [],
      "MessageId": "Base.1.15.0.Success",
      "MessageSeverity": "OK",
      "Resolution": "None"
        }
    ]
   }
```

d. If the remote server public key must be revoked, use the following command before repeating the previous step:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d
'{"RemoteServerIP":"<remote_server_ip>"}' https://<bmc_ip>/redfish/v1/UpdateService/Actions/Oem/
NvidiaUpdateService.RevokeAllRemoteServerPublicKeys
```

Where:

- `remote_server_ip` - remote server's IP address
- `bmc_ip` - BMC IP address

2. Start BFB image transfer using the following command on the remote server:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X POST -d '{"TransferProtocol":"SCP",
"ImageURI":"<image_uri>","Targets":["redfish/v1/UpdateService/FirmwareInventory/DPU_OS"],
"Username":"<username>"}' https://<bmc_ip>/redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate
```

After the BMC boots, it may take a few seconds (6-8 in NVIDIA® BlueField®-2, and 2 in BlueField-3) until the DPU BSP (`DPU_OS`) is up.

This command uses SCP for the image transfer, initiates a soft reset on the BlueField and then pushes the boot stream. For Ubuntu BFBs, the eMMC is flashed automatically once the bootstream is pushed. On success, a "running" message is received with the current task ID.

Where:

- `image_uri` - the image URI format should be `<remote_server_ip>/<path_to_bfb>`
- `username` - username on the remote server
- `bmc_ip` - BMC IP address

  Examples:
  - If RShim is disabled:

    ```
    {
      "error": {
        "@Message.ExtendedInfo": [
          {
            "@odata.type": "#Message.v1_1_1.Message",
            "Message": "The requested resource of type Target named '/dev/rshim0/boot' was not
    found.",
            "MessageArgs": [
              "Target",
              "/dev/rshim0/boot"
            ],
            "MessageId": "Base.1.15.0.ResourceNotFound",
            "MessageSeverity": "Critical",
            "Resolution": "Provide a valid resource identifier and resubmit the request."
          }
        ],
        "code": "Base.1.15.0.ResourceNotFound",
        "message": "The requested resource of type Target named '/dev/rshim0/boot' was not
    found."
    }
    ```

  - If a username or any other required field is missing:

    ```
    {
      "Username@Message.ExtendedInfo": [
        {
          "@odata.type": "#Message.v1_1_1.Message",
          "Message": "The create operation failed because the required property Username was
    missing from the request.",
          "MessageArgs": [
            "Username"
          ],
          "MessageId": "Base.1.15.0.CreateFailedMissingReqProperties",
          "MessageSeverity": "Critical",
          "Resolution": "Correct the body to include the required property with a valid value
    and resubmit the request if the operation failed."
        }
      ]
    }
    ```

  - If the request is valid and a task is created:

    ```
    {
      "@odata.id":
       "/redfish/v1/TaskService/Tasks/<task_id>",
      "@odata.type": "#Task.v1_4_3.Task",
      "Id": "<task_id>",
      "TaskState": "Running",
      "TaskStatus": "OK"
    }
    ```

3. Wait 2 seconds and run the following on the host to track image transfer progress:

```
curl -k -u root:'<password>' -X GET https://<bmc_ip>/redfish/v1/TaskService/Tasks/<task_id>
```

The transfer takes ~8 minutes for BlueField-3, and ~40 minutes for BlueField-2. During the transfer, the `PercentComplete` value remains at 0. If no errors occur, the `TaskState` is set to `Running`, and a keep-alive message is generated every 5 minutes with the content "Transfer is still in progress (X minutes elapsed). Please wait". Once the transfer is completed, the `PercentComplete` is set to 100, and the `TaskState` is updated to `Completed`.

Upon failure, a message is generated with the relevant resolution.

Where:

a. `bmc_ip` - BMC IP address
b. `task_id` - task ID

Troubleshooting:

- If host identity is not confirmed or the provided host key is wrong:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>,
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": " Unknown Host: Please provide server's public key using
PublicKeyExchange ",
      "Severity": "Critical"
    }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

In this case, revoke the remote server key (step 1.d.), and repeat steps 1.a. to 1.c.

- If the BMC identity is not confirmed:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": "Unauthorized Client: Please use the PublicKeyExchange action to
receive the system's public key and add it as an authorized key on the remote server",
      "Severity": "Critical"
      }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

In this case, verify that the BMC key has been added correctly to the `authorized_key` file on the remote server.

- If SCP fails:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Transfer of image '<file_name>' to '/dev/rshim0/boot' failed.",
      "MessageArgs": [
        "<file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferFailed",
      "Resolution": "Failed to launch SCP",
      "Severity": "Critical"
      }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

- The keep-alive message:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": " <file_name>' is being transferred to '/dev/rshim0/boot'.",
      "MessageArgs": [
        " <file_name>",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferringToComponent",
      "Resolution": "Transfer is still in progress (5 minutes elapsed): Please wait",
      "Severity": "OK"
    }
…
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Running",
  "TaskStatus": "OK"
```

- Upon completion of transfer of the BFB image to the DPU, the following is received:

```
{
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Device 'DPU' successfully updated with image '<file_name>'.",
      "MessageArgs": [
        "DPU",
        "<file_name>"
      ],
      "MessageId": "Update.1.0.UpdateSuccessful",
      "Resolution": "None",
      "Severity": "OK"
    },
…
"PercentComplete": 100,
  "StartTime": "<start_time>",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Completed",
  "TaskStatus": "OK"
```

4. When the BFB transfer is complete, dump the current RShim miscellaneous messages to check the update status.

   Refer to section "BMC Dump Operations" under "BMC and BlueField Logs" for information on dumping the `rshim.log` which contains the current RShim miscellaneous messages.

5. Verify that the new BFB is running by checking its version:

```
curl -k -u root:'<password>' -H "Content-Type: application/json" -X GET https://<bmc_ip>/redfish/v1/
UpdateService/FirmwareInventory/DPU_OS
```

5.2.4.1.2 Direct SCP

```
scp <path_to_bfb> root@<bmc_ip>:/dev/rshim0/boot
```

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

5.2.5 Verify BFB is Installed

After installation of the Ubuntu OS is complete, the following note appears in `/dev/rshim0/misc` on first boot:

```
...
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
```

"DPU is ready" indicates that all the relevant services are up and users can login the system.

After the installation of the Ubuntu 20.04 BFB, the configuration detailed in the following sections is generated.

Make sure all the services (including cloud-init) are started on BlueField and to perform a graceful shutdown before power cycling the host server.

BlueField OS image version is stored under `/etc/mlnx-release` in the BlueField:

```
# cat /etc/mlnx-release
bf-bundle-2.7.0-<version>_ubuntu-22.04_prod
```

5.2.6 Firmware Upgrade

To upgrade firmware:

1. Access the BlueField using one of the available interfaces (RShim console, BMC console, SSH via `oob_net0` or `tmfifo_net0` interfaces).
2. Upgrade the firmware on the DPU. Run:

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

   Example output:

   ```
   Device #1:
   ----------

     Device Type:      BlueField-2
     [...]
     Versions:         Current        Available
   ```

```
      FW              <Old_FW>        <New_FW>
```

Important! To apply NVConfig changes, stop here and follow the steps in section "Updating NVConfig Params". In this case, the following step #3 is redundant.

3. Perform a BlueField system reboot for the upgrade to take effect.

5.2.7 Updating NVConfig Params

1. Optional. To reset the BlueField NIC firmware configuration (aka Nvconfig params) to their factory default values, run the following from the BlueField ARM OS or from the host OS:

```
# sudo mlxconfig -d /dev/mst/<MST device> -y reset

Reset configuration for device /dev/mst/<MST device>? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

For now, please ignore tool's instruction to reboot

To learn what MST device the BlueField DPU has on your setup, run:

```
mst start
mst status
```

Example output taken on a multiple DPU host:

```
// The MST device corresponds with PCI Bus address.

MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded

MST devices:
------------
/dev/mst/mt41692_pciconf0         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:03:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
/dev/mst/mt41692_pciconf1         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:83:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
/dev/mst/mt41686_pciconf0         - PCI configuration cycles access.
                                    domain:bus:dev.fn=0000:a3:00.0 addr.reg=88 data.reg=92
cr_bar.gw_offset=-1
                                    Chip revision is: 01
```

The MST device IDs for the BlueField-2 and BlueField-3 DPUs in this example are `/dev/mst/mt41686_pciconf0` and `/dev/mst/mt41692_pciconf0` respectively.

2. (Optional) Enable NVMe emulation. Run:

```
sudo mlxconfig -d <MST device> -y s NVME_EMULATION_ENABLE=1
```

3. Skip this step if your BlueField DPU is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your DPU type.
If you have a VPI DPU, the default link type of the ports will be configured to IB. If you want to change the link type to Ethernet, please run the following configuration:

```
sudo mlxconfig -d <MST device> -y s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

4. Perform a BlueField system-level reset for the new settings to take effect.

5.3 Deploying BlueField Software Using BFB with PXE

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

PXE installation is not supported for NIC mode on NVIDIA® BlueField®-3.

5.3.1 PXE Server Preparations

1. Provide the image from BFB file. Run:

   ```
   # mlx-mkbfb -x <BFB>
   ```

   For example:

   ```
   # mlx-mkbfb -x DOCA_2.6.0_BSP_4.6.0_Ubuntu_22.04-<version>.bfb
   ```

   `mlx-mkbfb` is a Python script that can be found in BlueField release tarball under the `/bin` directory or in the BlueField Arm file system `/usr/bin/mlx-mkbfb`.
2. Copy the 2 dumped files, `dump-image-v0` and `dump-initramfs-v0` into the PXE server `tftp` path.
3. In the PXE server create a boot entry. For example:

   ```
   /var/lib/tftpboot/grub.cfg

   set default=0
   set timeout=5
   menuentry 'Bluefield_Ubuntu_22_04_From_BFB' --class red --class gnu-linux --class gnu --class os {
      linux (tftp)/ubuntu22.04/dump-image-v0 ro ip=dhcp console=hvc0 console=ttyAMA0
      initrd (tftp)/ubuntu22.04/dump-initramfs-v0
   }
   ```

   If additional parameters must be set, use the `bf.cfg` configuration file, then add the `bfks` parameter to the Linux command line in the `grub.cfg` above.

   ```
   menuentry 'Ubuntu22.04 From BFB with bf.cfg' --class red --class gnu-linux --class gnu --class os {
      linux (tftp)/ubuntu22.04/dump-image-v0 console=hvc0 console=ttyAMA0 bfnet=oob_net0:dhcp bfks=http://
   15.22.82.40/bfks
      initrd (tftp)/ubuntu22.04/dump-initramfs-v0
   }
   ```

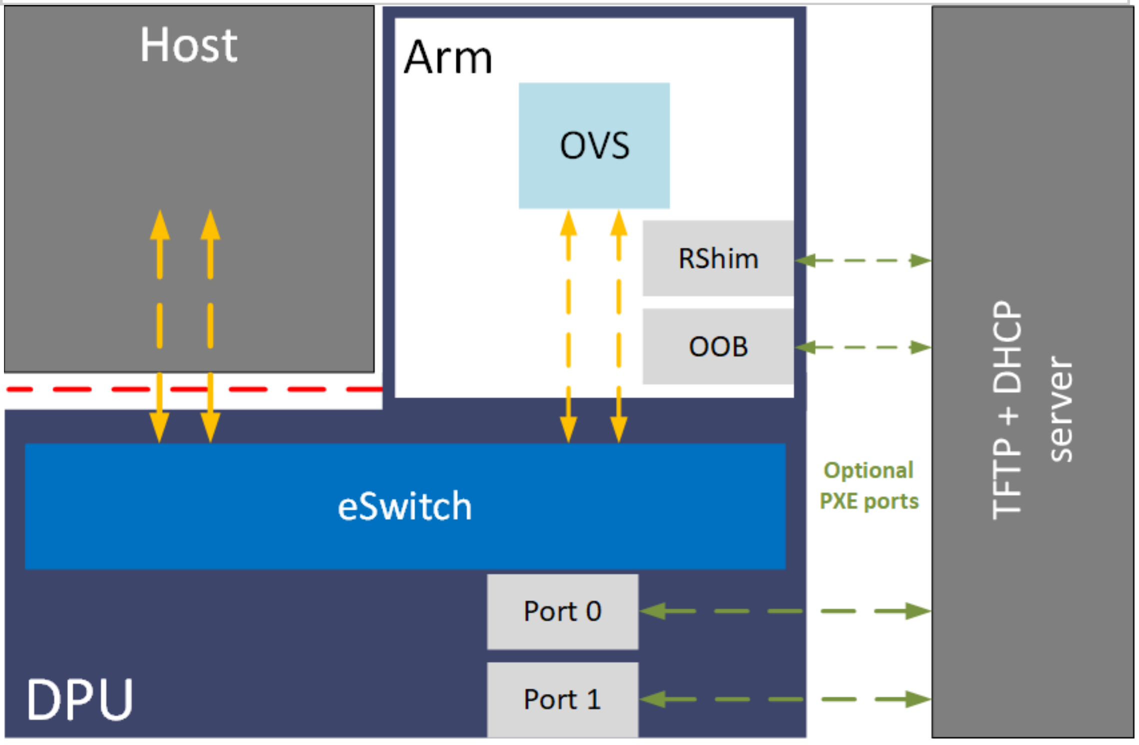
   Where `bfks` is a BASH script that will run by BFB's install.sh script at the beginning of the BFB installation process. Here is an example of bfks that creates a `/etc/bf.cfg` file:

   ```
   cat > /etc/bf.cfg << 'EOF'
   DEBUG=yes
   ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'
   ```

```
EOF
```

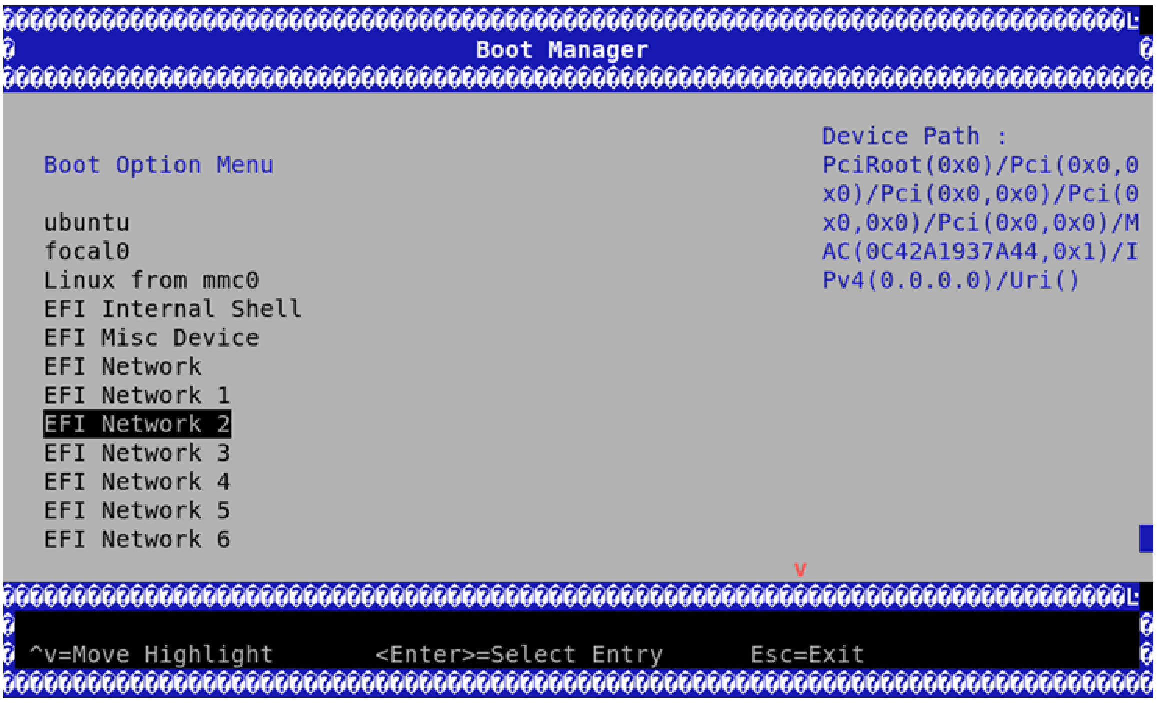
4. Define DHCP.

```
/etc/dhcp/dhcpd.conf

allow booting;
allow bootp;

subnet 192.168.100.0 netmask 255.255.255.0 {
  range 192.168.100.10 192.168.100.20;
  option broadcast-address 192.168.100.255;
  option routers 192.168.100.1;
  option domain-name-servers <ip-address-list>
  option domain-search <domain-name-list>;
  next-server 192.168.100.1;
  filename "/BOOTAA64.EFI";
}

# Specify the IP address for this client.
host tmfifo_pxe_client {
  hardware ethernet 00:1a:ca:ff:ff:01;
  fixed-address 192.168.100.2;
}
subnet 20.7.0.0 netmask 255.255.0.0 {
  range 20.7.8.10 20.7.254.254;
  next-server 20.7.6.6;
  filename "/BOOTAA64.EFI";
}
```



5.3.2 PXE Sequence

1. Connect to the BlueField console via UART or RShim console.
2. Reboot Arm.
3. Interrupt the boot process into UEFI menu.
4. Access the Boot Manager menu.
5. Select the relevant port to PXE from.



```
Boot Manager

Boot Option Menu                                    Device Path :
                                                    PciRoot(0x0)/Pci(0x0,0
ubuntu                                              x0)/Pci(0x0,0x0)/Pci(0
focal0                                              x0,0x0)/Pci(0x0,0x0)/M
Linux from mmc0                                     AC(0C42A1937A44,0x1)/I
EFI Internal Shell                                  Pv4(0.0.0.0)/Uri()
EFI Misc Device
EFI Network
EFI Network 1
EFI Network 2
EFI Network 3
EFI Network 4
EFI Network 5
EFI Network 6

^v=Move Highlight       <Enter>=Select Entry      Esc=Exit
```

5.3.3 PXE Sequence with Redfish

ISO upgrade via Redfish to set UEFI HTTPs/PXE boot by setting UEFI first boot source. To set the UEFI first boot source using Redfish:

1. Follow the instructions under section "PXE Server Preparations".
2. Check the current boot override settings by doing a GET on the ComputerSystem schema over 1GbE to the DPU's BMC. Look for the `"Boot"` property.

```
curl -k -X GET -u root:<password> https://<DPU-BMC-IP>/redfish/v1/Systems/<SystemID>/ | python3 -m
json.tool
{
...
"Boot": {
        "BootNext": "",
        "BootOrderPropertySelection": "BootOrder",
        "BootSourceOverrideEnabled": "Disabled",
        "BootSourceOverrideMode": "UEFI",
        "BootSourceOverrideTarget": "None",
        "UefiTargetBootSourceOverride": "None",
        .....
        },
  ....
  "BootSourceOverrideEnabled@Redfish.AllowableValues": [
            "Once",
            "Continuous",
            "Disabled"
        ],
    "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "UefiHttp",
            "UefiShell",
            "UefiTarget",
            "UefiBootNext"
        ],
  ....
}
```

Boot override enables overriding the first boot source, either once or continuously.

3. The sample output above shows the `BootSourceOverrideEnabled` property is `Disabled` and `BootSourceOverrideTarget` is `None`. The `BootSourceOverrideMode` property should always be set to `UEFI`. Allowable values of `BootSourceOverrideEnabled` and `BootSourceOverrideTarget` are defined in the metadata

(`BootSourceOverrideEnabled@Redfish.AllowableValues` and `BootSourceOverrideTarget@Redfish.AllowableValues` respectively).

4. If `BootSourceOverrideEnabled` is set to `Once`, after the first boot, boot override is disabled, and any related properties are reset to their former values to avoid repetition. If it is set to `Continuous`, then on every reboot the DPU keeps performing boot override (HTTPBoot).
5. To perform boot override, perform a PATCH to pending settings URI over 1GbE to the DPU's BMC.

```
curl -k -X PATCH -d '{"Boot": {"BootSourceOverrideEnabled":"Once", "BootSourceOverrideMode":"UEFI",
"BootSourceOverrideTarget": "UefiHttp", "HttpBootUri":"http://<HTTP-Server-Ip>/Image.iso"}}' -u
root:<password> https://<DPU-BMC-IP>/redfish/v1/Systems/<SystemID>/Settings | python3 -m json.tool
```

For example:

```
curl -k -X GET -u root:<password> https://<DPU-BMC-IP>/redfish/v1/Systems/<SystemID>/ | python3 -m
json.tool
{
...
"Boot": {
         "BootNext": "",
         "BootOrderPropertySelection": "BootOrder",
         "BootSourceOverrideEnabled": "Once",
         "BootSourceOverrideMode": "UEFI",
         "BootSourceOverrideTarget": "UefiHttp",
         "UefiTargetBootSourceOverride": "None",
         .....
         },
    .....
}
```

6. After performing the above PATCH successfully, reboot the DPU using Redfish Manager schema over 1GbE to the DPU's BMC:

```
curl -k -u root:<password> -H "Content-Type: application/json" -X POST https://<DPU-BMC-IP>/redfish/v1/
Systems/Bluefield/Actions/ComputerSystem.Reset -d '{"ResetType" : "GracefulRestart"}'
```

7. Once UEFI has completed, check whether the settings are applied by performing a GET on ComputerSystem schema over 1GbE OOB to the DPU BMC.

The `HttpBootUri` property is parsed by the Redfish server and the URI is presented to the DPU as part of DHCP lease when the DPU performs the HTTP boot.

5.4 Deploying NVIDIA Converged Accelerator

It is recommended to upgrade your BlueField product to the latest software and firmware versions available to benefit from new features and latest bug fixes.

This section assumes that you have installed the BlueField OS BFB on your NVIDIA® Converged Accelerator using any of the following guides:

- Deploying DPU OS Using BFB from Host
- Deploying BlueField Software Using BFB from BMC
- Deploying BlueField Software Using BFB with PXE

NVIDIA® CUDA® (GPU driver) must be installed in order to use the GPU. For information on how to install CUDA on your Converged Accelerator, refer to NVIDIA CUDA Installation Guide for Linux.

5.4.1 Configuring Operation Mode

After installing the BFB, you may now select the mode you want your NVIDIA Converged Accelerator to operate in.

- Standard (default) – the NVIDIA® BlueField® DPU and the GPU operate separately (GPU is owned by the host)
- BlueField-X – the GPU is exposed to the DPU and is no longer visible on the host (GPU is owned by the DPU)

It is important to know your device name (e.g., `mt41686_pciconf0`).

MST tool is necessary for this purpose which is installed by default on the DPU.

Run:

```
mst status -v
```

Example output:

```
MST modules:
------------
    MST PCI module is not loaded
    MST PCI configuration module loaded
PCI devices:
------------
DEVICE_TYPE             MST                           PCI       RDMA            NET
NUMA
BlueField2(rev:1)       /dev/mst/mt41686_pciconf0.1   3b:00.1   mlx5_1          net-ens1f1
0

BlueField2(rev:1)       /dev/mst/mt41686_pciconf0     3b:00.0   mlx5_0          net-ens1f0
0
```

5.4.1.1 BlueField-X Mode

1. Run the following command from the host:

```
mlxconfig -d /dev/mst/<device-name> s PCI_DOWNSTREAM_PORT_OWNER[4]=0xF
```

2. Perform a BlueField system-level reset for the `mlxconfig` settings to take effect.

5.4.1.2 Standard Mode

To return the DPU from BlueField-X mode to Standard mode:

1. Run the following command from the host:

```
mlxconfig -d /dev/mst/<device-name> s PCI_DOWNSTREAM_PORT_OWNER[4]=0x0
```

2. Perform a BlueField system-level reset for the `mlxconfig` settings to take effect.

5.4.2 Verifying Configured Operational Mode

Use the following command from the host or BlueField:

```
$ sudo mlxconfig -d /dev/mst/<device-name> q PCI_DOWNSTREAM_PORT_OWNER[4]
```

- Example of Standard mode output:

```
Device #1:
----------

[...]

Configurations:                              Next Boot
         PCI_DOWNSTREAM_PORT_OWNER[4]        DEVICE_DEFAULT(0)
```

- Example of BlueField-X mode output:

```
Device #1:
----------

[...]

Configurations:                              Next Boot
         PCI_DOWNSTREAM_PORT_OWNER[4]        EMBEDDED_CPU(15)
```

5.4.3 Verifying GPU Ownership

The following are example outputs for when the DPU is configured to BlueField-X mode.

The GPU is no longer visible from the host:

```
root@host:~# lspci | grep -i nv
None
```

The GPU is now visible from the DPU:

```
ubuntu@dpu:~$ lspci | grep -i nv
06:00.0 3D controller: NVIDIA Corporation GA20B8 (rev a1)
```

5.4.4 GPU Firmware

5.4.4.1 Get GPU Firmware

```
smbpbi: (See SMBPBI spec)

root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x00 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x00 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x39 0x32 0x2e 0x30
root@dpu:~#
root@dpu:~#
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x01 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x01 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x2e 0x36 0x42
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x02 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x02 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x2e 0x30 0x30 0x2e
root@dpu:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x03 0x80 s
root@dpu:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x03 0x5f
root@dpu:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x31 0x00 0x00
root@dpu:~#

39 32 2e 30 30 2e 36 42 2e 30 30 2e 30 31 00 00   92.00.6B.00.01
```

5.4.4.2 Updating GPU Firmware

```
root@dpu:~# scp root@10.23.201.227:/<path-to-fw-bin>/1004_0230_891__92006B0001-dbg-ota.bin /tmp/gpu_images/
root@10.23.201.227's password:
1004_0230_891__92006B0001-dbg-ota.bin                                         100%  384KB 384.4KB/s   00:01

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@dpu:~# cat /tmp/gpu_images/progress.txt
TaskState=Frimware update succeeded.
TaskStatus=OK
TaskProgress=100
```

5.5 Installing Repo Package on Host Side

This section assumes that a BlueField DPU has already been installed in a server according to the instructions detailed in the DPU's hardware user guide.

The following procedure instructs users on upgrading DOCA local repo package for host.

5.5.1 Removing Previously Installed DOCA Runtime Packages

If an older DOCA software version is installed on your host, make sure to uninstall it before proceeding with the installation of the new version:

Ubuntu	`host# for f in $( dpkg --list | grep doca | awk '{print $2}' ); do echo $f ; apt remove --purge $f -y ; done`
	`host# sudo apt-get autoremove`

CentOS/RHEL	`host# for f in $(rpm -qa |grep -i doca ) ; do yum -y remove $f; done`
	`host# yum autoremove`
	`host# yum makecache`

5.5.2 Downloading DOCA Runtime Packages

The following table provides links to DOCA Runtime packages depending on the OS running on your host.

OS	Arch	Link
Alinux 3.2	x86	doca-host-2.7.0-204000_24.04_alinux32.x86_64.rpm
Anolis	aarch64	doca-host-2.7.0-204000_24.04_anolis86.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_anolis86.x86_64.rpm

OS	Arch	Link
BCLinux 21.10 SP2	aarch64	doca-host-2.7.0-204000_24.04_bclinux2110sp2.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_bclinux2110sp2.x86_64.rpm
CTyunOS 2.0	aarch64	doca-host-2.7.0-204000_24.04_ctyunos20.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_ctyunos20.x86_64.rpm
CTyunOS 23.01	aarch64	doca-host-2.7.0-204000_24.04_ctyunos2301.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_ctyunos2301.x86_64.rpm
Debian 10.13	aarch64	doca-host_2.7.0-204000-24.04-debian1013_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-debian1013_amd64.deb
Debian 10.8	aarch64	doca-host_2.7.0-204000-24.04-debian108_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-debian108_amd64.deb
Debian 10.9	x86	doca-host_2.7.0-204000-24.04-debian109_amd64.deb
Debian 11.3	aarch64	doca-host_2.7.0-204000-24.04-debian113_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-debian113_amd64.deb
Debian 12.1	aarch64	doca-host_2.7.0-204000-24.04-debian121_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-debian121_amd64.deb
EulerOS 20 SP11	aarch64	doca-host-2.7.0-204000_24.04_euleros20sp11.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_euleros20sp11.x86_64.rpm
EulerOS 20 SP12	aarch64	doca-host-2.7.0-204000_24.04_euleros20sp12.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_euleros20sp12.x86_64.rpm

OS	Arch	Link
Fedora32	x86	doca-host-2.7.0-204000_24.04_fc32.x86_64.rpm
Kylin 1.0 SP2	aarch64	doca-host-2.7.0-204000_24.04_kylin10sp2.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_kylin10sp2.x86_64.rpm
Kylin 1.0 SP3	aarch64	doca-host-2.7.0-204000_24.04_kylin10sp3.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_kylin10sp3.x86_64.rpm
Mariner 2.0	x86	doca-host-2.7.0-204000_24.04_mariner20.x86_64.rpm
Oracle Linux 7.9	x86	doca-host-2.7.0-204000_24.04_ol79.x86_64.rpm
Oracle Linux 8.4	x86	doca-host-2.7.0-204000_24.04_ol84.x86_64.rpm
Oracle Linux 8.6	x86	doca-host-2.7.0-204000_24.04_ol86.x86_64.rpm
Oracle Linux 8.7	x86	doca-host-2.7.0-204000_24.04_ol87.x86_64.rpm
Oracle Linux 8.8	x86	doca-host-2.7.0-204000_24.04_ol88.x86_64.rpm
Oracle Linux 9.0	x86	doca-host-2.7.0-204000_24.04_ol90.x86_64.rpm
Oracle Linux 9.1	x86	doca-host-2.7.0-204000_24.04_ol91.x86_64.rpm
Oracle Linux 9.2	x86	doca-host-2.7.0-204000_24.04_ol92.x86_64.rpm
openEuler 20.03 SP3	aarch64	doca-host-2.7.0-204000_24.04_openeuler2003sp3.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_openeuler2003sp3.x86_64.rpm
openEuler 22.03	aarch64	doca-host-2.7.0-204000_24.04_openeuler2203.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_openeuler2203.x86_64.rpm
RHEL/CentOS 8.0	aarch64	doca-host-2.7.0-204000_24.04_rhel80.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel80.x86_64.rpm
RHEL/CentOS 8.1	aarch64	doca-host-2.7.0-204000_24.04_rhel81.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel81.x86_64.rpm

OS	Arch	Link
RHEL/CentOS 8.2	aarch64	doca-host-2.7.0-204000_24.04_rhel82.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel82.x86_64.rpm
RHEL/CentOS 8.3	aarch64	doca-host-2.7.0-204000_24.04_rhel83.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel83.x86_64.rpm
RHEL/CentOS 8.4	aarch64	doca-host-2.7.0-204000_24.04_rhel84.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel84.x86_64.rpm
RHEL/CentOS 8.5	aarch64	doca-host-2.7.0-204000_24.04_rhel85.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel85.x86_64.rpm
RHEL/Rocky 8.6	aarch64	doca-host-2.7.0-204000_24.04_rhel86.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel86.x86_64.rpm
RHEL/Rocky 8.7	aarch64	doca-host-2.7.0-204000_24.04_rhel87.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel87.x86_64.rpm
RHEL/Rocky 8.8	aarch64	doca-host-2.7.0-204000_24.04_rhel88.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel88.x86_64.rpm
RHEL/Rocky 8.9	aarch64	doca-host-2.7.0-204000_24.04_rhel89.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel89.x86_64.rpm
RHEL/Rocky 8.10	aarch64	doca-host-2.7.0-204000_24.04_rhel810.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel810.x86_64.rpm
RHEL/Rocky 9.0	aarch64	doca-host-2.7.0-204000_24.04_rhel90.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel90.x86_64.rpm
RHEL/Rocky 9.1	aarch64	doca-host-2.7.0-204000_24.04_rhel91.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel91.x86_64.rpm

OS	Arch	Link
RHEL/Rocky 9.2	aarch64	doca-host-2.7.0-204000_24.04_rhel92.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel92.x86_64.rpm
RHEL/Rocky 9.3	aarch64	doca-host-2.7.0-204000_24.04_rhel93.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel93.x86_64.rpm
RHEL/Rocky 9.4	aarch64	doca-host-2.7.0-204000_24.04_rhel94.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_rhel94.x86_64.rpm
SLES 15 SP2	aarch64	doca-host-2.7.0-204000_24.04_sles15sp2.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_sles15sp2.x86_64.rpm
SLES 15 SP3	aarch64	doca-host-2.7.0-204000_24.04_sles15sp3.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_sles15sp3.x86_64.rpm
SLES 15 SP4	aarch64	doca-host-2.7.0-204000_24.04_sles15sp4.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_sles15sp4.x86_64.rpm
SLES 15 SP5	aarch64	doca-host-2.7.0-204000_24.04_sles15sp5.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_sles15sp5.x86_64.rpm
SLES 15 SP6	x86	doca-host-2.7.0-204000_24.04_sles15sp6.x86_64.rpm
TencentOS 3.3	aarch64	doca-host-2.7.0-204000_24.04_tencentos33.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_tencentos33.x86_64.rpm
Ubuntu 20.04	aarch64	doca-host_2.7.0-204000-24.04-ubuntu2004_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-ubuntu2004_amd64.deb
Ubuntu 22.04	aarch64	doca-host_2.7.0-204000-24.04-ubuntu2204_arm64.deb

OS	Arch	Link
	x86	doca-host_2.7.0-204000-24.04-ubuntu2204_amd64.deb
Ubuntu 24.04	aarch64	doca-host_2.7.0-204000-24.04-ubuntu2404_arm64.deb
	x86	doca-host_2.7.0-204000-24.04-ubuntu2404_amd64.deb
UOS20.1060	aarch64	doca-host-2.7.0-204000_24.04_uos201060.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_uos201060.x86_64.rpm
UOS20.1060A	aarch64	doca-host-2.7.0-204000_24.04_uos201060a.aarch64.rpm
	x86	doca-host-2.7.0-204000_24.04_uos201060a.x86_64.rpm
XenServer 8.2	x86	doca-host-2.7.0-204000_24.04_xenserver82.x86_64.rpm

5.5.3 Installing Local Repo Package for Host Dependencies

1. Install DOCA local repo package for host:

OS	Procedure

Ubuntu	a. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the host.
	b. Unpack the deb repo. Run:
	`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`
	c. Perform apt update. Run:
	`host# sudo apt-get update`
	d. Run `apt install` for DOCA runtime, tools, and SDK:
	`host# sudo apt install -y doca-runtime doca-sdk`

OS	Procedure

CentOS	a. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the x86 host.
	b. Install the following software dependencies. Run:
	`host# sudo yum install -y epel-release`
	c. For CentOS 8.2 only, also run:
	`host# yum config-manager --set-enabled PowerTools`
	d. Unpack the RPM repo. Run:
	`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	e. Run `yum install` for DOCA runtime, tools, and SDK.
	`host# sudo yum install -y doca-runtime doca-sdk`

RHEL	a. Open a RedHat account.
	i. Log into RedHat website via the developers tab.
	ii. Create a developer user.
	b. Run:
	`host# subscription-manager register --username=<username> --password=PASSWORD`
	To extract pool ID:
	`host# subscription-manager list --available --all`
	`...`
	`Subscription Name:   Red Hat Developer Subscription for Individuals`
	`Provides:            Red Hat Developer Tools (for RHEL Server for ARM)`
	`                     ...`
	`                     Red Hat CodeReady Linux Builder for x86_64`
	`...`
	`Pool ID:             <pool-id>`
	`...`
	And use the pool ID for the `Subscription Name` and `Provides` that include `Red Hat CodeReady Linux Builder for x86_64`.
	c. Run:
	`host# subscription-manager attach --pool=<pool-id>`
	`host# subscription-manager repos --enable codeready-builder-for-rhel-8-x86_64-rpms`
	`host# yum makecache`
	d. Install the DOCA local repo package for host. Run:
	`host# rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`
	`host# sudo yum install -y doca-runtime doca-sdk`
	e. Sign out from your RHEL account. Run:
	`host# subscription-manager remove --all`
	`host# subscription-manager unregister`

2. Assign a dynamic IP to `tmfifo_net0` interface (RShim host interface).

```
host# ifconfig tmfifo_net0 192.168.100.1 netmask 255.255.255.252 up
```

3. Verify that RShim is active.

```
host# sudo systemctl status rshim
```

This command is expected to display "`active (running)`". If RShim service does not launch automatically, run:

```
host# sudo systemctl enable rshim
host# sudo systemctl start rshim
```

5.6 Installing Popular Linux Distributions on BlueField

5.6.1 Building Your Own BFB Installation Image

Users wishing to build their own customized NVIDIA® BlueField® OS image can use the BFB build environment. See this GitHub webpage for more information.

For any customized BlueField OS image to boot on the UEFI secure-boot-enabled DPU (default DPU secure boot setting), the OS must be either signed with an existing key in the UEFI DB (e.g., the Microsoft key), or UEFI secure boot must be disabled. See "Secure Boot" and its subpages for more details.

5.6.2 Installing Linux Distributions

Contact NVIDIA Enterprise Support for information on the installation of Linux distributions other than Ubuntu.

5.6.3 BlueField Linux Drivers

The following table lists the BlueField drivers which are part of the Official Ubuntu Linux distribution for BlueField. Some of the drivers are not in the upstream Linux kernel yet.

Driver	Description	BlueField-2	BlueField-3
`bluefield-edac`	BlueField-specific EDAC driver	✓	✗
`dw_mmc_bluefield`	BlueField DW Multimedia Card driver	✓	✓
`sdhci-of-dwcmshc`	SDHCI platform driver for Synopsys DWC MSHC	✓	✓
`gpio-mlxbf2`	GPIO driver	✓	✗
`gpio-mlxbf3`	GPIO driver	✗	✓
`i2c-mlx`	I2C bus driver (`i2c-mlxbf.c` upstream)	✓	✗

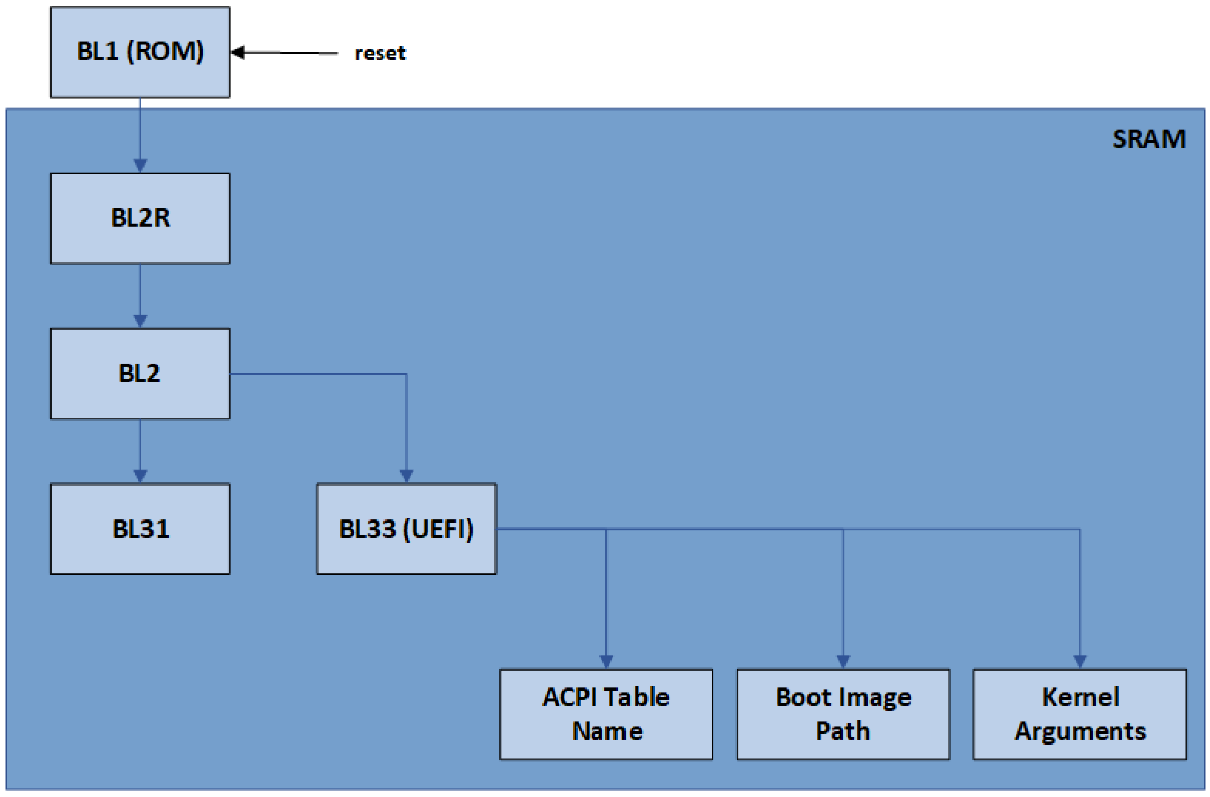
Driver	Description	BlueField-2	BlueField-3
`ipmb-dev-int`	Driver needed to receive IPMB messages from a BMC and send a response back. This driver works with the I2C driver and a user-space program such as OpenIPMI.	✓	✗
`ipmb-host`	Driver needed on the DPU to send IPMB messages to the BMC on the IPMB bus. This driver works with the I2C driver. It only loads successfully if it executes a successful handshake with the BMC.	✓	✗
`mlxbf-gige`	Gigabit Ethernet driver	✓	✓
`mlxbf-livefish`	BlueField HCA firmware burning driver. This driver supports burning firmware for the embedded HCA in the BlueField SoC.	✓	✗
`mlxbf-pka`	BlueField PKA kernel module	✓	✓
`mlxbf-pmc`	Performance monitoring counters. The driver provides access to available performance modules through the `sysfs` interface. The performance modules in BlueField are present in several hardware blocks and each block has a certain set of supported events.	✓	✗
`mlxbf-ptm`	Kernel driver that provides a debufgs interface for the system software to monitor the BlueField device's power and thermal management parameters.	✗	✓
`mlxbf-tmfifo`	TMFIFO driver for BlueField SoC	✓	✓
`mlx-bootctl`	Boot control driver. This driver provides a `sysfs` interface for systems management software to manage reset time actions.	✓	✓
`mlx-trio`	TRIO driver for BlueField SoC	✓	✗
`pwr-mlxbf`	Supports reset or low-power mode handling for BlueField.	✓	✓
`pinctrl-mlxbf`	Allows multiplexing individual GPIOs to switch from the default hardware mode to software-controlled mode.	✗	✓
`mlxbf-pmc`	Mellanox PMC driver	✗	✓

5.7 Updating DPU Software Packages Using Standard Linux Tools

Unable to render include or excerpt-include. Could not retrieve page.

5.7.1 Upgrading Boot Software

This section describes how to use the BlueField alternate boot partition support feature to safely upgrade the boot software. We give the requirements that motivate the feature and explain the software interfaces that are used to configure it.



5.7.1.1 BFB File Overview

The default BlueField bootstream (BFB) shown above (located at `/lib/firmware/mellanox/boot/default.bfb`) is assumed to be loaded from the eMMC. In it, there is a hard-coded boot path pointing to a GUID partition table (GPT) on the eMMC device. Once loaded, as a side effect, this path would be also stored in the UPVS (UEFI Persistent Variable Store) EEPROM. That is, if you use the `bfrec` tools provided in the `mlxbf-bfscripts` package to write this BFB to the eMMC boot partition (see bfrec man for more information), then during boot, the DPU would load this from the boot FIFO, and the UEFI would assume to boot off the eMMC.

BFB files can be useful for many things such as installing new software on a BlueField DPU. For example, the installation BFB for BlueField platforms normally contains an initramfs file in the BFB chain. Using the initramfs (and Linux kernel Image also found in the BFB) you can do things like set the boot partition on the eMMC using `mlx-bootctl` or flash new HCA firmware using MFT utilities. You can also install a full root file system on the eMMC while running out of the initramfs.

The following table presents the types of files possible in a BFB.

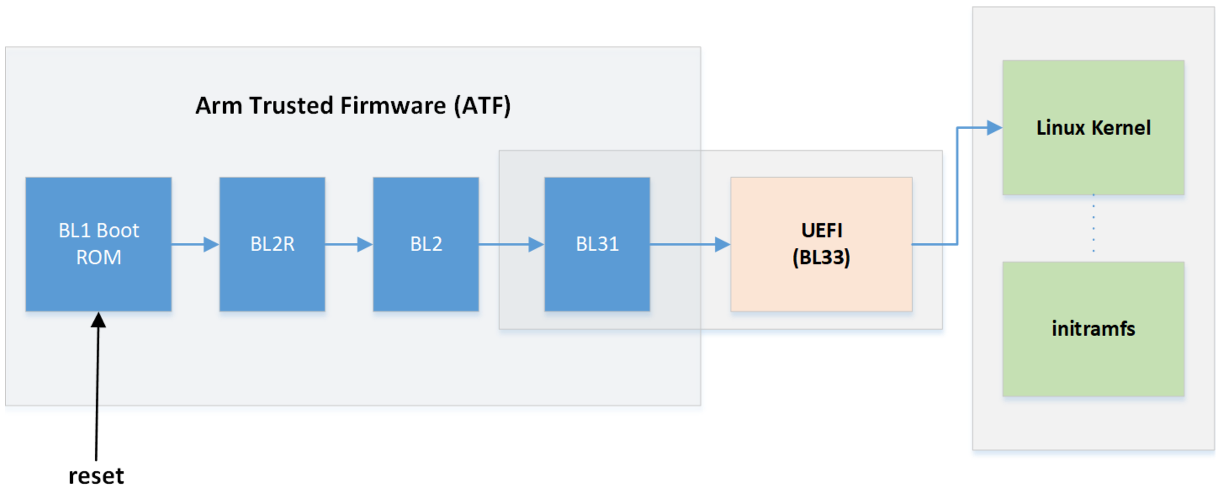
Filename	Description	ID	Read By
Bl2r-cert	Secure Firmware BL2R (RIoT Core) certificate	33	BL1

Filename	Description	ID	Read By
Bl2r	Secure Firmware BL2R (RIoT Core)	28	BL1
bl2-cert	Trusted Boot Firmware BL2 certificate	6	BL1/BL2R[a]
bl2	Trusted Boot Firmware BL2	1	BL1/BL2R[a]
trusted-key-cert	Trusted key certificate	7	BL2
bl31-key-cert	EL3 Runtime Firmware BL3-1 key certificate	9	BL2
bl31-cert	EL3 Runtime Firmware BL3-1 certificate	13	BL2
bl31	EL3 Runtime Firmware BL3-1	3	BL2
bl32-key-cert	Secure Payload BL3-2 (Trusted OS) key certificate	10	BL2
bl32-cert	Secure Payload BL3-2 (Trusted OS) certificate	14	BL2
bl32	Secure Payload BL3-2 (Trusted OS)	4	BL2
bl33-key-cert	Non-Trusted Firmware BL3-3 key certificate	11	BL2
bl33-cert	Non-Trusted Firmware BL3-3 certificate	15	BL2
bl33	Non-Trusted Firmware BL3-3	5	BL2
boot-acpi	Name of the ACPI table	55	UEFI
boot-dtb	Name of the DTB file	56	UEFI
boot-desc	Default boot menu item description	57	UEFI
boot-path	Boot image path	58	UEFI
boot-args	Arguments for boot image	59	UEFI
boot-timeout	Boot menu timeout	60	UEFI
image	Boot image	62	UEFI
initramfs	In-memory filesystem	63	UEFI

[a] When BL2R is booted in BlueField-2 devices, both the BL2 image and the BL2 certificate are read by BL2R. Thus, the BL2 image and certificate are read by BL1. BL2R is not booted in BlueField-1 devices.

Before explaining the implementation of the solution, the BlueField boot process needs to be expanded upon.

5.7.1.2 BlueField Boot Process



The BlueField boot flow is comprised of 4 main phases:

- Hardware loads Arm Trusted Firmware (ATF)
- ATF loads UEFI—together ATF and UEFI make up the booter software
- UEFI loads the operating system, such as the Linux kernel
- The operating system loads applications and user data

When booting from eMMC, these stages make use of two different types of storage within the eMMC part:

- ATF and UEFI are loaded from a special area known as the eMMC boot partition. Data from a boot partition is automatically streamed from the eMMC device to the eMMC controller under hardware control during the initial boot-up. Each eMMC device has two boot partitions, and the partition which is used to stream the boot data is chosen by a non-volatile configuration register in the eMMC.
- The operating system, applications, and user data come from the remainder of the chip, known as the user area. This area is accessed via block-size reads and writes, done by a device driver or similar software routine.

5.7.1.3 Upgrading Bootloader

In most deployments, the Arm cores of BlueField are expected to obtain their bootloader from an on-board eMMC device. Even in environments where the final OS kernel is not kept on eMMC—for instance, systems which boot over a network—the initial booter code still comes from the eMMC.

Most software stacks need to be modified or upgraded in their lifetime. Ideally, the user can to install the new software version on their BlueField system, test it, and then fall back to an older version if the new one does not work. In some environments, it is important that this fallback operation happen automatically since there may be no physical access to the system. In others, there may be an external agent, such as a service processor, which could manage the process.

To satisfy the requests listed above, the following must be performed:

1. Provision two software partitions on the eMMC, 0 and 1. At any given time, one area must be designated the primary partition, and the other the backup partition. The primary partition is the one booted on the next reboot or reset.
2. Allow software running on the Arm cores to declare that the primary partition is now the backup partition, and vice versa. (For the remainder of this section, this operation is referred to as "swapping the partitions" even though only the pointer is modified, and the data on the partitions does not move.)
3. Allow an external agent, such as a service processor, to swap the primary and backup partitions.
4. Allow software running on the Arm cores to reboot the system, while activating an upgrade watchdog timer. If the upgrade watchdog expires (due to the new image being broken, invalid, or corrupt), the system automatically reboots after swapping the primary and backup partitions.

5.7.1.4 Updating Boot Partition

The Bluefield software distribution provides a boot file that can be used to update the eMMC boot partitions. The BlueField boot file (BFB) is located in the boot directory `<BF_INST_DIR>/boot/` and contains all the necessary boot loader images (i.e. ATF binary file images and UEFI binary image).

The table below presents the pre-built boot images included within the BlueField software release:

Filename	Description
bl1.bin	The trusted firmware bootloader stage 1 (BL1) image, already stored into the on-chip boot ROM. It is executed when the device is reset.
bl2r.bin	The secure firmware (RIoT core) image. This image provides support for crypto operation and calculating measurements for security attestation and is relevant to BlueField-2 devices only.
bl2.bin	The trusted firmware bootloader stage 2 (BL2) image
bl31.bin	The trusted firmware bootloader stage 3-1 (BL31) image
BLUEFIELD_EFI.fd	The UEFI firmware image. It is also referred to as the non-trusted firmware bootloader stage 3-3 (BL33) image.
default.bfb	The BlueField boot file (BFB) which encapsulates all bootloader components such as bl2r.bin, bl2.bin, bl31.bin, and BLUEFIELD_EFI.fd. This file may be used to boot the BlueField devices from the RShim interface. It also could be installed into the eMMC boot partition.

It is also possible to build bootloader images from sources and create the BlueField boot file (BFB). Please refer to the sections below for more details.

The software image includes various tools and utilities to update the eMMC boot partitions. It also embeds a boot file in `/lib/firmware/mellanox/boot/default.bfb`. To update the eMMC boot partitions using the embedded boot file, execute the following command from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec
```

`bfrec` is also available under `/usr/bin`.

The boot partitions update is initiated by the `bfrec` tool at runtime. With no options specified, the "bfrec" uses the default boot file `/lib/firmware/mellanox/boot/default.bfb` to update the boot partitions of device `/dev/mmcblk0`. This might be done directly in an OS using the "mlxbf-bootctl" utility, or at a later stage after reset using the capsule interface.

The syntax of `bfrec` is as follows:

```
Syntax: bfrec [--help]
              [--bootctl [<FILE>]]
              [--capsule [<FILE>]]


Description:
--help                  : print help
--bootctl [<FILE>]      : update the boot partition via the kernel path. If no FILE is specified, then default is
used.
--capsule [<FILE>]      : update the boot partition via the capsule path. If no FILE is specified, then default is
used.
--policy POLICY         : determines the update policy. May be: single - updates the secondary partition and swaps
to it, dual - updates both boot partitions, does not swap. If this flag is not specified, 'single' policy is
assumed.
```

When `bfrec` is called with the option `--bootctl`, the tool uses the boot file FILE, if given, rather than the default `/lib/firmware/mellanox/boot/default.bfb` in order to update the boot partitions. The command line usage is as follows:

```
$ bfrec --bootctl
$ bfrec --bootctl FILE
```

Where FILE represents the BlueField boot file encapsulating the new bootloader images to be written to the eMMC boot partitions.

For example, if the new bootstream file which we would like to install and validate is called `newdefault.bfb`, download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
# /opt/mellanox/scripts/bfrec --bootctl newdefault.bfb
```

The `--capsule` option updates the boot partition via the capsule interface. The capsule update image is reported in UEFI, so that at a later point the bootloader consumes the capsule file and performs the boot partition update. This option might be executed with or without additional arguments. The command line usage is as follows:

```
$ bfrec --capsule
$ bfrec --capsule FILE
```

Where FILE represents the capsule update image file encapsulating the new boot image to be written to the eMMC boot partitions.

For example, if the new bootstream file which we want to install and validate is called "`newdefault.bfb`", download the file to the BlueField and update the eMMC boot partitions by executing the following commands from the BlueField console:

```
$ /opt/mellanox/scripts/bfrec --capsule newdefault.bfb $ reboot
```

For more information about the capsule updates, please refer to `<BF_INST_DIR>/Documentation/HOWTO-capsule`.

After reset, the BlueField platform boots from the newly updated boot partition. To verify the version of ATF and UEFI, execute the following command:

```
$ /opt/mellanox/scripts/bfver
```

5.7.1.4.1 mlxbf-bootctl

It is also possible to update the eMMC boot partitions directly with the `mlxbf-bootctl` tool. The tool is shipped as part of the software image (under `/sbin`) and the sources are shipped in the `src` directory in the BlueField Runtime Distribution. A simple `make` command builds the utility.

The syntax of `mlxbf-bootctl` is as follows:

```
syntax: mlxbf-bootctl [--help | -h] [--swap | -s]
                      [--device | -d MMCFILE]
                      [--output | -o OUTPUT] [--read | -r INPUT]
                      [--bootstream | -b BFBFILE]
                      [--overwrite-current]
                      [--watchdog-swap interval | --nowatchdog-swap]
```

Where:

- `--device` - use a device other than the default `/dev/mmcblk0`
- `--bootstream` - write the specified bootstream to the alternate partition of the device. This queries the base device (e.g. `/dev/mmcblk0`) for the alternate partition, and uses that information to open the appropriate boot partition device (e.g. `/dev/mmcblk0boot0`).
- `--overwrite-current` (used with "`--bootstream`") - overwrite the current boot partition instead of the alternate one

  Not recommended as there is no easy way to recover if the new bootloader code does not bring the system up. Use `--swap` instead.

- `--output` (used with "`--bootstream`") - specify a file to which to write the boot partition data (creating it if necessary), rather than using an existing master device and deriving the boot partition device
- `--watchdog-swap` - arrange to start the Arm watchdog timer with a countdown of the specified number of seconds until it triggers; also, set the boot software so that it swaps the primary and alternate partitions at the next reset
- `--nowatchdog-swap` - ensure that after the next reset, no watchdog is started, and no swapping of boot partitions occurs

To update the boot partitions, execute the following command:

```
$ mlxbf-bootctl --swap --device /dev/mmcblk0 --bootstream default.bfb
```

This writes the new bootstream to the alternate boot partition, swaps alternate and primary so that the new bootstream is used on the next reboot.

It is recommended to enable the watchdog when calling `mlxbf-bootcl` in order to ensure that the Arm bootloader can perform alternate boot in case of a nonfunctional bootloader code within the primary boot partition. If something goes wrong on the next reboot and the system does not come up properly, it will reboot and return to the original configuration. To do so, the user may run:

```
$ mlxbf-bootctl --bootstream bootstream.new --swap --watchdog-swap 60
```

This reboots the system, and if it hangs for 60 seconds or more, the watchdog fires and resets the chip, the bootloader swaps the partitions back again to the way they were before, and the system reboots back with the original boot partition data. Similarly, if the system comes up but panics and resets, the bootloader will again swap the boot partition back to the way it was before.

The user must ensure that Linux after the reboot is configured to boot up with the `sbsa_gwdt` driv er enabled. This is the Server Base System Architecture (SBSA) Generic WatchDog Timer. As soon as the driver is loaded, it begins refreshing the watchdog and preventing it from firing, which allows the system to finish booting up safely. In the example above, 60 seconds are allowed from system reset until the Linux watchdog kernel driver is loaded. At that point, the user's application may open /dev/watchdog explicitly, and the application would then become responsible for refreshing the watchdog frequently enough to keep the system from rebooting.

For documentation on the Linux watchdog subsystem, see Linux watchdog documentation.

To disable the watchdog completely, run:

```
$ echo V > /dev/watchdog
```

The user may select to incorporate other features of the Arm generic watchdog into their application code using the programming API as well.

Once the system has booted up, in addition to disabling or reconfiguring the watchdog itself if the user desires, they must also clear the "swap on next reset" functionality from the bootloader by running:

```
$ mlxbf-bootctl --nowatchdog-swap
```

Otherwise, next time the system is reset (via reboot, external reset, etc.) it assumes a failure or watchdog reset occurred and swaps the eMMC boot partition automatically.

5.7.1.4.2 LVFS and fwupd

Officially released bootloaders (ATF and UEFI) may be alternatively installed from the LVFS (Linux Vendor Firmware Service). LVFS is a free service operated by the Linux Foundation, which allows vendors to host stable firmware images for easy download and installation.

> The DPU must have a functioning connection to the Internet.

Interaction with LVFS is carried out through a standard open-source tool called fwupd. fwupd is an updater daemon that runs in the background, waiting for commands from a management application. fwupd and the command line manager, fwupdmgr, comes pre-installed on the BlueField Ubuntu image.

To verify bootloader support for a fwupd update, run the following command:

```
$ fwupdmgr get-devices
```

If "UEFI Device Firmware" device appears, then your currently installed bootloader supports the update process. Other devices may appear depending on your distribution of choice. Version numbers similar to 0.0.0.1 may appear if you are using an older version of the bootloader.

1. Before updating, a fresh list of release metadata must be obtained. Run:

   ```
   $ fwupdmgr refresh
   ```
2. Optionally, to confirm if a new release is available, run:

   ```
   $ fwupdmgr get-releases
   ```
3. Update your system bootloader, run "upgrade" with the GUID of the UEFI device. Run:

   ```
   $ fwupdmgr upgrade 39342586-4e0e-4833-b4ba-1256b0ffb471
   ```
   This will upgrade the ATF and UEFI to the latest available stable version of the bootloader through a UEFI capsule update, without upgrading the root file system. If your system is already at the latest available version, this upgrade command will do nothing.
4. Reboot the DPU to complete the upgrade.

Installing boot firmware directly through mlxbf-bootctl may cause fwupdmgr to detect an incorrect version string. If your workflow depends on fwupd, try to update the bootloader through capsule update (i.e. bfrec --capsule) or fwupdmgr only.

For more information about LVFS and fwupd, please refer to the official website of LVFS.

5.7.1.4.3 Updating Boot Partitions with BMC

The Arm cores notify the BMC prior to the reboot that an upgrade is about to happen. Software running on the BMC can then be implemented to watch the Arm cores after reboot. If after some time the BMC does not detect the Arm cores come up properly, it can use its USB debug connection to the Arm cores to properly reset the Arm cores. It first sets a suitable mode bit that the Arm bootloader responds to by switching the primary and alternating boot partitions as part of resetting into its original state.

5.7.1.5 Creating BlueField Boot File

The BlueField software distribution provides tools to format and to package the bootloader images into a single bootable file.

To create the BlueField boot file, use the `mlx-mkbfb` tool with the appropriate images. The bootloader images are embedded within the BSD under `<BF_INST_DIR>/boot/`. It is also possible to build the binary images from sources. Please refer to the following sections for further details.

1. First, set the PATH variable:

```
$ export PATH=$PATH:<BF_INST_DIR>/bin
```

2. Then, generate the boot file by using the `mlx-mkbfb` command:

```
$ mlx-mkbfb \ --bl2 bl2.bin \ --bl31 bl31.bin \ --bl33 BLUEFIELD_EFI.fd \ --boot-acpi "=default" \
default.bfb
```

   This command creates the `default.bfb` from `bl2.bin`, `bl31.bin`, and `BLUEFILED_EFI.fd`. The generated file might be used to update the eMMC boot partitions.

To verify the content of the boot file, run:

```
$ mlx-mkbfb -d default.bfb
```

To extract the bootloader images from the boot file, run:

```
$ mlx-mkbfb -x default.bfb
```

To obtain further details about the tool options, run the tool with `-h` or `--help`.

5.7.1.6 UEFI Boot Management

The UEFI firmware provides boot management function that can be configured by modifying architecturally defined global variables which are stored in the UPVS EEPROM. The boot manager will attempt to load and boot the OS in an order defined by the persistent variables.

The UEFI boot manager can be configured; boot entries may be added or removed from the boot menu. The UEFI firmware can also effectively generate entries in this boot menu, according to the available network interfaces and possibly the disks attached to the system.

5.7.1.6.1 Boot Option

The boot option is a unique identifier for a UEFI boot entry. This identifier is assigned when the boot entry is created, and it does not change. It also represents the boot option in several lists, including the BootOrder array, and it is the name of the directory on disk in which the system stores data related to the boot entry, including backup copies of the boot entry. A UEFI boot entry ID has the format "`Bootxxxx`" where `xxxx` is a hexadecimal number that reflects the order in which the boot entries are created.

Besides the boot entry ID, the UEFI boot entry has the following fields:

- Description (e.g. Yocto, CentOS, Linux from RShim)
- Device Path (e.g. VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image)
- Boot arguments (e.g. console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs)

5.7.1.6.2 List UEFI Boot Options

To display the boot option already installed in the NVIDIA® BlueField® system, reboot and go to the UEFI menu screen. To get to the UEFI menu, hit Esc when prompted (in the RShim or UART console) before the countdown timer runs out.

```
Press <ESC> twice to enter UEFI menu
3 seconds remaining
2 seconds remaining
1 seconds remaining
```

Boot options are listed as soon as you select the "Boot Manager" entry.

```
������������������������������������������������
�                                  Boot Manager                                      �
������������������������������������������������

                                                            Device Path :
   Boot Option Menu                                         HD(1,GPT,B55B6B71-964E
                                                            -714B-AAF8-7AE8D768372
   focal0                                                   7,0x800,0x19000)/\EFI\
   ubuntu                                                   ubuntu\shimaa64.efi
   Linux from rshim
   Linux from mmc0
   EFI Internal Shell
   EFI Misc Device
   EFI Network
   EFI Network 1
   EFI Network 2
   EFI Network 3
   EFI Network 4
   EFI Network 5
                                                         v
������������������������������������������������
�                                      �
� ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit                     �
������������������������������������������������
```

It is also possible to retrieve more details about the boot entries. To do so, select "EFI Internal Shell" entry from the Boot Manager screen.

```
UEFI Interactive Shell v2.1
EDK II
UEFI v2.50 (EDK II, 0x00010000)
Mapping table
      FS1: Alias(s):F1:
          VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)
      FS0: Alias(s):HD0b:;BLK1:
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD)
     BLK0: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
     BLK2: Alias(s):
          VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)/HD(2,GPT,9E61E8B5-EC9C-4299-8A0B-1B42E3DB)


Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell>
```

From the UEFI shell, you may run the following command to display the option list:

```
Shell> bcfg boot dump -v
```

Here `-v` displays the option list with extra info including boot parameters. The following is an output example:

```
Option: 00. Variable: Boot0000
  Desc    - Linux from rshim
  DevPath - VenHw(F019E406-8C9C-11E5-8797-001ACA00BFC4)/Image
  Optional- Y
  00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
  00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00  *t.t.y.A.M.A.0. .*
  00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
  00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
  00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
  00000050: 30 00 20 00 20 00 69 00-6E 00 69 00 74 00 72 00  *0. . .i.n.i.t.r.*
```

```
    00000060: 64 00 3D 00 69 00 6E 00-69 00 74 00 72 00 61 00  *d.=.i.n.i.t.r.a.*
    00000070: 6D 00 66 00 73 00 00 00-                         *m.f.s...*
  Option: 01. Variable: Boot0002
    Desc    - Yocto Poky
    DevPath - HD(1,GPT,3DCADB7E-BCCC-4897-A766-3C070EDD7C25,0x800,0xAE800)/Image
    Optional- Y
    00000000: 63 00 6F 00 6E 00 73 00-6F 00 6C 00 65 00 3D 00  *c.o.n.s.o.l.e.=.*
    00000010: 74 00 74 00 79 00 41 00-4D 00 41 00 30 00 20 00 *t.t.y.A.M.A.0. .*
    00000020: 65 00 61 00 72 00 6C 00-79 00 63 00 6F 00 6E 00  *e.a.r.l.y.c.o.n.*
    00000030: 3D 00 70 00 6C 00 30 00-31 00 31 00 2C 00 30 00  *=.p.l.0.1.1.,.0.*
    00000040: 78 00 30 00 31 00 30 00-30 00 30 00 30 00 30 00  *x.0.1.0.0.0.0.0.*
    00000050: 30 00 20 00 72 00 6F 00-6F 00 74 00 3D 00 2F 00  *0. .r.o.o.t.=./.*
    00000060: 64 00 65 00 76 00 2F 00-6D 00 6D 00 63 00 62 00  *d.e.v./.m.m.c.b.*
    00000070: 6C 00 6B 00 30 00 70 00-32 00 20 00 72 00 6F 00  *l.k.0.p.2. .r.o.*
    00000080: 6F 00 74 00 77 00 61 00-69 00 74 00              *o.t.w.a.i.t.*
      Option: 02. Variable: Boot0003
        Desc    - EFI Misc Device
        DevPath - VenHw(8C91E049-9BF9-440E-BBAD-7DC5FC082C02)
        Optional- N
      Option: 03. Variable: Boot0004
        Desc    - EFI Network
        DevPath - MAC(001ACAFFFF01,0x1)
        Optional- N
      Option: 04. Variable: Boot0005
        Desc    - EFI Network 1
        DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)
        Optional- N
      Option: 05. Variable: Boot0006
        Desc    - EFI Network 2
        DevPath - MAC(001ACAFFFF01,0x1)/IPv6(0000:0000:0000:0000:0000:0000:0000:0000)
        Optional- N
      Option: 06. Variable: Boot0007
        Desc    - EFI Network 3
        DevPath - MAC(001ACAFFFF01,0x1)/IPv4(0.0.0.0)/Uri()
        Optional- N
      Option: 07. Variable: Boot0008
        Desc    - EFI Internal Shell
        DevPath - MemoryMapped(0xB,0xFE5FE000,0xFEAE357F)/FvFile(7C04A583-9E3E-4F1C-AD65-E05268D0B4D1)
        Optional- N
```

Boot arguments are printed in Hex mode, but you may recognize the boot parameters printed on the side in ASCII format.

5.7.1.6.3 UEFI System Configuration

UEFI System Configuration menu can be accessed under UEFI menu → Device Manager → System Configuration.

The following options are supported:

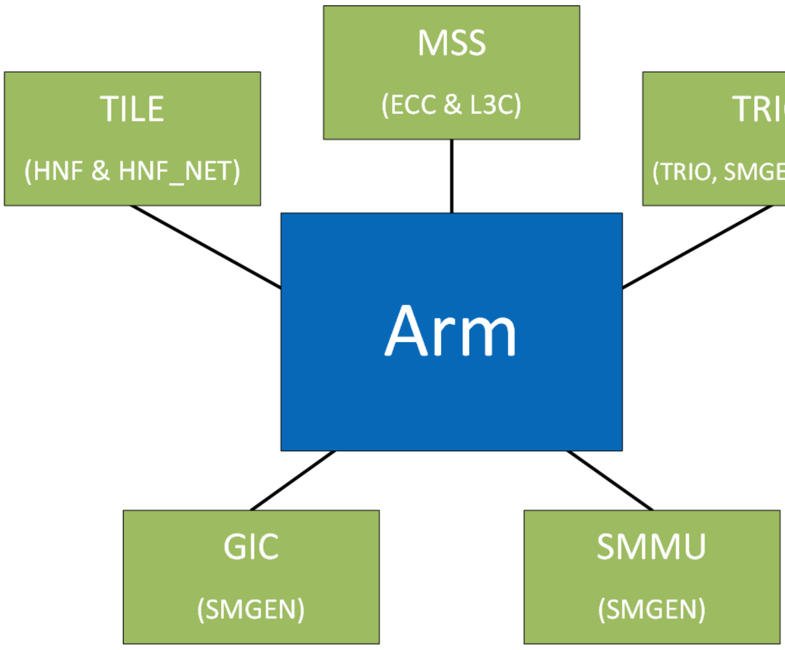
- Set Password - set a password for UEFI. Default: No password.
- Select SPCR UART - choose UART for Port Console Redirection. Default: Disabled.
- Enable SMMU - enable SMMU in ACPI. Default: Disabled.
- Disable SPMI - disable/enable ACPI SPMI Table. Default: Enabled.
- Enable 2nd eMMC - this option is relevant only for some BlueField Reference Platform boards. Default: Disabled.
- Boot Partition Protection – enable eMMC boot partition so it can be updated by the UEFI capsule only
- Disable PCIe – disable PCIe in ACPI. Default: Enabled.
- Disable ForcePXERetry – if ForcePXE is enabled from the BMC, the boot process keeps retrying PXE boot if it fails unless this option is enabled. If ForcePXERetry is disabled, the boot process only attempts PXE boot once, then it retries the normal boot flow if all PXE boot entries fail.
- Reset EFI Variables – clears all EFI variables to factory default state and disables SMMU and wipes the BOOT option variables and secure boot keys
- Reset MFG Info – clears the manufacturing information

All the above options, except for password and the two reset options, are also programmatically configurable via the BlueField Linux `/etc/bf.cfg`. Refer to section "bf.cfg Parameters" for further information.

6 Management

- Performance Monitoring Counters
- Intelligent Platform Management Interface
- Redfish
- Logging
- SoC Management Interface
- BlueField OOB Ethernet Interface

6.1 Performance Monitoring Counters



The performance modules in NVIDIA® BlueField® are present in several hardware blocks and each block has a certain set of supported events.

The `mlx_pmc` driver provides access to all of these performance modules through a sysfs interface. The driver creates a directory under `/sys/class/hwmon` under which each of the blocks explained above has a subdirectory. Please note that all directories under `/sys/class/hwmon` are named as "`hwmon<N>`" where `N` is the `hwmon` device number corresponding to the device. This is assigned by Linux and could change with the addition of more devices to the `hwmon` class. Each `hwmon` directory has a "`name`" node which can be used to identify the correct device. In this case, reading the "`name`" file should return "`bfperf`".

The hardware blocks that include performance modules are:

- Tile (block containing 2 cores and a shared L2 cache) has 2 sets of counters, one set for HNF and HNF_NET events. These are present as "tile" and "tilenet" directories in the sysfs interface of the driver.
- TRIO (PCIe root complex) has 3 sets of counters, one each for TRIO, SMGEN and PCIE TLR events. The sysfs directories for these are called "trio", "triogen" and "pcie" respectively.
- MSS (memory sub-system containing the memory controller and L3 cache)

- GIC and SMMU with one set of counters each for the SMGEN events. These are simply labelled "gic" and "smmu" respectively.

The number of Tile, TRIO and MSS blocks depends on the system. There is a maximum of 8 Tile, 3 TRIO and 2 MSS blocks in BlueField, and this is added as a suffix to the sysfs directory names. For example, this is a list of directories present in a BlueField-2 system:

```
ubuntu@dpu:/$ ls /sys/class/hwmon/hwmon0/
device l3cachehalf0 pcie0 smmu0 tile1 tilenet0 tilenet3 triogen0
ecc l3cachehalf1 pcie1 subsystem tile2 tilenet1 trio0 triogen1
gic0 name power  tile0 tile3 tilenet2 trio1 uevent
```

The PCIe TLR statistics for each TRIO are under the "pcie" block.

6.1.1 Performance Data Collection Mechanisms

The performance data of the BlueField hardware is collected using two mechanisms:

1. Programming hardware counters to monitor specific events
2. Reading registers that hold performance/event statistics

All blocks except "ecc" and "pcie" use the mechanism 1.

6.1.1.1 Using Hardware Counters

For blocks that use hardware counters to collect data, each counter present in the block is represented by "`event<N>`" and "`counter<N>`" sysfs files.

For example:

```
ubuntu@dpu:/$ ls /sys/class/hwmon/hwmon0/tile0/
counter0 counter1 counter2 counter3 event0 event1 event2 event3 event_list
```

An `event<N>` and `counter<N>` pair can be used to program and monitor events. The "`event_list`" sysfs file displays the list of events supported by that block along with the hexadecimal value corresponding to each event.

Use the `echo` command to write the event number to the `event<N>` file, and use the `cat` command to read the counter value from the corresponding counter (`counter<N>`).

The counters are enabled individually once the event number is written to the corresponding event file. However, the L3 cache performance counters cannot be enabled or disabled individually and can only be triggered or stopped all at the same time.

So in the example provided, all 4 event files may be programmed with the necessary event numbers and then the "enable" file may be used to start the counters. Writing `0` to the enable file stops the counters while `1` starts them.

6.1.1.2 Reading Registers

For "ecc" and "pcie" blocks, the counters cannot be started or stopped by the user, instead the statistics are automatically collected by HW and stored in registers. These register names are exposed within the directory and can be read by the user at any time.

6.1.2 List of Supported Events

6.1.2.1 SMGEN Performance Module

Hex Value	Name	Description
0x0	AW_REQ	Reserved for internal use
0x1	AW_BEATS	Reserved for internal use
0x2	AW_TRANS	Reserved for internal use
0x3	AW_RESP	Reserved for internal use
0x4	AW_STL	Reserved for internal use
0x5	AW_LAT	Reserved for internal use
0x6	AW_REQ_TBU	Reserved for internal use
0x8	AR_REQ	Reserved for internal use
0x9	AR_BEATS	Reserved for internal use
0xa	AR_TRANS	Reserved for internal use
0xb	AR_STL	Reserved for internal use
0xc	AR_LAT	Reserved for internal use
0xd	AR_REQ_TBU	Reserved for internal use
0xe	TBU_MISS	The number of TBU miss
0xf	TX_DAT_AF	Mesh Data channel write FIFO almost Full. This is from the TRIO toward the Arm memory.
0x10	RX_DAT_AF	Mesh Data channel read FIFO almost Full. This is from the Arm memory toward the TRIO.
0x11	RETRYQ_CRED	Reserved for internal use

6.1.2.2 Tile HNF Performance Module

Hex Value	Name	Description
0x45	HNF_REQUESTS	Number of REQs that were processed in HNF
0x46	HNF_REJECTS	Reserved for internal use
0x47	ALL_BUSY	Reserved for internal use
0x48	MAF_BUSY	Reserved for internal use
0x49	MAF_REQUESTS	Reserved for internal use
0x4a	RNF_REQUESTS	Number of REQs sent by the RN-F selected by HNF_PERF_CTL register RNF_SEL field
0x4b	REQUEST_TYPE	Reserved for internal use
0x4c	MEMORY_READS	Number of reads to MSS
0x4d	MEMORY_WRITES	Number of writes to MSS

Hex Value	Name	Description
0x4e	VICTIM_WRITE	Number of victim lines written to memory
0x4f	POC_FULL	Reserved for internal use
0x50	POC_FAIL	Number of times that the POC Monitor sent RespErr Okay status to an Exclusive WriteNoSnp or CleanUnique REQ
0x51	POC_SUCCESS	Number of times that the POC Monitor sent RespErr ExOkay status to an Exclusive WriteNoSnp or CleanUnique REQ
0x52	POC_WRITES	Number of Exclusive WriteNoSnp or CleanUnique REQs processed by POC Monitor
0x53	POC_READS	Number of Exclusive ReadClean/ReadShared REQs processed by POC Monitor
0x54	FORWARD	Reserved for internal use
0x55	RXREQ_HNF	Reserved for internal use
0x56	RXRSP_HNF	Reserved for internal use
0x57	RXDAT_HNF	Reserved for internal use
0x58	TXREQ_HNF	Reserved for internal use
0x59	TXRSP_HNF	Reserved for internal use
0x5a	TXDAT_HNF	Reserved for internal use
0x5b	TXSNP_HNF	Reserved for internal use
0x5c	INDEX_MATCH	Reserved for internal use
0x5d	A72_ACCESS	Access requests (Reads, Writes, CopyBack, CMO, DVM) from A72 clusters
0x5e	IO_ACCESS	Accesses requests (Reads, Writes) from DMA IO devices
0x5f	TSO_WRITE	Total Store Order write Requests from DMA IO devices
0x60	TSO_CONFLICT	Reserved for internal use
0x61	DIR_HIT	Requests that hit in directory
0x62	HNF_ACCEPTS	Reserved for internal use
0x63	REQ_BUF_EMPTY	Number of cycles when request buffer is empty
0x64	REQ_BUF_IDLE_MAF	Reserved for internal use
0x65	TSO_NOARB	Reserved for internal use
0x66	TSO_NOARB_CYCLES	Reserved for internal use
0x67	MSS_NO_CREDIT	Number of cycles that a Request could not be sent to MSS due to lack of credits
0x68	TXDAT_NO_LCRD	Reserved for internal use
0x69	TXSNP_NO_LCRD	Reserved for internal use
0x6a	TXRSP_NO_LCRD	Reserved for internal use
0x6b	TXREQ_NO_LCRD	Reserved for internal use

Hex Value	Name	Description
0x6c	TSO_CL_MATCH	Reserved for internal use
0x6d	MEMORY_READS_BYPASS	Number of reads to MSS that bypass Home Node
0x6e	TSO_NOARB_TIMEOUT	Reserved for internal use
0x6f	ALLOCATE	Number of times that Directory entry was allocated
0x70	VICTIM	Number of times that Directory entry allocation did not find an Invalid way in the set
0x71	A72_WRITE	Write requests from A72 clusters
0x72	A72_Read	Read requests from A72 clusters
0x73	IO_WRITE	Write requests from DMA IO devices
0x74	IO_Reads	Read requests from DMA IO devices
0x75	TSO_Reject	Reserved for internal use
0x80	TXREQ_RN	Reserved for internal use
0x81	TXRSP_RN	Reserved for internal use
0x82	TXDAT_RN	Reserved for internal use
0x83	RXSNP_RN	Reserved for internal use
0x84	RXRSP_RN	Reserved for internal use
0x85	RXDAT_RN	Reserved for internal use

6.1.2.3 TRIO Performance Module

Hex Value	Name	Description
0xa0	TPIO_DATA_BEAT	Data beats from Arm PIO to TRIO
0xa1	TDMA_DATA_BEAT	Data beats from Arm memory to PCI completion
0xa2	MAP_DATA_BEAT	Reserved for internal use
0xa3	TXMSG_DATA_BEAT	Reserved for internal use
0xa4	TPIO_DATA_PACKET	Data packets from Arm PIO to TRIO
0xa5	TDMA_DATA_PACKET	Data packets from Arm memory to PCI completion
0xa6	MAP_DATA_PACKET	Reserved for internal use
0xa7	TXMSG_DATA_PACKET	Reserved for internal use
0xa8	TDMA_RT_AF	The in-flight PCI DMA READ request queue is almost full
0xa9	TDMA_PBUF_MAC_AF	Indicator of the buffer of Arm memory reads is too full awaiting PCIe access
0xaa	TRIO_MAP_WRQ_BUF_EMPTY	PCIe write transaction buffer is empty
0xab	TRIO_MAP_CPL_BUF_EMPTY	Arm PIO request completion queue is empty
0xac	TRIO_MAP_RDQ0_BUF_EMPTY	The buffer of MAC0's read transaction is empty
0xad	TRIO_MAP_RDQ1_BUF_EMPTY	The buffer of MAC1's read transaction is empty

Hex Value	Name	Description
0xae	TRIO_MAP_RDQ2_BUF_EMPTY	The buffer of MAC2's read transaction is empty
0xaf	TRIO_MAP_RDQ3_BUF_EMPTY	The buffer of MAC3's read transaction is empty
0xb0	TRIO_MAP_RDQ4_BUF_EMPTY	The buffer of MAC4's read transaction is empty
0xb1	TRIO_MAP_RDQ5_BUF_EMPTY	The buffer of MAC5's read transaction is empty
0xb2	TRIO_MAP_RDQ6_BUF_EMPTY	The buffer of MAC6's read transaction is empty
0xb3	TRIO_MAP_RDQ7_BUF_EMPTY	The buffer of MAC7's read transaction is empty

6.1.2.4 L3 Cache Performance Module

The L3 cache interfaces with the Arm cores via the SkyMesh. The CDN is used for control data. The NDN is used for responses. The DDN is for the actual data transfer.

Hex Value	Name	Description
0x00	DISABLE	Reserved for internal use
0x01	CYCLES	Timestamp counter
0x02	TOTAL_RD_REQ_IN	Read Transaction control request from the CDN of the SkyMesh
0x03	TOTAL_WR_REQ_IN	Write transaction control request from the CDN of the SkyMesh
0x04	TOTAL_WR_DBID_ACK	Write transaction control responses from the NDN of the SkyMesh
0x05	TOTAL_WR_DATA_IN	Write transaction data from the DDN of the SkyMesh
0x06	TOTAL_WR_COMP	Write completion response from the NDN of the SkyMesh
0x07	TOTAL_RD_DATA_OUT	Read transaction data from the DDN
0x08	TOTAL_CDN_REQ_IN_BANK0	CHI CDN Transactions Bank 0
0x09	TOTAL_CDN_REQ_IN_BANK1	CHI CDN Transactions Bank 1
0x0a	TOTAL_DDN_REQ_IN_BANK0	CHI DDN Transactions Bank 0
0x0b	TOTAL_DDN_REQ_IN_BANK1	CHI DDN Transactions Bank 1
0x0c	TOTAL_EMEM_RD_RES_IN_BANK0	Total EMEM Read Response Bank 0
0x0d	TOTAL_EMEM_RD_RES_IN_BANK1	Total EMEM Read Response Bank 1
0x0e	TOTAL_CACHE_RD_RES_IN_BANK0	Total Cache Read Response Bank 0
0x0f	TOTAL_CACHE_RD_RES_IN_BANK1	Total Cache Read Response Bank 1
0x10	TOTAL_EMEM_RD_REQ_BANK0	Total EMEM Read Request Bank 0
0x11	TOTAL_EMEM_RD_REQ_BANK1	Total EMEM Read Request Bank 1
0x12	TOTAL_EMEM_WR_REQ_BANK0	Total EMEM Write Request Bank 0
0x13	TOTAL_EMEM_WR_REQ_BANK1	Total EMEM Write Request Bank 1

Hex Value	Name	Description
0x14	TOTAL_RD_REQ_OUT	EMEM Read Transactions Out
0x15	TOTAL_WR_REQ_OUT	EMEM Write Transactions Out
0x16	TOTAL_RD_RES_IN	EMEM Read Transactions In
0x17	HITS_BANK0	Number of Hits Bank 0
0x18	HITS_BANK1	Number of Hits Bank 1
0x19	MISSES_BANK0	Number of Misses Bank 0
0x1a	MISSES_BANK1	Number of Misses Bank 1
0x1b	ALLOCATIONS_BANK0	Number of Allocations Bank 0
0x1c	ALLOCATIONS_BANK1	Number of Allocations Bank 1
0x1d	EVICTIONS_BANK0	Number of Evictions Bank 0
0x1e	EVICTIONS_BANK1	Number of Evictions Bank 1
0x1f	DBID_REJECT	Reserved for internal use
0x20	WRDB_REJECT_BANK0	Reserved for internal use
0x21	WRDB_REJECT_BANK1	Reserved for internal use
0x22	CMDQ_REJECT_BANK0	Reserved for internal use
0x23	CMDQ_REJECT_BANK1	Reserved for internal use
0x24	COB_REJECT_BANK0	Reserved for internal use
0x25	COB_REJECT_BANK1	Reserved for internal use
0x26	TRB_REJECT_BANK0	Reserved for internal use
0x27	TRB_REJECT_BANK1	Reserved for internal use
0x28	TAG_REJECT_BANK0	Reserved for internal use
0x29	TAG_REJECT_BANK1	Reserved for internal use
0x2a	ANY_REJECT_BANK0	Reserved for internal use
0x2b	ANY_REJECT_BANK1	Reserved for internal use

6.1.2.5 PCIe TLR Statistics

Hex Value	Name	Description
0x0	PCIE_TLR_IN_P_PKT_CNT	Incoming posted packets
0x10	PCIE_TLR_IN_NP_PKT_CNT	Incoming non-posted packets
0x18	PCIE_TLR_IN_C_PKT_CNT	Incoming completion packets
0x20	PCIE_TLR_OUT_P_PKT_CNT	Outgoing posted packets
0x28	PCIE_TLR_OUT_NP_PKT_CNT	Outgoing non-posted packets
0x30	PCIE_TLR_OUT_C_PKT_CNT	Outgoing completion packets
0x38	PCIE_TLR_IN_P_BYTE_CNT	Incoming posted bytes
0x40	PCIE_TLR_IN_NP_BYTE_CNT	Incoming non-posted bytes

Hex Value	Name	Description
0x48	PCIE_TLR_IN_C_BYTE_CNT	Incoming completion bytes
0x50	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing posted bytes
0x58	PCIE_TLR_OUT_NP_BYTE_CNT	Outgoing non-posted bytes
0x60	PCIE_TLR_OUT_C_BYTE_CNT	Outgoing completion bytes

6.1.2.6 Tile HNFNET Performance Module

Hex Value	Name	Description
0x12	CDN_REQ	The number of CDN requests
0x13	DDN_REQ	The number of DDN requests
0x14	NDN_REQ	The number of NDN requests
0x15	CDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the CDN network
0x16	CDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the CDN network
0x17	CDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the CDN network
0x18	CDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the CDN network
0x19	CDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the CDN network
0x1a	CDN_DIAG_N_EGRESS	Packets sent out from north port in the CDN network
0x1b	CDN_DIAG_S_EGRESS	Packets sent out from south port in the CDN network
0x1c	CDN_DIAG_E_EGRESS	Packets sent out from east port in the CDN network
0x1d	CDN_DIAG_W_EGRESS	Packets sent out from west port in the CDN network
0x1e	CDN_DIAG_C_EGRESS	Packets sent out from core port in the CDN network
0x1f	CDN_DIAG_N_INGRESS	Packets received by north port in the CDN network
0x20	CDN_DIAG_S_INGRESS	Packets received by south port in the CDN network
0x21	CDN_DIAG_E_INGRESS	Packets received by east port in the CDN network
0x22	CDN_DIAG_W_INGRESS	Packets received by west port in the CDN network
0x23	CDN_DIAG_C_INGRESS	Packets received by core port in the CDN network
0x24	CDN_DIAG_CORE_SENT	Packets completed from core port in the CDN network

Hex Value	Name	Description
0x25	DDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the DDN network
0x26	DDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the DDN network
0x27	DDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the DDN network
0x28	DDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the DDN network
0x29	DDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the DDN network
0x2a	DDN_DIAG_N_EGRESS	Packets sent out from north port in the DDN network
0x2b	DDN_DIAG_S_EGRESS	Packets sent out from south port in the DDN network
0x2c	DDN_DIAG_E_EGRESS	Packets sent out from east port in the DDN network
0x2d	DDN_DIAG_W_EGRESS	Packets sent out from west port in the DDN network
0x2e	DDN_DIAG_C_EGRESS	Packets sent out from core port in the DDN network
0x2f	DDN_DIAG_N_INGRESS	Packets received by north port in the DDN network
0x30	DDN_DIAG_S_INGRESS	Packets received by south port in the DDN network
0x31	DDN_DIAG_E_INGRESS	Packets received by east port in the DDN network
0x32	DDN_DIAG_W_INGRESS	Packets received by west port in the DDN network
0x33	DDN_DIAG_C_INGRESS	Packets received by core port in the DDN network
0x34	DDN_DIAG_CORE_SENT	Packets completed from core port in the DDN network
0x35	NDN_DIAG_N_OUT_OF_CRED	Number of cycles that north input port FIFO runs out of credits in the NDN network
0x36	NDN_DIAG_S_OUT_OF_CRED	Number of cycles that south input port FIFO runs out of credits in the NDN network
0x37	NDN_DIAG_E_OUT_OF_CRED	Number of cycles that east input port FIFO runs out of credits in the NDN network
0x38	NDN_DIAG_W_OUT_OF_CRED	Number of cycles that west input port FIFO runs out of credits in the NDN network
0x39	NDN_DIAG_C_OUT_OF_CRED	Number of cycles that core input port FIFO runs out of credits in the NDN network
0x3a	NDN_DIAG_N_EGRESS	Packets sent out from north port in the NDN network
0x3b	NDN_DIAG_S_EGRESS	Packets sent out from south port in the NDN network

Hex Value	Name	Description
0x3c	NDN_DIAG_E_EGRESS	Packets sent out from east port in the NDN network
0x3d	NDN_DIAG_W_EGRESS	Packets sent out from west port in the NDN network
0x3e	NDN_DIAG_C_EGRESS	Packets sent out from core port in the NDN network
0x3f	NDN_DIAG_N_INGRESS	Packets received by north port in the NDN network
0x40	NDN_DIAG_S_INGRESS	Packets received by south port in the NDN network
0x41	NDN_DIAG_E_INGRESS	Packets received by east port in the NDN network
0x42	NDN_DIAG_W_INGRESS	Packets received by west port in the NDN network
0x43	NDN_DIAG_C_INGRESS	Packets received by core port in the NDN network
0x44	NDN_DIAG_CORE_SENT	Packets completed from core port in the NDN network

6.1.3 Programming Counter to Monitor Events

To program a counter to monitor one of the events from the event list, the event name or number needs to be written to the corresponding event file.

Let us call the `/sys/class/hwmon/hwmon<N>` folder corresponding to this driver as `BFPERF_DIR`.

For example, to monitor the event `HNF_REQUESTS` (`0x45`) on `tile2` using counter 3:

```
$ echo 0x45 > <BFPERF_DIR>/tile2/event3
```

Or:

```
$ echo HNF_REQUESTS > <BFPERF_DIR>/tile2/event3
```

Once this is done, `counter3` resets the counter and starts monitoring the number of `HNF_REQUESTS`.

To read the counter value, run:

```
$ cat <BFPERF_DIR>/tile2/counter3
```

To see what event is currently being monitored by a counter, just read the corresponding event file to get the event name and number.

```
$ cat <BFPERF_DIR>/tile2/event3
```

In this case, reading the `event3` file returns "`0x45: HNF_REQUESTS`".

To clear the counter, write 0 to the counter file.

```
$ echo 0 > <BFPERF_DIR>/tile2/counter3
```

This resets the accumulator and the counter continues monitoring the same event that has previously been programmed, but starts the count from 0 again. Writing non-zero values to the counter files is not allowed.

To stop monitoring an event, write `0xff` to the corresponding event file.

This is slightly different for the l3cache blocks due to the restriction that all counters can only be enabled, disabled, or reset together. So once the event is written to the event file, the counters will have to be enabled to start monitoring their respective events by writing "1" to the "enable" file. Writing "0" to this file will stop all the counters. The most reliable way to get accurate counter values would be by disabling the counters after a certain time period and then proceeding to read the counter values.

> Programming a counter to monitor a new event automatically stops all the counters. Also, enabling the counters resets the counters to 0 first.

For blocks that have performance statistics registers (mechanism 2), all of these statistics are directly made available to be read or reset.

For example, to read the number of incoming posted packets to TRIO2:

```
$ cat <BFPERF_DIR>/pcie2/IN_P_PKT_CNT
```

The count can be reset to 0 by writing 0 to the same file. Again, non-zero writes to these files are not allowed.

6.2 Intelligent Platform Management Interface

IPMB requests can be initiated in 2 directions:

- DPU BMC-to-BlueField
- BlueField-to-DPU BMC

> The BlueField `ipmb_dev_int` driver is registered at the 7-bit I^{2}C address 0x30 by default. The I^{2}C address of the BlueField can be changed in the file `/usr /bin/set_emu_param.sh`.
>
> - BlueField Controller cards provide connection from the host server BMC to BlueField Arm I^{2}C bus
> - BlueField DPUs provide connection from the host server BMC to the BlueField NC-SI port
> - BlueField Reference Platforms provide connection from its on-board BMC to BlueField Arm I^{2}C bus

6.2.1 DPU BMC IPMI Commands

The DPU BMC is able to retrieve data from NVIDIA® BlueField® DPU software over its Intelligent Platform Management Bus (IPMB).

The DPU BMC may request information about itself using the following command format:

```
$ ipmitool <ipmitool command>
```

Issue a command with the following format from the DPU BMC to retrieve information from the BlueField:

```
ipmitool -I ipmb <ipmitool command>
```

The following table provides a list of supported ipmitool command arguments:

Command Description	ipmitool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Get device ID	mc info	20.1
Broadcast "Get Device ID"	Part of "mc info"	20.9
Get BMC global enables	mc getenables	22.2
Get device SDR info	sdr info	35.2
Get device SDR	"sdr get", "sdr list" or "sdr elist"	35.3
Get sensor hysteresis	sdr get <sensor-id>	35.7
Set sensor threshold	sensor thresh <sensor-id> <threshold> <setting> • sensor-id - name of the sensor for which a threshold is to be set threshold - which threshold to set • ucr - upper critical • unc - upper non-critical • lnc - lower non-critical • lcr - lower critical • setting - the value to set the threshold to To configure all lower thresholds, use: sensor thresh <sensor-id> lower <lnr> <lcr> <lnc> The lower non-recoverable <lnr> option is not supported To configure all upper thresholds, use: sensor thresh <sensor-id> upper <unc> <ucr> <unr> The upper non-recoverable <unr> option is not supported	35.8
Get sensor threshold	sdr get <sensor-id>	35.9
Get sensor event enable	sdr get <sensor-id>	35.11

Command Description	ipmitool Command	Relevant IPMI 2.0 Rev 1.1 Spec Section
Get sensor reading	sensor reading <sensor-id>	35.14
Get sensor type	sdr type <type>	35.16
Read FRU data	fru read <fru-number> <file-to-write-to>	34.2
Get SDR repository info	sdr info	33.9
Get SEL info	"sel" or "sel info"	40.2
Get SEL allocation info	"sel" or "sel info"	40.3
Get SEL entry	"sel list" or "sel elist"	40.5
Add SEL entry	sel add <filename>	40.6
Delete SEL entry	sel delete <id>	40.8
Clear SEL	sel clear	40.9
Get SEL time	sel time get	40.1
Set SEL time	sel time set "MM/DD/YYYY HH:M:SS"	40.11

6.2.1.1 List of IPMI Supported Sensors

Sensor	ID	Description
`bluefield_temp`	0	Support NIC monitoring of BlueField's temperature
`ddr0_0_temp` [1]	1	Support monitoring of DDR0 temp (on memory controller 0)
`ddr0_1_temp` [1]	2	Support monitoring of DDR1 temp (on memory controller 0)
`ddr1_0_temp` [1]	3	Support monitoring of DDR0 temp (on memory controller 1)
`ddr1_1_temp` [1]	4	Support monitoring of DDR1 temp (on memory controller 1)
`p0_temp`	5	Port 0 temperature
`p1_temp`	6	Port 1 temperature
`p0_link`	7	Port0 link status
`p1_link`	8	Port1 link status

1. On BlueField-2 and BlueField-3 based boards, DDR sensors and FRUs are not supported. They will appear as no reading. ↩ ↩ ↩ ↩

6.2.1.2 List of IPMI Supported FRUs

FRU	ID	Description
`update_timer`	0	`set_emu_param.service` is responsible for collecting data on sensors and FRUs every 3 seconds. This regular update is required for sensors but not for FRUs whose content is less susceptible to change. `update_timer` is used to sample the FRUs every hour instead. Users may need this timer in the case where they are issuing several raw IPMItool FRU read commands. This helps in assessing how much time users have to retrieve large FRU data before the next FRU update. `update_timer` is a hexadecimal number.
`fw_info`	1	NVIDIA® ConnectX® firmware information, Arm firmware version, and MLNX_OFED version. The `fw_info` is in ASCII format.
`nic_pci_dev_info`	2	NIC vendor ID, device ID, subsystem vendor ID, and subsystem device ID. The `nic_pci_dev_info` is in ASCII format.
`cpuinfo`	3	CPU information reported in `lscpu` and `/proc/cpuinfo`. The `cpuinfo` is in ASCII format.
`ddr0_0_spd` [2]	4	FRU for SPD MC0 DIMM 0 (MC = memory controller). The `ddr0_0_spd` is in binary format.
`ddr0_1_spd` [2]	5	FRU for SPD MC0 DIMM1. The `ddr0_1_spd` is in binary format.
`ddr1_0_spd` [2]	6	FRU for SPD MC1 DIMM0. The `ddr1_0_spd` is in binary format.
`ddr1_1_spd` [2]	7	FRU for SPD MC1 DIMM1. The `ddr1_1_spd` is in binary format.
`emmc_info`	8	eMMC size, list of its partitions, and partitions usage (in ASCII format). eMMC CID, CSD, and extended CSD registers (in binary format). The ASCII data is separated from the binary data with "StartBinary" marker.
`qsfp0_eeprom`	9	FRU for QSFP 0 EEPROM page 0 content (256 bytes in binary format)
`qsfp1_eeprom`	10	FRU for QSFP 1 EEPROM page 0 content (256 bytes in binary format)

`ip_addresses`	11	This FRU file can be used to write the BMC port 0 and port 1 IP addresses to the BlueField. It is empty to begin with.
		The file passed through the `ipmitool fru write 11 <file>` command must have the following format:
		`BMC: XXX.XXX.XXX.XXX`
		`P0: XXX.XXX.XXX.XXX`
		`P1: XXX.XXX.XXX.XXX`
		The size of the written file should be exactly 61 bytes.

`dimms_ce_ue`	12	FRU reporting the number of correctable and uncorrectable errors in the DIMMs. This FRU is updated once every 3 seconds.

FRU	ID	Description
`eth0`	13	Network interface 0 information. Updated once every minute.
`eth1`	14	Network interface 1 information. Updated once every minute.
`bf_uid`	15	BlueField UID
`eth_hw_counters`	16	List of ConnectX interface hardware counters

2. On BlueField-2 and BlueField-3 based boards, DDR sensors and FRUs are not supported. They will appear as no reading.

6.2.2 BlueField IPMI Commands

The BlueField is able to retrieve data from the DPU BMC over IPMB.

Issue a command with the following format from the BlueField to retrieve information from the BMC:

```
$ ipmitool <ipmitool command>
```

The BlueField may request information about itself using the following command format:

```
$ ipmitool -U ADMIN -P ADMIN -p 9001 -H localhost <ipmitool command>
```

The `ipmb_host` driver allows the BlueField to send requests to the BMC. Once `set_emu_param.service` is started, it will try to load the `ipmb_host` drivers. If the BMC is down or not responsive when BlueField tries to load the `ipmb_host` driver, the latter will not load successfully. In that case, make sure the BMC is up and operational, and run the following from BlueField's console:

```
echo 0x1011 > /sys/bus/i2c/devices/i2c-2/delete_device
rmmod ipmb_host
```

The `set_emu_param.service` script will try to load the driver again.

6.2.2.1 I^{2}C Addresses for BMC-initiated Requests

Device	I^{2}C Address
BlueField `ipmb_dev_int`	0x30
BMC `ipmb_host`	0x20

6.2.2.2 I^{2}C Addresses for BlueField-initiated Requests

Device	I^{2}C Address
BlueField `ipmb_host`	0x11
BMC `ipmb_dev_int`	0x10

6.2.2.3 Changing I^{2}C Addresses

To use a different BlueField or BMC I^{2}C address, you must make changes to the following files' variables.

Filename Path	Parameter Change

`/usr/bin/set_emu_param.sh`	The `ipmb_dev_int` and `ipmb_host` drivers are registered at the following I^{2}C addresses:
	• `IPMB_DEV_INT_ADD=<BlueField I2C Address 1>`
	• `IPMB_HOST_ADD=<BlueField I2C Address 2>`
	These addresses must be different from one another. Otherwise, one of the drives will fail to register.
	To change the BMC I^{2}C address:
	`IPMB_HOST_CLIENTADDR=<BMC I2C Address>`
	`<I2C Address> must be equal to: 0x1000+<7-bit I2C address>`

6.2.3 External Host IPMI Commands

It is possible for the external host to retrieve data from the BlueField via the IPMI LAN interface (either OOB or ConnectX).

To do that:

1. Set the network interface address properly in `progconf`. For example, if the OOB IP address is 192.168.101.2, edit the `OOB_IP` variable in the `/etc/ipmi/progconf` file as follows:

```
root@localhost:~# cat /etc/ipmi/progconf
SUPPORT_IPMB="NONE"
LOOP_PERIOD=3
BF_FAMILY=$(/usr/bin/bffamily | tr -d '[:space:]')
OOB_IP="192.168.101.2"
```

2. Then reboot or restart the IPMI service as follows:

```
systemctl restart mlx_ipmid
```

3. To get information from the BlueField, issue commands from the external host in the following format:

```
ipmitool -I lanplus -H 192.168.101.2 -U ADMIN -P ADMIN <ipmitool command>
```

6.2.4 Loading and Using IPMI on BlueField Running CentOS

1. Load the BlueField CentOS image:

   The following steps are performed from the BlueField CentOS prompt. The BlueField is running CentOS 7.6 with kernel 5.4. The CentOS installation was done using the CentOS everything ISO image.

   The following drivers need to be loaded on the BlueField running CentOS:
   - `jc42.ko`
   - `ee1004.ko`
   - `at24.ko`
   - `eeprom.ko`
   - `i2c-dev.ko`

   Example of loading `ee1004.ko`, `at24.ko`, and `eeprom.ko`:

   ```
   modprobe ee1004
   modprobe at24
   modprobe eeprom
   ```

   The `i2c-dev` module is built into the kernel 5.4.60 on CentOS 7.6.

2. (Optional) Update the `i2c-mlx` driver if the installed version is older than `i2c-mlx-1.0-0.gab579c6.src.rpm`.
   a. Re-compile `i2c-mlx`. Run:

   ```
   $ yum remove -y kmod-i2c-mlx
   $ modprobe -rv i2c-mlx
   ```

   b. Transfer the `i2c-mlx` RPM from the BlueField software tarball under distro/SRPM onto the Arm. Run:

   ```
   $ rpmbuild --rebuild /root/i2c-mlx-1.0-0.g422740c.src.rpm
   $ yum install -y /root/rpmbuild/RPMS/aarch64/i2c-
   mlx-1.0-0.g422740c_5.4.17_mlnx.9.ga0bea68.aarch64.rpm
   $ ls -l /lib/modules/$(uname -r)/extra/i2c-mlx/i2c-mlx.ko
   ```

   c. Load `i2c-mlx`. Run:

   ```
   $ modprobe i2c-mlx
   ```

3. Install the following packages:

   ```
   $ yum install ipmitool lm_sensors
   ```

   If the above operation fails for ipmitool, run the following to install it:

   ```
   wget http://sourceforge.net/projects/ipmitool/files/ipmitool/1.8.18/ipmitool-1.8.18.tar.gz
   tar -xvzf ipmitool-1.8.18.tar.gz
   cd ipmitool-1.8.18
   ./bootstrap
   ```

```
./configure
make
make install DESTDIR=/tmp/package-ipmitool
```

4. The `i2c-tools` package is also required, but the version contained in the CentOS Yum repository is old and does not work with BlueField. Therefore, please download i2c-tools version 4.1, and then build and install it.

```
# Build i2c-tools from a newer source
wget http://mirrors.edge.kernel.org/pub/software/utils/i2c-tools/i2c-tools-4.1.tar.gz
tar -xvzf i2c-tools-4.1.tar.gz
cd i2c-tools-4.1
make
make install PREFIX=/usr

# create a link to the libraries
ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so
ln -sfn /usr/lib/libi2c.so.0.1.1 /lib64/libi2c.so.0
```

5. Generate an RPM binary from the BlueField's mlx-OpenIPMI-2.0.25 source RPM.
   The following packages might be needed to build the binary RPM depending on which version of CentOS you are using.

```
$ yum install libtool rpm-devel rpmdevtools rpmlint wget ncurses-devel automake
$ rpmbuild --rebuild mlx-OpenIPMI-2.0.25-0.g581ebbb.src.rpm
```

You may obtain this rpm file by means of scp from the server host's Bluefield Distribution folder. For example:

```
$ scp <BF_INST_DIR>/distro/SRPMS/mlx-OpenIPMI-2.0.25-0.g4fdc53d.src.rpm <ip-address>:/
<target_directory>/
```

If there are issues with building the OpenIPMI RPM, verify that the swig package is not installed.

```
$ yum remove -y swig
```

6. Generate a binary RPM from the ipmb-dev-int source RPM and install it. Run:

```
$ rpmbuild --rebuild ipmb-dev-int-1.0-0.g304ea0c.src.rpm
```

7. Generate a binary RPM from the `ipmb-host` source RPM and install it. Run:

```
$ rpmbuild --rebuild ipmb-host-1.0-0.g304ea0c.src.rpm
```

8. Load OpenIPMI, `ipmb-host`, and `ipmb-dev-int` RPM packages. Run:

```
$ yum install -y /root/rpmbuild/RPMS/aarch64/mlx-
OpenIPMI-2.0.25-0.g581ebbb_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-dev-int-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
$ yum install -y /root/rpmbuild/RPMS/aarch64/ipmb-host-1.0-0.g304ea0c_5.4.0_49.el7a.aarch64.aarch64.rpm
```

9. Load the IPMB driver. Run:

```
$ modprobe ipmb-dev-int
```

10. Install and start `rasdaemon` package. Run:

```
yum install rasdaemon
systemctl enable rasdaemon
systemctl start rasdaemon
```

11. Start the IPMI daemon. Run:

```
$ systemctl enable mlx_ipmid
$ systemctl start mlx_ipmid
$ systemctl enable set_emu_param
$ systemctl start set_emu_param
```

6.3 Redfish

Redfish provides a RESTful interface designed to manage IT infrastructure and is implemented using a modern toolchain (HTTP(s)/TLS/JSON).

Redfish supports the operations listed in this section.

6.3.1 BIOS Configuration Schema

The BIOS schema contains properties related to the BIOS attribute registry. The attribute registry describes the system-specific BIOS attributes and actions for changing to BIOS settings. It is likely that a client finds the `@Redfish.Settings` term in this resource, and if it is found, the client makes requests to change BIOS settings by modifying the resource identified by the `@Redfish.Settings` annotation.

URI	`/redfish/v1/Systems/{ComputerSystemId}/Bios`
Schema file	`http://redfish.dmtf.org/schemas/v1/Bios.v1_1_1.json`
Operations	GET; PATCH

Example response:

```
{
    "@Redfish.Settings": {
        "@odata.type": "#Settings.v1_3_5.Settings",
        "SettingsObject": {
            "@odata.id": "/redfish/v1/Systems/Bluefield/Bios/Settings"
        }
    },
    "@odata.id": "/redfish/v1/Systems/Bluefield/Bios",
    "@odata.type": "#Bios.v1_2_0.Bios",
    "Actions": {
        "#Bios.ChangePassword": {
            "target": "/redfish/v1/Systems/Bluefield/Bios/Actions/Bios.ChangePassword"
        },
        "#Bios.ResetBios": {
            "target": "/redfish/v1/Systems/Bluefield/Bios/Actions/Bios.ResetBios"
        }
    },
    "Attributes": {
        "Boot Partition Protection": false,
        "CurrentUefiPassword": "",
        "DateTime": "2024-04-24T19:56:59Z",
        "DefaultPasswordPolicy": true,
        "Disable PCIe": false,
        "Disable SPMI": false,
        "Disable TMFF": false,
        "EmmcWipe": false,
        "Enable 2nd eMMC": false,
        "Enable OP-TEE": false,
        "Enable SMMU": true,
        "Field Mode": false,
        "Host Privilege Level": "Privileged",
        "Internal CPU Model": "Embedded",
        "LegacyPasswordEnable": true,
        "NicMode": "DpuMode",
        "NvmeWipe": false,
        "OsArgs": "",
        "ResetEfiVars": false,
        "SPCR UART": "Disabled",
        "UefiArgs": "",
        "UefiPassword": ""
    },
    "Description": "BIOS Configuration Service",
    "Id": "BIOS",
    "Links": {
        "SoftwareImages": [
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_ATF"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_BOARD"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_BSP"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_NIC"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_NODE"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_OFED"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_OS"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_SYS_IMAGE"
            },
            {
                "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_UEFI"
            }
        ],
        "SoftwareImages@odata.count": 9
    },
    "Name": "BIOS Configuration",
    "ResetBiosToDefaultsPending": false
}
```

The following table explains each of the attributes listed in the code:

Attribute	Description
`Boot Partition Protection`	See description in section "System Configuration"
`CurrentUefiPassword`	See "Set Password" in section "System Configuration"
`DateTime`	See "Set RTC" in section "System Configuration"
`DefaultPasswordPolicy`	See "Password Settings" in section "System Configuration"
`Disable PCIe`	See description in section "System Configuration"
`Disable SPMI`	See description in section "System Configuration"
`Disable TMFF`	See description in section "System Configuration"
`EmmcWipe`	See description in section "System Configuration"
`Enable 2nd eMMC`	See description in section "System Configuration"
`Enable OP-TEE`	See description in section "System Configuration"
`Enable SMMU`	See description in section "System Configuration"
`Field Mode`	See description in section "System Configuration"
`Host Privilege Level`	See "BlueField Modes" in section "System Configuration"
`Internal CPU Model`	See "BlueField Modes" in section "System Configuration"
`LegacyPasswordEnable`	See "Password Settings" in section "System Configuration"
`NicMode`	See "BlueField Modes" under section "System Configuration"

Attribute	Description
`NvmeWipe`	See description in section "System Configuration"
`OsArgs`	Arguments to pass to the OS kernel
`ResetEfiVars`	See "Reset EFI Variables" in section "System Configuration"
`SPCR UART`	See "Select SPCR UART" in section "System Configuration"
`UefiArgs`	Arguments to pass to the UEFI
`UefiPassword`	See "Set Password" in section "System Configuration"

6.3.2 BlueField Platform Inventory

The BlueField Platform provides inventory information in the `ComputerSystemCollection` schema. To identify the DPU `ComputerSystem` instance, fetch the `ComputerSystemCollection` first.

DPUs are identified with the `SystemType` attribute `DPU`. The DPU instance identifier value (`DPU.Embedded.1_NIC.Slot.2` in this case) differs from one server vendor to another but will uniquely identify the DPU in all cases.

The following is a simple example of fetching Redfish inventory information from a server's BMC:

```
root@localhost:~$ python3 /usr/local/bin/redfishtool.py -r <bmc_ip> -u <USER> -p <PASSWORD> raw GET /redfish/v1/
Systems/
{
    "@odata.context": "/redfish/v1/$metadata#ComputerSystemCollection.ComputerSystemCollection",
    "@odata.id": "/redfish/v1/Systems",
    "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
    "Description": "Collection of Computer Systems",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Systems/System.Embedded.1"
        },
        {
            "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2"
        }
    ],
    "Members@odata.count": 2,
    "Name": "Computer System Collection"
}

root@localhost:~$ python3 /usr/local/bin/redfishtool.py -r <bmc_ip> -u <USER> -p <PASSWORD> raw GET /redfish/v1/
Systems/DPU.Embedded.1_NIC.Slot.2
{
    "@odata.context": "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
    "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2",
    "@odata.type": "#ComputerSystem.v1_12_0.ComputerSystem",
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Actions/ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "ForceRestart",
                "Nmi"
            ]
        }
    },
    "Bios": {
        "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Bios"
    },
    "BiosVersion": null,
    "Boot": {
        "BootOptions": {
            "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/BootOptions"
        },
        "BootOrder": [],
        "BootOrder@odata.count": 0,
        "BootSourceOverrideEnabled": null,
        "BootSourceOverrideMode": null,
        "BootSourceOverrideTarget": null,
        "UefiTargetBootSourceOverride": null,
        "BootSourceOverrideTarget@Redfish.AllowableValues": []
    },
    "Description": "DPU System",
    "Id": "DPU.Embedded.1_NIC.Slot.2",
    "Manufacturer": "DELL",
    "Model": "NVIDIA Bluefield-2 25GbE 2p Crypto DPU",
    "Name": "DPU System",


    "Oem": {
        "Dell": {
            "@odata.type": "#DellComputerSystem.v1_1_0.DellComputerSystem",
            "DPUConfig": {
                "FQDD": "DPU.Embedded.1:NIC.Slot.2",
                "BootStatus": "OSBooting",
                "DPUBootSynchronization": "Enabled",
                "DPUTrust": "Enabled",
                "IdenticalSBDF": [
                    "0:23:0:0",
                    "0:23:0:1"
                ],
                "LastResetReason": null,
                "OSName": null,
                "OSReadyTimeout": 20,
                "OSInstallationTimeout": 30,
                "OSVersion": null,
                "OSVendor": null,
                "OSStatus": "Unknown",
                "Slot": "2",
                "PCIeSlotState": "Enabled",
                "PostCode": null,
                "VendorID": "0x15B3",
                "DeviceID": "0xA2D6",
                "SubVendorID": "0x15B3",
                "SubDeviceID": "0x0129"
            },
            "Name": "DPUConfig",
            "Id": "DPU.Embedded.1_NIC.Slot.2"
        }
    },
    "PartNumber": "JNDCMX01",
    "SecureBoot": {
        "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/SecureBoot"
    },
    "SerialNumber": "IL740311A5000A",
    "SKU": "0JNDCM",
    "Status": {
        "Health": "Ok",
        "HealthRollup": "Ok",
        "State": "Enabled"
    },
    "SystemType": "DPU",
    "UUID": "ec6dd921-882a-ec11-8000-08c0eb5180ba",
    "@Redfish.Settings": {
        "@odata.context": "/redfish/v1/$metadata#Settings.Settings",
        "@odata.type": "#Settings.v1_3_3.Settings",
        "SettingsObject": {
            "@odata.id": "/redfish/v1/Systems/DPU.Embedded.1_NIC.Slot.2/Settings"
        }
    }
}
```

6.3.3 Boot Override

This example demonstrates how to boot a BlueField Platform while overriding the existing boot options and using HTTP boot to obtain the image.

Check the current boot override settings by doing a GET on `ComputerSystem` schema. Look for the Boot property.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/ | python3 -m json.tool
{
...
"Boot": {
        "BootNext": "",
        "BootOrderPropertySelection": "BootOrder",
        "BootSourceOverrideEnabled": "Disabled",
        "BootSourceOverrideMode": "UEFI",
        "BootSourceOverrideTarget": "None",
        "UefiTargetBootSourceOverride": "None",
        .....
        },
  ....
  "BootSourceOverrideEnabled@Redfish.AllowableValues": [
            "Once",
            "Continuous",
            "Disabled"
        ],
    "BootSourceOverrideTarget@Redfish.AllowableValues": [
            "None",
            "Pxe",
            "UefiHttp",
            "UefiShell",
            "UefiTarget",
            "UefiBootNext"
        ],
  ....
}
```

The sample output above shows the `BootSourceOverrideEnabled` property is `Disabled` and `BootSourceOverrideTarget` is `None`. The `BootSourceOverrideMode` property should always be set to `UEFI`. Allowable values of `BootSourceOverrideEnabled` and `BootSourceOverrideTarget` are defined in the meta-data `BootSourceOverrideEnabled@Redfish.AllowableValues` and `BootSourceOverrideTarget@Redfish.AllowableValues` respectively.

To perform boot override, you must perform a PATCH to pending settings URI:

```
curl -vk -X PATCH -d '{"Boot": {"BootSourceOverrideEnabled":"Once", "BootSourceOverrideMode":"UEFI",
"BootSourceOverrideTarget": "UefiHttp", "HttpBootUri":"http://<HTTP-Server-Ip>/Image.iso"}}' -u "user:password"
https://<bmc_ip>/redfish/v1/Systems/SystemId/Settings | python3 -m json.tool
```

After performing the above PATCH successfully, reboot the BlueField Platform. Once UEFI has completed, check whether the settings are applied by performing a GET on `ComputerSystem` schema.

Note that the `HttpBootUri` property is parsed by the Redfish server and the URI is presented to the DPU as part of DHCP lease when the DPU performs the HTTP boot.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/ | python3 -m json.tool
{
...
"Boot": {
        "BootNext": "",
        "BootOrderPropertySelection": "BootOrder",
        "BootSourceOverrideEnabled": "Once",
        "BootSourceOverrideMode": "UEFI",
        "BootSourceOverrideTarget": "UefiHttp",
        "UefiTargetBootSourceOverride": "None",
        .....
        },
   .....
}
```

After confirming the settings are applied (see PATCH properties above), reboot the DPU for the settings to take effect. If `BootSourceOverrideEnabled` is set to `Once`, boot override is disabled and any related properties are reset to their former values to avoid repetition. If it is set to `Continuous`, then on every reboot the DPU will keep performing boot override (`HTTPBoot`).

6.3.4 Boot Order

The following is an example of changing the boot order and fetching the details of a boot option.

1. Check the current boot order by doing GET on the `ComputerSystem` schema. Look for the `BootOrder` attribute under the `Boot` property.
2. Get the details of a particular entity in the `BootOrder` array by performing a GET to the respective BootOption URL. For example, to get details of `Boot0006`, run:

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/BootOptions/Boot0006 |
python3 -m json.tool

{
    "@odata.type": "#BootOption.v1_0_3.BootOption",
    "@odata.id": "/redfish/v1/Systems/SystemId/BootOptions/Boot0006",
    "Id": "Boot0006",
    "BootOptionEnabled": true,
    "BootOptionReference": "Boot0006",
    "DisplayName": "UEFI HTTPv6 (MAC:B8CEF6B8A006)",
    "UefiDevicePath":   "PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/
MAC(B8CEF6B8A006,0x1)/
IPv6(0000:0000:0000:0000:0000:0000:0000:0000,0x0,Static,0000:0000:0000:0000:0000:0000:0000:0000,0x40,0000:0
000:0000:0000:0000:0000:0000:0000)/Uri()"
}
```

3. To change the boot order, the entire `BootOrder` array must be PATCHed to the pending settings URI. For the above example of the `BootOrder` array, if you intend to have `Boot0006` at the beginning of the array, then the PATCH operation is as follows.

```
curl -vk -X PATCH -d '{ "Boot": { "BootOrder": [ "Boot0006", "Boot0017", "Boot0001", "Boot0002",
"Boot0003", "Boot0004", "Boot0005", "Boot0007", ] }}' -u "user:password" https://<bmc_ip>/redfish/v1/
Systems/SystemId/Settings | python3 -m json.tool
```

Updating the `BootOrder` array results in a permanent boot order change (persistent across reboots).

After a successful PATCH, reboot the DPU and check if the settings were applied by doing a GET on the `ComputerSystem` schema. If the `BootOrder` array is updated as intended, then the settings were applied and the BlueField Platform should boot as per the order in proceeding cycles.

6.3.5 BIOS Attributes

The following is an example of fetching and setting a DPU BIOS attribute.

1. Check UEFI attributes and their values by doing a GET on Bios URL. Look for `Attributes` property.

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios | python3 -m json.tool

{
    ....
    "Attributes": {
        "Boot Partition Protection": false,
        "CurrentUefiPassword": "",
        "DateTime": "2022-07-05T16:02:12Z",
        "Disable PCIe": false,
        "Disable SPMI": false,
        "Disable TMFF": false,
        "Enable 2nd eMMC": false,
        "Enable OP-TEE": false,
        "Enable SMMU": true,
        "Field Mode": false,
        "Host Privilege Level": "Privileged",
        "Internal CPU Model": "Embedded",
        "ResetEfiVars": false,
        "SPCR UART": "Disabled",
        "UefiPassword": ""
    },
   ....
}
```

For Security reasons, `CurrentUefiPassword` and `UefiPassword` strings might be empty.

2. The following example updates the UEFI password. Perform PATCH to Bios pending settings URI as follows:

```
curl -vk -X PATCH -d '{"Attributes":{"CurrentUefiPassword": "CURRENTPASSWD", "UefiPassword":
"NEWPASSWORD"}}' -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios/Settings | python3 -m
json.tool
```

To update the password, both the current password and the new password (requesting) should be specified as demonstrated above. Otherwise, the change does not work. To modify other attributes no password is required.

3. To confirm whether the PATCH request is successful, perform a GET to the BIOS pending settings URI:

```
curl -vk -X GET -u "user:password" https://<bmc_ip>/redfish/v1/Systems/SystemId/Bios/Settings | python3 -m
json.tool
```

4. For requests to take effect, reboot the DPU. If the `CurrentUefiPassword` is correct, then the UEFI password is updated during the UEFI Redfish phase of boot.

The UEFI password is only required to enter the UEFI menu using the serial console.

6.4 Logging

6.4.1 RShim Logging

RShim logging uses an internal 1KB HW buffer to track booting progress and record important messages. It is written by the NVIDIA® BlueField® Arm cores and is displayed by the RShim driver from the USB/PCIe host machine. Starting in release 2.5.0, ATF has been enhanced to support the RShim logging.

The RShim log messages can be displayed described in the following:

1. Check the `DISPLAY_LEVEL` level in file `/dev/rshim0/misc`.

```
# cat /dev/rshim0/misc
DISPLAY_LEVEL   0 (0:basic, 1:advanced, 2:log)
…
```

2. Set `DISPLAY_LEVEL` to 2.

```
# echo "DISPLAY_LEVEL 2" > /dev/rshim0/misc
```

3. Log messages are displayed in the misc file.
   The following is an example output for BlueField-2:

```
# cat /dev/rshim0/misc
...
---------------------------------------
    Log Messages
---------------------------------------
 INFO[BL2]: start
 INFO[BL2]: no DDR on MSS0
 INFO[BL2]: calc DDR freq (clk_ref 53836948)
 INFO[BL2]: DDR POST passed
 INFO[BL2]: UEFI loaded
 INFO[BL31]: start
 INFO[BL31]: runtime
 INFO[UEFI]: eMMC init
 INFO[UEFI]: eMMC probed
 INFO[UEFI]: PCIe enum start
 INFO[UEFI]: PCIe enum end
```

The following table details the ATF/UEFI messages for BlueField-2 and BlueField-3:

Message	Explanation	Action
`INFO[BL2]: start`	BL2 started	Informational
`INFO[BL2]: no DDR on MSS<N>`	DDR is not detected on memory controller <N>	Informational (depends on device)
`INFO[BL2]: calc DDR freq (clk_ref 156M, clk xxx)`	DDR frequency is calculated based on reference clock 156M	Informational
`INFO[BL2]: calc DDR freq (clk_ref 100M, clk xxx)`	DDR frequency is calculated based on reference clock 100M	Informational
`INFO[BL2]: calc DDR freq (clk_ref xxxx)`	DDR frequency is calculated based on reference clock xxxx	Informational
`INFO[BL2]: DDR POST passed`	BL2 DDR training passed	Informational
`INFO[BL2]: UEFI loaded`	UEFI image is loaded successfully in BL2	Informational
`ERR[BL2]: DDR init fail on MSS<N>`	DDR initialization failed on memory controller <N>	Informational (depends on device)
`ERR[BL2]: image <N> bad CRC`	Image with ID <N> is corrupted which will cause hang	Error message. Reset the device and retry. If problem persists, use a different image to retry it.
`ERR[BL2]: DDR BIST failed`	DDR BIST failed	Need to retry. Check the ATF booting message whether the detected OPN is correct or not, or whether it is supported by this image. If still fails, contact NVIDIA Support.
`ERR[BL2]: DDR BIST Zero Mem failed`	DDR BIST failed in the zero-memory operation	Power-cycle and retry. If the problem persists, contact your NVIDIA FAE.
`WARN[BL2]: DDR frequency unsupported`	DDR training is programmed with unsupported parameters	Check whether official FW is being used. If the problem persists, contact your NVIDIA FAE.
`WARN[BL2]: DDR min-sys(unknown)`	System type cannot be determined and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.
`WARN[BL2]: DDR min-sys(misconf)`	System type misconfigured and boot as a minimal system	Check whether the OPN or PSID is supported. If the problem persists, contact your NVIDIA FAE.

Message	Explanation	Action
`Exception(BL2): syndrome = xxxxxxxx …`	Exception in BL2 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
`PANIC(BL2): PC = xxx …`	Panic in BL2 with register dump. System will hung.	Capture the log, analyze the cause, and report to FAE if needed
`ERR[BL2]: load/auth failed`	Failed to load image (non-existent/corrupted), or image authentication failed when secure boot is enabled	Try again with the correct and properly signed image
`INFO[BL31]: start`	BL31 started	Informational
`INFO[BL31]: runtime`	BL31 enters the runtime state. This is the latest BL31 message in normal booting process.	Informational
`Exception(BL31): syndrome = xxxxxxxx cptr_el3 xx daif xx …`	Exception in BL31 with syndrome code and register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
`PANIC(BL31): PC = xxx cptr_el3 xxx daif xxx …`	Panic in BL31 with register dump. System hung.	Capture the log, analyze the cause, and report to FAE if needed
`INFO[UEFI]: eMMC init`	eMMC driver is initialized	Informational and should always be printed
`INFO[UEFI]: eMMC probed`	eMMC card is initialized	Informational and should always be printed
`ASSERT(UEFI): xxx : line-no`	Runtime assert message in UEFI	Contact your NVIDIA FAE with this information. Usually the system is able to continue running.
`INFO[UEFI]: PCIe enum start`	PCIe enumeration start	Informational
`INFO[UEFI]: PCIe enum end`	PCIe enumeration end	Informational
`ERR[UEFI]: Synchronous Exception at xxxxxx ERR[UEFI]: PC=xxxxxx ERR[UEFI]: PC=xxxxxx …`	UEFI Exception with PC value reported	Contact your NVIDIA FAE with this information
`ERR[BL2]: FW auth failed`	Image authentication error	Wrong image has been used in the current secure lifecycle. Switch to the correct image.
`ERR[BL2]: IROT cert sig not found`	Failed to load attestation certificates	Contact your NVIDIA FAE with this information
`ERR[BL2]: IROT cert sig not found`	Failed to load certification update record. Only relevant for certain BlueField-3 DPUs.	Contact your NVIDIA FAE with this information

Message	Explanation	Action
`INFO[BL31]: PSC Turtle Mode detected`	PSC enters turtle mode BlueField-3 only.	Informational
`INFO[BL31]: In Enhanced NIC mode`	BlueField-3 enters enhanced NIC mode	Informational
`ERR[BL31]: (set_page err | pmbus_lsb err | mfr_vr_mc err | set_vout err)`	BlueField-3 power management programming error. Usually happens when the I2C voltage regulator is not accessible.	Contact your NVIDIA FAE with this information
`INFO [BL31]: MB8: VDD adjustment complete`	BlueField-3 MainBin 8-core board VDD CPU adjustment	Informational
`INFO [BL31]: VDD adjustment complete`	BlueField-3 (non-8-core board) VDD CPU adjustment	Informational
`INFO [BL31]: VDD: xxx mV`	BlueField-3 VDD CPU voltage	Informational
`ERR[BL31]: cannot access vr0 (or access vr1)`	BlueField-3 unable to access voltage regulator (vr0 or vr1) via I2C	Contact your NVIDIA FAE with this information
`ERR[BL31]: ATX power not detected!`	ATX power is not connected	Contact your NVIDIA FAE with this information
`INFO[BL31]: PTMERROR: Unknown OPN`	Unable to detect the OPN on this device	Contact your NVIDIA FAE with this information
`INFO[BL31]: PTMERROR: VR access error`	Unable to access the voltage regulator on this device This also means power capping will be disabled.	Contact your NVIDIA FAE with this information
`INFO[BL31]: power capping disabled`	BlueField-3 power capping disabled	Informational
`INFO[BL2]: boot mode (rshim | emmc | unknown)`	Device boot mode (from external RShim or eMMC)	Informational
`ERR[BL31]: ECC_SINGLE_ERROR_CNT=xxx`	Single ECC error counter report	Contact your NVIDIA FAE with this information
`ERR[BL31]: ECC_DOUBLE_ERROR_CNT=xxx`	Double ECC error counter report	Contact your NVIDIA FAE with this information
`ERR[BL31]: mss0|mss1: C0|C1 single-bit ecc, IRQ[%d]`	MSS (0 or 1) channel (0 or 1) single-bit ECC error interrupt #	Contact your NVIDIA FAE with this information

Message	Explanation	Action
`ERR[BL31]: mss0|mss1: C0|C1 Double bit ecc, IRQ[%d]`	MSS (0 or 1) channel (0 or 1) double-bit ECC error interrupt #	Contact your NVIDIA FAE with this information
`ERR[BL31]: Double-bit ECC also detected in same buffer`	Single/double ECC error detected in the same buffer	Contact your NVIDIA FAE with this information
`ERR[BL31]: l3c: double-bit ecc`	L3c double-bit ECC error detected	Contact your NVIDIA FAE with this information
`ERR[BL31]: MSS%d DIMM%d single|double bit ECC error detected`	MSS DRAM single (or double) bit error detected	Contact your NVIDIA FAE with this information
`ERR[BL31]: MSS%d SRAM double bit ECC error detected`	MSS SRAM double bit ECC error detected	Contact your NVIDIA FAE with this information

6.4.2 IPMI Logging in UEFI

During UEFI boot, the BlueField sends IPMI SEL messages over IPMB to the BMC in order to track boot progress and report errors. The BMC must be in responder mode to receive the log messages.

6.4.2.1 SEL Record Format

The following table presents standard SEL records (record type = 0x02).

Byte(s)	Field	Description
1 2	Record ID	ID used to access SEL record. Filled in by the BMC. Is initialized to zero when coming from UEFI.
3	Record Type	Record type
4 5 6 7	Timestamp	Time when event was logged. Filled in by BMC. Is initialized to zero when coming from UEFI.
8 9	Generator ID	This value is always 0x0001 when coming from UEFI
10	EvM Rev	Event message format revision which provides the version of the standard a record is using. This value is 0x04 for all records generated by UEFI.
11	Sensor Type	Sensor type code for sensor that generated the event
12	Sensor Number	Number of the sensor that generated the event. These numbers are arbitrarily chosen by the OEM.
13	Event Dir | Event Type	[7] - 0b0 = Assertion, 0b1 = Deassertion [6:0] - Event type code

Byte(s)	Field	Description
14	Event Data 1	[7:6] - Type of data in Event Data 2 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension [5:4] - Type of data in Event Data 3 • 0b00 = unspecified • 0b10 = OEM code • 0b11 = Standard sensor-specific event extension [3:0] - Event Offset; offers more detailed event categories. See *IPMI 2.0 Specification* section 29.7 for more detail.
15	Event Data 2	Data attached to the event. 0xFF for unspecified. Under some circumstances, this may be used to specify more detailed event categories.
16	Event Data 3	Data attached to the event. 0xFF for unspecified.

See *IPMI 2.0 Specification* section 32.1 for more detail.

6.4.2.2 Possible SEL Field Values

BlueField UEFI implements a subset of the IPMI 2.0 SEL standard. Each field may have the following values:

Field	Possible Values	Description of Values
Record Type	0x02	Standard SEL record. All events sent by UEFI are standard SEL records.
Event Dir	0b0	All events sent by UEFI are assertion events
Event Type	0x6F	Sensor-specific discrete events. Events with this type do not deviate from the standard.
Sensor Number	0x06	UEFI boot progress “sensor”. If value is 0x06, the sensor type will always be “System Firmware Progress” (0x0F).

For Sensor Type, Event Offset, and Event Data 1-3 definitions, see next table.

6.4.2.3 Event Definitions

Events are defined by a combination of Record Type, Event Type, Sensor Type, Event Offset (occupies Event Data 1), and sometimes Event Data 2 (referred to as the Event Extension if it defines sub-events).

The following tables list all currently implemented IPMI events (with Record Type = 0x02, Event Type = 0x6F).

Note that if an Event Data 2 or Event Data 3 value is not specified, it can be assumed to be Unspecified (0xFF).

Sensor Type	Sensor Type Code	Event Offset	Event Description, Actions to Take
System Firmware Progress	0x0F	0x00	System firmware error (POST error). Event Data 2: • 0x06 – Unrecoverable EMMC error. Contact NVIDIA support.
		0x02	System firmware progress: Informational message, no actions needed. Event Data 2: • 0x02 – Hard Disk Initialization. Logged when EMMC is initialized. • 0x04 – User Authentication. Logged when a user enters the correct UEFI password. This event is never logged if there is no UEFI password. • 0x07 – PCI Resource Configuration. Logged when PCI enumeration has started. • 0x0B – SMBus Initialization. This event is logged as soon as IPMB is configured in UEFI. • 0x13 – Starting OS Boot Process. Logged when Linux begins booting.

6.4.2.4 Reading IPMI SEL Log Messages

Log messages may be read from the BMC by issuing it a “Get SEL Entry Command” while it is in responder mode, either from a remote host, or from the BlueField DPU itself once it is booted.

```
$ ipmitool sel list
  7b | Pre-Init |0000691604| System Firmwares #0x06 | SMBus initialization | Asserted
  7c | Pre-Init |0000691604| System Firmwares #0x06 | Hard-disk initialization | Asserted
  7d | Pre-Init |0000691654| System Firmwares #0x06 | System boot initiated
$ ipmitool sel get 0x7d
SEL Record ID          : 007d
 Record Type           : 02
 Timestamp             : 01/09/1970 00:07:34
 Generator ID          : 0001
 EvM Revision          : 04
 Sensor Type           : System Firmwares
 Sensor Number         : 06
 Event Type            : Sensor-specific Discrete
 Event Direction       : Assertion Event
 Event Data            : c213ff
 Description           : System boot initiated
$ ipmitool sel clear
Clearing SEL.  Please allow a few seconds to erase.
$ ipmitool sel list
SEL has no entries
```

6.4.3 ACPI BERT Logging

ACPI boot error record table (BERT) is supported to log `last boot error` in Linux. Once Linux `printk` is enabled (e.g., by adding "`kernel.printk=8`" to `/etc/sysctl.conf`), it will try to report the errors automatically for last boot. The following is an example of such error reports:

```
[    2.635539] BERT: Error records from previous boot:
[    2.640434] [Hardware Error]: event severity: fatal
[    2.645331] [Hardware Error]:  Error 0, type: fatal
[    2.650236] [Hardware Error]:   section type: unknown, c6adf9e6-1108-4760-8827-003d059fe2e1
[    2.658606] [Hardware Error]:   section length: 0x35
[    2.663580] [Hardware Error]:   00000000: 52524520 4645555b 203a5d49 0a0d0a0d   ERR[UEFI]: ....
[    2.672284] [Hardware Error]:   00000010: 636e7953 6e6f7268 2073756f 65637845  Synchronous Exce
[    2.680987] [Hardware Error]:   00000020: 6f697470 7461206e 36783020 37313643  ption at 0x6C617
[    2.689696] [Hardware Error]:   00000030: 34 37 30 0d 0a
...
```

6.5 SoC Management Interface

The SoC management interface, formerly known as RShim, allows an external agent such as the host CPU or BMC to operate the DPU and monitor its operational state. This interface allows provisioning of the DPU, resetting Arm cores, and obtaining logs.

For instructions for Windows support, please refer to page "Windows Support".

6.5.1 Installation and Upgrade

Please refer to section Updating Repo Package on Host Side.

6.5.1.1 Configuration File

The configuration file for the SoC management interface is located at `/etc/rshim.conf` and includes the parameters listed in the table below.

Parameter	Default	Description
`BOOT_TIMEOUT`	150	Timeout value in seconds when pushing BFB while Arm side is not reading the boot stream.
`DROP_MODE`	0	Once set to 1, the RShim driver ignores all RShim writes and returns 0 for RShim read. This is used in cases such as during `FW_RESET` or bypassing the RShim PF to VM.
`PCIE_RESET_DELAY`	10	Delay in seconds for RShim over PCIe, which is added after chip reset and before pushing the boot stream.
`PCIE_INTR_POLL_INTERVAL`	10	Interrupt polling interval in seconds when running RShim over direct memory mapping.
`PCIE_HAS_VFIO`	1	Setting this parameter to 0 disallows RShim memory mapping via VFIO.
`PCIE_HAS_UIO`	1	Setting this parameter to 0 disallows RShim memory mapping via UIO.

Configuring RShim is optional. The default parameters are designed to support out-of-box deployment scenarios including multiple DPUs on a single host.

Users may control which RShim index maps to which device by following this procedure:

```
# Uncomment the 'rshim<N>' line to configure the mapping.
#
# device-name pci-device
rshim0      pcie-0000:21:00.2
rshim1      pcie-0000:81:00.2

#
# Ignored devices.
# Uncomment the 'none' line to configure the ignored devices.
#
#none         usb-1-1.4
#none         pcie-lf-0000:84:00.0
```

If any of these configurations are changed, then the SoC management interface must be restarted by running:

```
systemctl restart rshim
```

6.5.2 Host-side Interface Configuration

The NVIDIA® BlueField® DPU registers on the host OS a "DMA controller" for DPU management over PCIe. This can be verified by running the following:

```
#  lspci -d 15b3: | grep 'SoC Management Interface'
27:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

A special SoC management driver must be installed and run on the host OS to expose the various BlueField management interfaces to the OS. Currently, this driver is named RShim and is automatically installed as part of the DOCA installation. Refer to section "Install RShim on Host" for information on how to obtain and install the host-side SoC management interface driver.

When the SoC management interface driver runs properly on the host side, a sysfs device, `/dev/rshim0/*`, and a virtual Ethernet interface, `tmfifo_net0`, become available. The following is an example for querying the status of the SoC management interface driver on the host side:

```
# systemctl status rshim
● rshim.service - rshim driver for BlueField SoC
     Loaded: loaded (/lib/systemd/system/rshim.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-05-31 14:57:07 IDT; 1 day 1h ago
       Docs: man:rshim(8)
    Process: 90322 ExecStart=/usr/sbin/rshim $OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 90323 (rshim)
      Tasks: 11 (limit: 76853)
     Memory: 3.3M
     CGroup: /system.slice/rshim.service
             └─90323 /usr/sbin/rshim
May 31 14:57:07 …  systemd[1]: Starting rshim driver for BlueField SoC...
May 31 14:57:07  … systemd[1]: Started rshim driver for BlueField SoC.
May 31 14:57:07  … rshim[90323]: Probing pcie-0000:a3:00.2(vfio)
May 31 14:57:07  … rshim[90323]: Create rshim pcie-0000:a3:00.2
May 31 14:57:07  … rshim[90323]: rshim pcie-0000:a3:00.2 enable
May 31 14:57:08  … rshim[90323]: rshim0 attached
```

If the SoC management interface driver device does not appear, refer to section "RShim Troubleshooting and How-Tos".

6.5.2.1 Virtual Ethernet Interface

On the host, the SoC management interface driver exposes a virtual Ethernet device called `tmfifo_net0`. This virtual Ethernet can be thought of as a peer-to-peer tunnel connection between the host and the DPU OS. The DPU OS also configures a similar device. The DPU OS's BFB images are customized to configure the DPU side of this connection with a preset IP of 192.168.100.2/30. It is up to the user to configure the host side of this connection. Configuration procedures vary for different OSs.

The following example configures the host side of `tmfifo_net0` with a static IP and enables IPv4-based communication to the DPU OS:

```
#  ip addr add dev tmfifo_net0 192.168.100.1/30
```

For instructions on persistent IP configuration of the tmfifo_net0 interface, refer to step "Assign a static IP to tmfifo_net0" under "Updating Repo Package on Host Side".

Logging in from the host to the DPU OS is now possible over the virtual Ethernet. For example:

```
ssh ubuntu@192.168.100.2
```

6.5.2.2 SoC Management Interface Driver Support for Multiple DPUs

Multiple DPUs may connect to the same host machine. When the SoC management interface driver is loaded and operating correctly, each BlueField device is expected to have its own device directory on sysfs, `/dev/rshim<N>`, and a virtual Ethernet device, `tmfifo_net<N>`.

Important!

`<N>` correlates to the number of BlueField DPUs used where the SoC management interfaces of the first DPU is 0, incrementing by 1 for each added BlueField.

The following are some guidelines on how to set up the SoC management virtual Ethernet interfaces properly if multiple DPUs are installed in the host system.

There are two methods to manage multiple `tmfifo_net` interfaces on a Linux platform:

- Using a bridge, with all `tmfifo_net<N>` interfaces on the bridge - the bridge device bares a single IP address on the host while each DPU has unique IP in the same subnet as the bridge
- Directly over the individual `tmfifo_net<N>` - each interface has a unique subnet IP and each DPU has a corresponding IP per subnet

Whichever method is selected, the host-side `tmfifo_net` interfaces should have different MAC addresses, which can be:

- Configured using `ifconfig`. For example:

  ```
  $ ifconfig tmfifo_net0 192.168.100.1/24 hw ether 02:02:02:02:02:02
  ```

- Or saved in configuration via the `/udev/rules` as can be seen later in this section.

In addition, each Arm-side `tmfifo_net` interface must have a unique MAC and IP address configuration, as BlueField OS comes uniformly pre-configured with a generic MAC, and 192.168.100.2. The latter must be configured in each DPU manually or by DPU customization scripts during BlueField OS installation.

6.5.2.2.1 Multi-board Management Example

This example deals with two BlueField DPUs installed on the same server (the process is similar for more DPUs). The example assumes that the RShim package has been installed on the host server.

6.5.2.2.1.1 Configuring Management Interface on Host

This example is relevant for CentOS/RHEL operating systems only.

1. Create a `bf_tmfifo` interface under `/etc/sysconfig/network-scripts`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-br_tmfifo
```

2. Inside `ifcfg-br_tmfifo`, insert the following content:

```
DEVICE="br_tmfifo"
BOOTPROTO="static"
IPADDR="192.168.100.1"
NETMASK="255.255.255.0"
ONBOOT="yes"
TYPE="Bridge"
```

3. Create a configuration file for the first BlueField DPU, `tmfifo_net0`. Run:

```
vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

4. Inside `ifcfg-tmfifo_net0`, insert the following content:

```
DEVICE=tmfifo_net0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

5. Create a configuration file for the second BlueField DPU, `tmfifo_net1`. Run:

```
DEVICE=tmfifo_net1
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=br_tmfifo
```

6. Create the rules for the `tmfifo_net` interfaces. Run:

```
vim /etc/udev/rules.d/91-tmfifo_net.rules
```

7. Restart the network for the changes to take effect. Run:

```
# /etc/init.d/network restart
Restarting network (via systemctl):            [  OK  ]
```

6.5.2.2.1.2 Configuring BlueField DPU Side

BlueField DPUs arrive with the following factory default configurations for tmfifo_net0.

Address	Value
MAC	`00:1a:ca:ff:ff:01`
IP	`192.168.100.2`

Therefore, if you are working with more than one DPU, you must change the default MAC and IP addresses.

Updating RShim Network MAC Address

This procedure is relevant for Ubuntu/Debian (`sudo` needed), and CentOS BFBs. The procedure only affects the `tmfifo_net0` on the Arm side.

1. Use a Linux console application (e.g. screen or minicom) to log into each BlueField. For example:

```
# sudo screen /dev/rshim<0|1>/console 115200
```

2. Create a configuration file for `tmfifo_net0` MAC address. Run:

```
# sudo vi /etc/bf.cfg
```

3. Inside `bf.cfg`, insert the new MAC:

```
NET_RSHIM_MAC=00:1a:ca:ff:ff:03
```

4. Apply the new MAC address. Run:

```
sudo bfcfg
```

5. Repeat this procedure for the second BlueField DPU (using a different MAC address).

   Arm must be rebooted for this configuration to take effect. It is recommended to update the IP address before you do that to avoid unnecessary reboots.

For comprehensive list of the supported parameters to customize `bf.cfg` during BFB installation, refer to section "bf.cfg Parameters".

Updating IP Address

For Ubuntu:

1. Access the file `50-cloud-init.yaml` and modify the tmfifo_net0 IP address:

```
sudo vim /etc/netplan/50-cloud-init.yaml

                tmfifo_net0:
                     addresses:
                     - 192.168.100.2/30     ===>>>     192.168.100.3/30
```

2. Reboot the Arm. Run:

```
sudo reboot
```

3. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

For CentOS:

1. Access the file `ifcfg-tmfifo_net0`. Run:

```
# vim /etc/sysconfig/network-scripts/ifcfg-tmfifo_net0
```

2. Modify the value for `IPADDR`:

```
IPADDR=192.168.100.3
```

3. Reboot the Arm. Run:

```
reboot
```

   Or perform `netplan apply`.
4. Repeat this procedure for the second BlueField DPU (using a different IP address).

> Arm must be rebooted for this configuration to take effect. It is recommended to update the MAC address before you do that to avoid unnecessary reboots.

6.5.2.3 Permanently Changing Arm-side MAC Address

> It is assumed that the commands in this section are executed with root (or `sudo`) permission.

The default MAC address is `00:1a:ca:ff:ff:01`. It can be changed using `ifconfig` or by updating the UEFI variable as follows:

1. Log into Linux from the Arm console.
2. Run:

```
$ "ls /sys/firmware/efi/efivars".
```

3. If not mounted, run:

```
$ mount -t efivarfs none /sys/firmware/efi/efivars
$ chattr -i /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
$ printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > \
  /sys/firmware/efi/efivars/RshimMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
```

The `printf` command sets the MAC address to `00:1a:ca:ff:ff:03` (the last six bytes of the `printf` value). Either reboot the device or reload the tmfifo driver for the change to take effect.

The MAC address can also be updated from the server host side while the Arm-side Linux is running:

1. Enable the configuration. Run:

   ```
   # echo "DISPLAY_LEVEL 1" > /dev/rshim0/misc
   ```

2. Display the current setting. Run:

   ```
   # cat /dev/rshim0/misc
   DISPLAY_LEVEL   1 (0:basic, 1:advanced, 2:log)
   BOOT_MODE       1 (0:rshim, 1:emmc, 2:emmc-boot-swap)
   BOOT_TIMEOUT    300 (seconds)
   DROP_MODE       0 (0:normal, 1:drop)
   SW_RESET        0 (1: reset)
   DEV_NAME        pcie-0000:04:00.2
   DEV_INFO        BlueField-2(Rev 1)
   PEER_MAC        00:1a:ca:ff:ff:01 (rw)
   PXE_ID          0x00000000 (rw)
   VLAN_ID         0 0 (rw)
   ```

3. Modify the MAC address. Run:

   ```
   $ echo "PEER_MAC  xx:xx:xx:xx:xx:xx" > /dev/rshim0/misc
   ```

For more information and an example of the script that covers multiple DPU installation and configuration, refer to section "Installing Full DOCA Image on Multiple DPUs" of the *NVIDIA DOCA Installation Guide*.

6.5.3 SoC Management Interface Features and Functionality

	Function	Command	Comments
1	Push BFB	`bfb-install -r rshim<N> -b <bfb> [-c bf.cfg]`	Using `bf.cfg` in the command is optional. For more details about `bf.cfg`, refer to section "DPU Configuration File".

2	Open console	`screen /dev/rshim<N>/console 115200`
		`minicom -D /dev/rshim<N>/console`
			The `N` index depends on the number of DPUs in your setup. Use Linux's screen or minicom console applications to access the BlueField console.

3	Configure a virtual network interface	`ip addr add dev tmfifo_net<N> 192.168.100.1/30`	The `N` index depends on the number of DPUs in your setup. Refer to section "SoC Management Interface Driver Support for Multiple DPUs" for more information. The default IP address for the DPU is 192.168.100.2/30. The IP used in the command (192.168.100.1/30) is for example purposes only.

	Function	Command	Comments
4	Log into the DPU	`ssh -6 user@fe80::21a:caff:feff:ff01%tmfifo_net<N>`	The `N` index depends on the number of DPUs in your setup. Refer to section "SoC Management Interface Driver Support for Multiple DPUs" for more information.
5	PXE boot over RShim	N/A	Please refer to section "Deploying BlueField Software Using BFB with PXE" for more information.
6	Issue Arm software reset	`echo "SW_RESERT 1" > /dev/rshim<N>/misc`	
7	Expose log messages	N/A	For more information, please refer to section "Logging".

6.5.4 DPU Configuration File

The `bf.cfg` file contains configuration that can be pushed to customize the installation of the BFB.

Please see section "bf.cfg Parameters" for the `bf.cfg` file contents.

6.6 BlueField OOB Ethernet Interface

The BlueField OOB interface is a gigabit Ethernet interface which provides TCP/IP network connectivity to the Arm cores. This interface is named `oob_net0` and is intended to be used for management traffic (e.g., file transfer protocols, SSH, etc). The Linux driver that controls this interface is named `mlxbf_gige.ko`, and is automatically loaded upon boot. This interface can be configured and monitored using of standard tools (e.g., ifconfig, ethtool, etc). The OOB interface is subject to the following design limitations:

- Only supports 1Gb/s full-duplex setting
- Only supports GMII access to external PHY device
- Supports maximum packet size of 2KB (i.e., no support for jumbo frames)

The OOB interface can also be used for PXE boot. This OOB port is not a path for the BlueField boot stream. Any attempt to push a BFB to this port would not work. Refer to "How to use the UEFI boot menu" for more information about UEFI operations related to the OOB interface.

6.6.1 OOB Interface MAC Address

The MAC address to be used for the OOB port is burned into Arm-accessible UPVS EEPROM during the manufacturing process. This EEPROM device is different from the SPI Flash storage device used for the NIC firmware and associated NIC MACs/GUIDs. The value of the OOB MAC address is specific to each platform and is visible on the board-level sticker.

It is not recommended to reconfigure the MAC address from the MAC configured during manufacturing.

If there is a need to re-configure this MAC for any reason, follow these steps to configure a UEFI variable to hold new value for OOB MAC.:

The creation of an OOB MAC address UEFI variable will override the OOB MAC address defined in EEPROM, but the change can be reverted.

1. Log into Linux from the Arm console.
2. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

   ```
   mount -t efivarfs none /sys/firmware/efi/efivars
   ```
3. Run:

   ```
   chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```
4. Set the MAC address to 00:1a:ca:ff:ff:03 (the last six bytes of the printf value).

   ```
   printf "\x07\x00\x00\x00\x00\x1a\xca\xff\xff\x03" > /sys/firmware/efi/efivars/
   OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```
5. Reboot the device for the change to take effect.

To revert this change and go back to using the MAC as programmed during manufacturing, follow these steps:

1. Log into UEFI from the Arm console, go to "Boot Manager" then "EFI Internal Shell".
2. Delete the OOB MAC UEFI variable. Run:

   ```
   dmpstore -d OobMacAddr
   ```
3. Reboot the device by running "reset" from UEFI.
4. Log into Linux from the Arm console.
5. Issue the command `ls /sys/firmware/efi/efivars` to show whether efivarfs is mounted. If it is not mounted, run:

   ```
   mount -t efivarfs none /sys/firmware/efi/efivars
   ```
6. Run:

   ```
   chattr -i /sys/firmware/efi/efivars/OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```
7. Reconfigure the original MAC address burned by the manufacturer in the format `aa\bb\cc\dd\ee\ff`. Run:

   ```
   printf "\x07\x00\x00\x00\x00\<original-MAC-address>" > /sys/firmware/efi/efivars/
   OobMacAddr-8be4df61-93ca-11d2-aa0d-00e098032b8c
   ```
8. Reboot the device for the change to take effect.

6.6.2 Supported ethtool Options for OOB Interface

The Linux driver for the OOB port supports the handling of some basic ethtool requests: get driver info, get/set ring parameters, get registers, and get statistics.

To use the ethtool options available, use the following format:

```
$ ethtool [<option>] <interface>
```

Where `<option>` may be:

- `<no-argument>` - display interface link information
- `-i` - display driver general information
- `-S` - display driver statistics
- `-d` - dump driver register set
- `-g` - display driver ring information
- `-G` - configure driver ring(s)
- `-k` - display driver offload information
- `-a` - query the specified Ethernet device for pause parameter information
- `-r` - restart auto-negotiation on the specified Ethernet device if auto-negotiation is enabled

For example:

```
$ ethtool oob_net0
Settings for oob_net0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  1000baseT/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  1000baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 3
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: Unknown
        Link detected: yes
```

```
$ ethtool -i oob_net0
driver: mlxbf_gige
version:
firmware-version:
expansion-rom-version:
bus-info: MLNXBF17:00
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: yes
supports-priv-flags: no
```

```
# Display statistics specific to BlueField-2 design (i.e. statistics that are not shown in the output of "ifconfig
oob0_net")
$ ethtool -S oob_net0
NIC statistics:
    hw_access_errors: 0
    tx_invalid_checksums: 0
    tx_small_frames: 1
    tx_index_errors: 0
    sw_config_errors: 0
    sw_access_errors: 0
    rx_truncate_errors: 0
    rx_mac_errors: 0
    rx_din_dropped_pkts: 0
    tx_fifo_full: 0
    rx_filter_passed_pkts: 5549
    rx_filter_discard_pkts: 4
```

6.6.3 IP Address Configuration for OOB Interface

The files that control IP interface configuration are specific to the Linux distribution. The udev rules file (`/etc/udev/rules.d/92-oob_net.rules`) that renames the OOB interface to `oob_net0` and is the same for Yocto, CentOS, and Ubuntu:

```
SUBSYSTEM=="net", ACTION=="add", DEVPATH=="/devices/platform/MLNXBF17:00/net/eth[0-9]", NAME="oob_net0"
```

The files that control IP interface configuration are slightly different for CentOS and Ubuntu:

- CentOS configuration of IP interface:
    - Configuration file for `oob_net0` : `/etc/sysconfig/network-scripts/ifcfg-oob_net0`
    - For example, use the following to enable DHCP:

    ```
    NAME="oob_net0"
    DEVICE="oob_net0"
    NM_CONTROLLED="yes"
    PEERDNS="yes"
    ONBOOT="yes"
    BOOTPROTO="dhcp"
    TYPE=Ethernet
    ```

    - For example, to configure static IP use the following:

    ```
    NAME="oob_net0"
    DEVICE="oob_net0"
    IPV6INIT="no"
    NM_CONTROLLED="no"
    PEERDNS="yes"
    ONBOOT="yes"
    BOOTPROTO="static"
    IPADDR="192.168.200.2"
    PREFIX=30
    GATEWAY="192.168.200.1"
    DNS1="192.168.200.1"
    TYPE=Ethernet
    ```

- For Ubuntu configuration of IP interface, please refer to section "Default Network Interface Configuration".

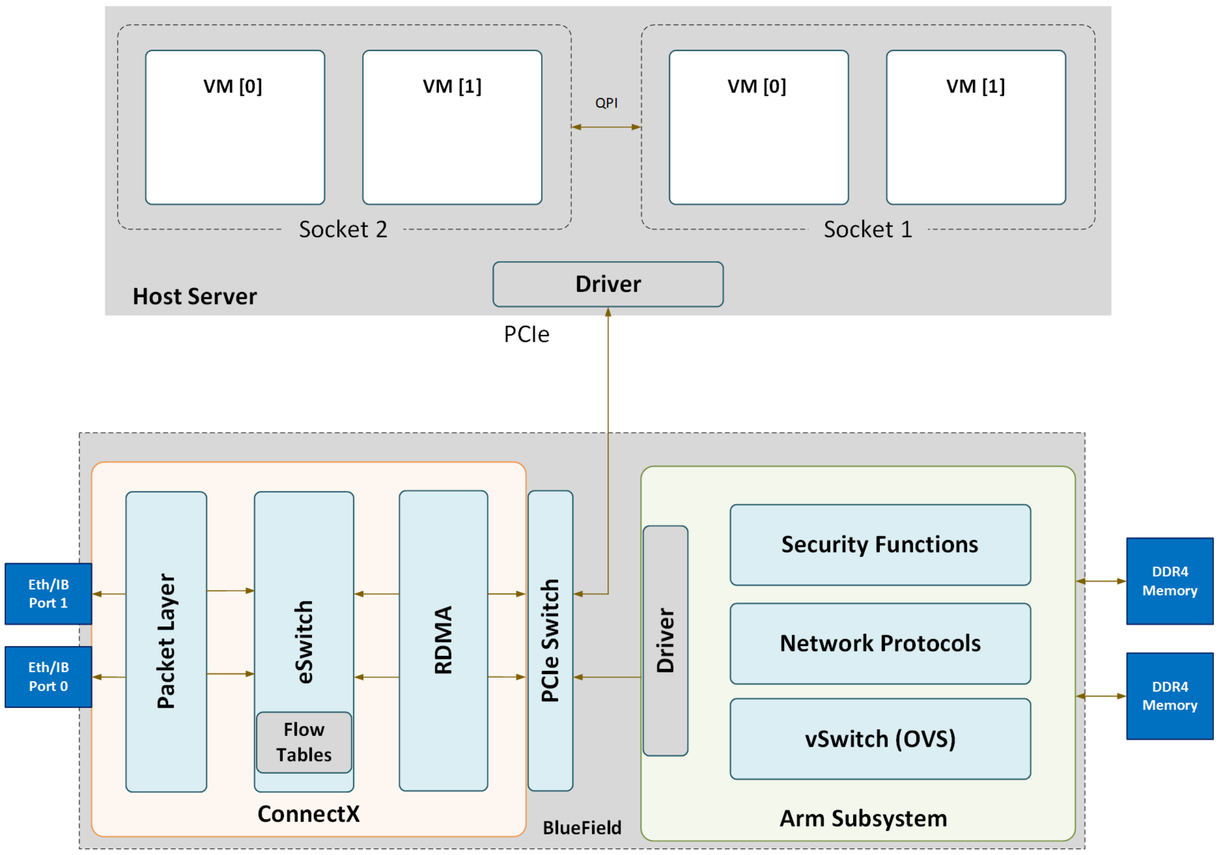
7 BlueField Operation

The NVIDIA® BlueField® networking platform family delivers the flexibility to accelerate a range of applications while leveraging ConnectX-based network controllers hardware-based offloads with unmatched scalability, performance, and efficiency.

- Functional Diagram
- Modes of Operation
- Kernel Representors Model
- Multi-Host
- Virtual Switch on DPU
- Configuring Uplink MTU
- Link Aggregation
- Scalable Functions
- RDMA Stack Support on Host and Arm System
- Controlling Host PF and VF Parameters
- DPDK on BlueField DPU
- BlueField SNAP
- BlueField SR-IOV
- Compression Acceleration
- Public Key Acceleration
- IPsec Functionality
- fTPM over OP-TEE
- QoS Configuration
- Virtio-net Emulated Devices
- Shared RQ Mode

7.1 Functional Diagram

The following is a functional diagram of the NVIDIA® BlueField®-2 DPU.



For each BlueField DPU network port, there are 2 physical PCIe networking functions exposed:

- To the embedded Arm subsystem
- To the host over PCIe

Different functions have different default grace period values during which functions can recover from/handle a single fatal error:

- ECPFs have a graceful period of 3 minutes
- PFs have a graceful period of 1 minute
- VFs/SFs have a graceful period of 30 seconds

The mlx5 drivers and their corresponding software stacks must be loaded on both hosts (Arm and the host server). The OS running on each one of the hosts would probe the drivers. BlueField-2 network interfaces are compatible with NVIDIA® ConnectX®-6 and higher. BlueField-3 network interfaces are compatible with ConnectX-7 and higher.

The same network drivers are used both for BlueField and the ConnectX NIC family.

7.2 Modes of Operation

The NVIDIA® BlueField® DPU has several modes of operation:

- DPU mode, or embedded function (ECPF) ownership, where the embedded Arm system controls the NIC resources and data path

- Zero-trust mode which is an extension of the ECPF ownership with additional restrictions on the host side
- NIC mode where the DPU behaves exactly like an adapter card from the perspective of the external host

Default operation modes

The default mode of operation for BlueField DPU is DPU mode

The default mode of operation for BlueField SuperNIC is NIC mode

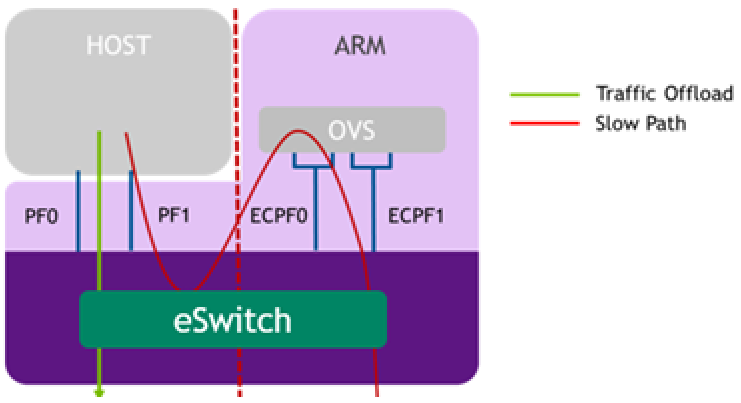
7.2.1 DPU Mode

This mode, known also as embedded CPU function ownership (ECPF) mode, is the default mode for BlueField DPU.

In DPU mode, the NIC resources and functionality are owned and controlled by the embedded Arm subsystem. All network communication to the host flows through a virtual switch control plane hosted on the Arm cores, and only then proceeds to the host. While working in this mode, the DPU is the trusted function managed by the data center and host administrator—to load network drivers, reset an interface, bring an interface up and down, update the firmware, and change the mode of operation on the DPU device.

A network function is still exposed to the host, but it has limited privileges. In particular:

1. The driver on the host side can only be loaded after the driver on the DPU has loaded and completed NIC configuration.
2. All ICM (Interface Configuration Memory) is allocated by the ECPF and resides in the DPU's memory.
3. The ECPF controls and configures the NIC embedded switch which means that traffic to and from the host (DPU) interface always lands on the Arm side.



When the server and DPU are initiated, the networking to the host is blocked until the virtual switch on the DPU is loaded. Once it is loaded, traffic to the host is allowed by default.

There are two ways to pass traffic to the host interface: Either using representors to forward traffic to the host (every packet to/from the host would be handled also by the network interface on the embedded Arm side) or push rules to the embedded switch which allows and offloads this traffic.

In DPU mode, OpenSM must be run from the DPU side (not the host side). Also, management tools (e.g., `sminfo`, `ibdev2netdev`, `ibnetdiscover`) can only be run from the DPU side (not from the host side).

7.2.2 Zero-trust Mode

Zero-trust mode is a specialization of DPU mode which implements an additional layer of security where the host system administrator is prevented from accessing the DPU from the host. Once zero-trust mode is enabled, the data center administrator should control the DPU entirely through the Arm cores and/or BMC connection instead of through the host.

For security and isolation purposes, it is possible to restrict the host from performing operations that can compromise the DPU. The following operations can be restricted individually when changing the DPU host to zero-trust mode:

- Port ownership – the host cannot assign itself as port owner
- Hardware counters – the host does not have access to hardware counters
- Tracer functionality is blocked
- RShim interface is blocked
- Firmware flash is restricted

7.2.2.1 Enabling Zero-trust Mode

To enable host restriction:

1. Start the MST service.
2. Set zero-trust mode. From the Arm side, run:

```
$ sudo mlxprivhost -d /dev/mst/<device> r --disable_rshim --disable_tracer --disable_counter_rd --
disable_port_owner
```

> If any `--disable_*` flags are used, users must perform BlueField system-level reset as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.

7.2.2.2 Disabling Zero-trust Mode

To disable host restriction, set the mode to privileged. Run:

```
$ sudo mlxprivhost -d /dev/mst/<device> p
```

The configuration takes effect immediately.

> If host restriction has been applied using any `--disable_*` flags, users must perform BlueField system-level reset as explained in the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page.

7.2.3 NIC Mode

In this mode, the DPU behaves exactly like an adapter card from the perspective of the external host.

The following instructions presume the DPU to operate in DPU mode. If the DPU is operating in zero-trust mode, please return to DPU mode before continuing.

The following notes are relevant for updating the BFB Bundle in NIC mode:

- During BFB Bundle installation, Linux is expected to boot to upgrade NIC firmware and BMC software
- During the BFB Bundle installation, it is expected for the mlx5 driver to error messages on the x86 host. These prints may be ignored as they are resolved by a mandatory, post-installation power cycle.
- It is mandatory to power cycle the host after the installation is complete for the changes to take effect
- As Linux is booting during BFB Bundle installation, it is expected for the mlx5 core driver to timeout on the BlueField Arm

7.2.3.1 NIC Mode for BlueField-3

When BlueField-3 is configured to operate in NIC mode, Arm OS will not boot.

NIC mode for BlueField-3 saves power, improves device performance, and improves the host memory footprint.

7.2.3.1.1 Configuring NIC Mode on BlueField-3 from Linux

7.2.3.1.1.1 Enabling NIC Mode on BlueField-3 from Linux

Before moving to NIC mode, make sure you are operating in DPU mode by running:

```
host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 -e q
```

The output should have `INTERNAL_CPU_MODEL= EMBBEDDED_CPU(1)` and `EXP_ROM_UEFI_ARM_ENABLE = True (1)` (default).

To enable NIC mode from DPU mode:

1. Run the following on the host or Arm:

   ```
   host/dpu> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s INTERNAL_CPU_OFFLOAD_ENGINE=1
   ```

2. Perform a BlueField system-level reset, for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.2.3.1.1.2 Disabling NIC Mode on BlueField-3 from Linux

To return to DPU mode from NIC mode:

1. Run the following on the host:

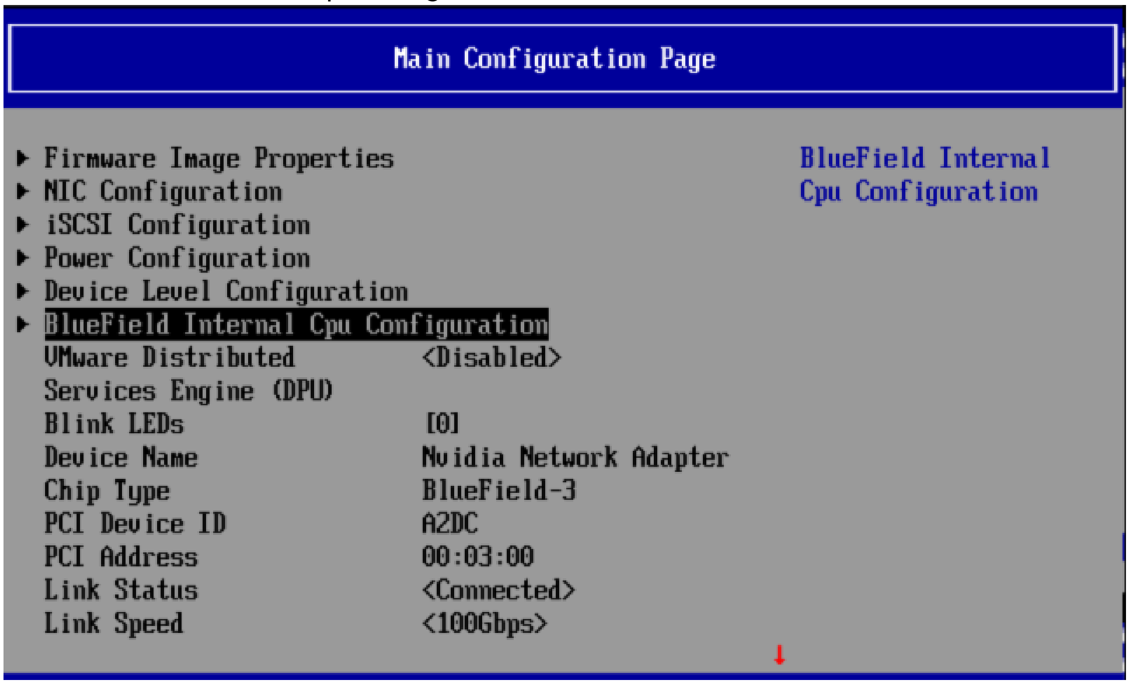
```
host> sudo mlxconfig -d /dev/mst/mt41692_pciconf0 s INTERNAL_CPU_OFFLOAD_ENGINE=0
```

2. Perform a BlueField system-level reset for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.2.3.1.2 Configuring NIC Mode on BlueField-3 from Host BIOS HII UEFI Menu

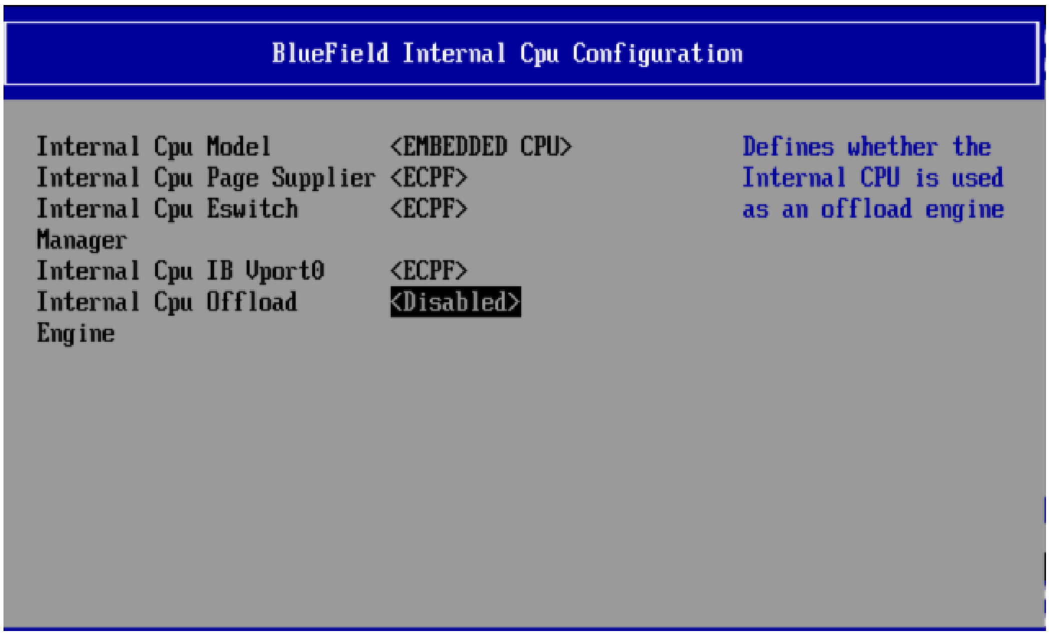
The screenshots in this section are examples only and may vary depending on the vendor of your specific host.

1. Select the network device that presents the uplink (i.e., select the device with the uplink MAC address).
2. Select "BlueField Internal Cpu Configuration".



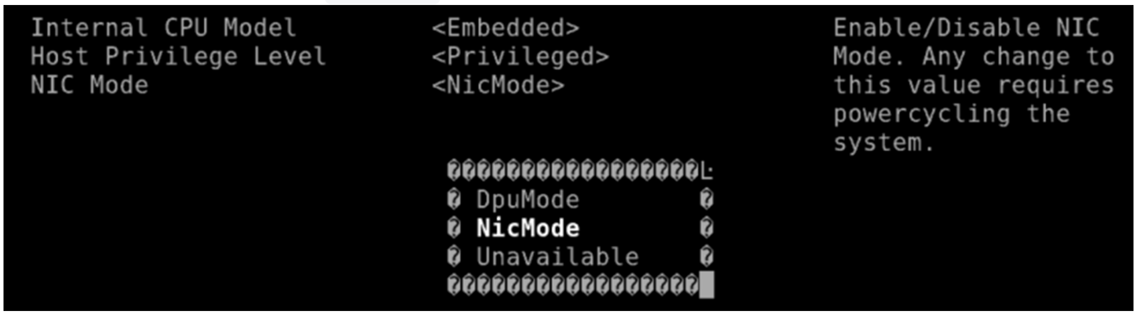
```
Main Configuration Page

▶ Firmware Image Properties                                  BlueField Internal
▶ NIC Configuration                                          Cpu Configuration
▶ iSCSI Configuration
▶ Power Configuration
▶ Device Level Configuration
▶ BlueField Internal Cpu Configuration
  VMware Distributed          <Disabled>
  Services Engine (DPU)
  Blink LEDs                  [0]
  Device Name                 Nvidia Network Adapter
  Chip Type                   BlueField-3
  PCI Device ID               A2DC
  PCI Address                 00:03:00
  Link Status                 <Connected>
  Link Speed                  <100Gbps>
```

- To enable NIC mode, set "Internal Cpu Offload Engine" to "Disabled".
- To switch back to DPU mode, set "Internal Cpu Offload Engine" to "Enabled".


```
BlueField Internal Cpu Configuration

Internal Cpu Model          <EMBEDDED CPU>          Defines whether the
Internal Cpu Page Supplier  <ECPF>                  Internal CPU is used
Internal Cpu Eswitch        <ECPF>                  as an offload engine
Manager
Internal Cpu IB Vport0      <ECPF>
Internal Cpu Offload        <Disabled>
Engine
```

7.2.3.1.3 Configuring NIC Mode on BlueField-3 from Arm UEFI

1. Access the Arm UEFI menu by pressing the Esc button twice.
2. Select "Device Manager".
3. Select "System Configuration".
4. Select "BlueField Modes".
5. Set the "NIC Mode" field to `NicMode` to enable NIC mode.


```
Internal CPU Model          <Embedded>           Enable/Disable NIC
Host Privilege Level        <Privileged>         Mode. Any change to
NIC Mode                    <NicMode>            this value requires
                                                 powercycling the
                                                 system.
                              DpuMode
                              NicMode
                              Unavailable
```

> Configuring `Unavailable` is inapplicable.

6. Exit "BlueField Modes" and "System Configuration" and make sure to save the settings. Exit the UEFI setup using the 'reset' option. The configuration is not yet applied and the DPU is expected to boot regularly, still in DPU Mode.
7. Perform a BlueField system-level reset, to change to NIC Mode. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.2.3.1.4 Configuring NIC Mode on BlueField-3 Using Redfish

Run the following from the BlueField BMC:

1. Get the current BIOS attributes:

```
sudo curl -k -u root:'<password>' -H 'content-type: application/json' -X GET https://<bmc_ip>/redfish/v1/
Systems/Bluefield/Bios/
```

2. Change BlueField mode from `DpuMode` to `NicMode`:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -d '{ "Attributes": { "NicMode":
"NicMode" } }' -X PATCH https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/Settings
```

To revert back to DPU mode, run:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -d '{ "Attributes": { "NicMode":
"DpuMode" } }' -X PATCH https://<bmc_ip>/redfish/v1/Systems/Bluefield/Bios/Settings
```

3. Verify that the BMC has registered the new settings:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X GET https://<bmc_ip>/redfish/v1/
Systems/Bluefield/Bios/Settings
```

4. Issue a software reset then power cycle the host for the change to take effect.
5. Verify the mode is changed:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X GET https://<bmc_ip>/redfish/v1/
Systems/Bluefield/Oem/Nvidia
```

To retrieve the mode via BIOS attributes, another BlueField software reset is required before running the command:

```
curl -k -u root:'<password>' -H 'content-type: application/json' -X GET https://<bmc_ip>/redfish/
v1/Systems/Bluefield/Bios
```

7.2.3.1.5 Updating Firmware Components in BlueField-3 NIC Mode

Once in NIC mode, updating ATF and UFEI can be done using the standard `*.bfb` image:

```
# bfb-install --bfb <BlueField-BSP>.bfb --rshim rshim0
```

7.2.3.2 NIC Mode for BlueField-2

In this mode, the ECPFs on the Arm side are not functional but the user is still able to access the Arm system and update `mlxconfig` options.

When NIC mode is enabled, the drivers and services on the Arm are no longer functional.

7.2.3.2.1 Configuring NIC Mode on BlueField-2 from Linux

7.2.3.2.1.1 Enabling NIC Mode on BlueField-2 from Linux

To enable NIC mode from DPU mode:

1. Run the following from the x86 host side:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s \
INTERNAL_CPU_PAGE_SUPPLIER=1 \
INTERNAL_CPU_ESWITCH_MANAGER=1 \
INTERNAL_CPU_IB_VPORT0=1 \
INTERNAL_CPU_OFFLOAD_ENGINE=1
```

   To restrict RShim PF (optional), make sure to configure `INTERNAL_CPU_RSHIM=1` as part of the `mlxconfig` command.

2. Perform BlueField system-level reset to load the new configuration.

   Refer to the troubleshooting section of the guide for a step-by-step procedure.

Multi-host is not supported when the DPU is operating in NIC mode.

To obtain firmware BINs for BlueField-2 devices, please refer to the BlueField-2 firmware download page.

7.2.3.2.1.2 Disabling NIC Mode on BlueField-2 from Linux

To change from NIC mode back to DPU mode:

1. Install and start the RShim driver on the host.
2. Disable NIC mode. Run:

```
$ mst start
$ mlxconfig -d /dev/mst/<device> s \
INTERNAL_CPU_PAGE_SUPPLIER=0 \
INTERNAL_CPU_ESWITCH_MANAGER=0 \
INTERNAL_CPU_IB_VPORT0=0 \
INTERNAL_CPU_OFFLOAD_ENGINE=0
```

   If `INTERNAL_CPU_RSHIM=1`, then make sure to configure `INTERNAL_CPU_RSHIM=0` as part of the `mlxconfig` command.

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.2.3.2.2 Configuring NIC Mode on BlueField-2 from Arm UEFI

Follow the same instructions in section "Configuring NIC Mode on BlueField-3 from Arm UEFI".

7.2.3.2.3 Configuring NIC Mode on BlueField-2 Using Redfish

Follow the same instructions in section "Configuring NIC Mode on BlueField-3 Using Redfish".

7.3 Kernel Representors Model

This model is only applicable when the DPU is operating in DPU mode.

BlueField® DPU uses netdev representors to map each one of the host side physical and virtual functions.

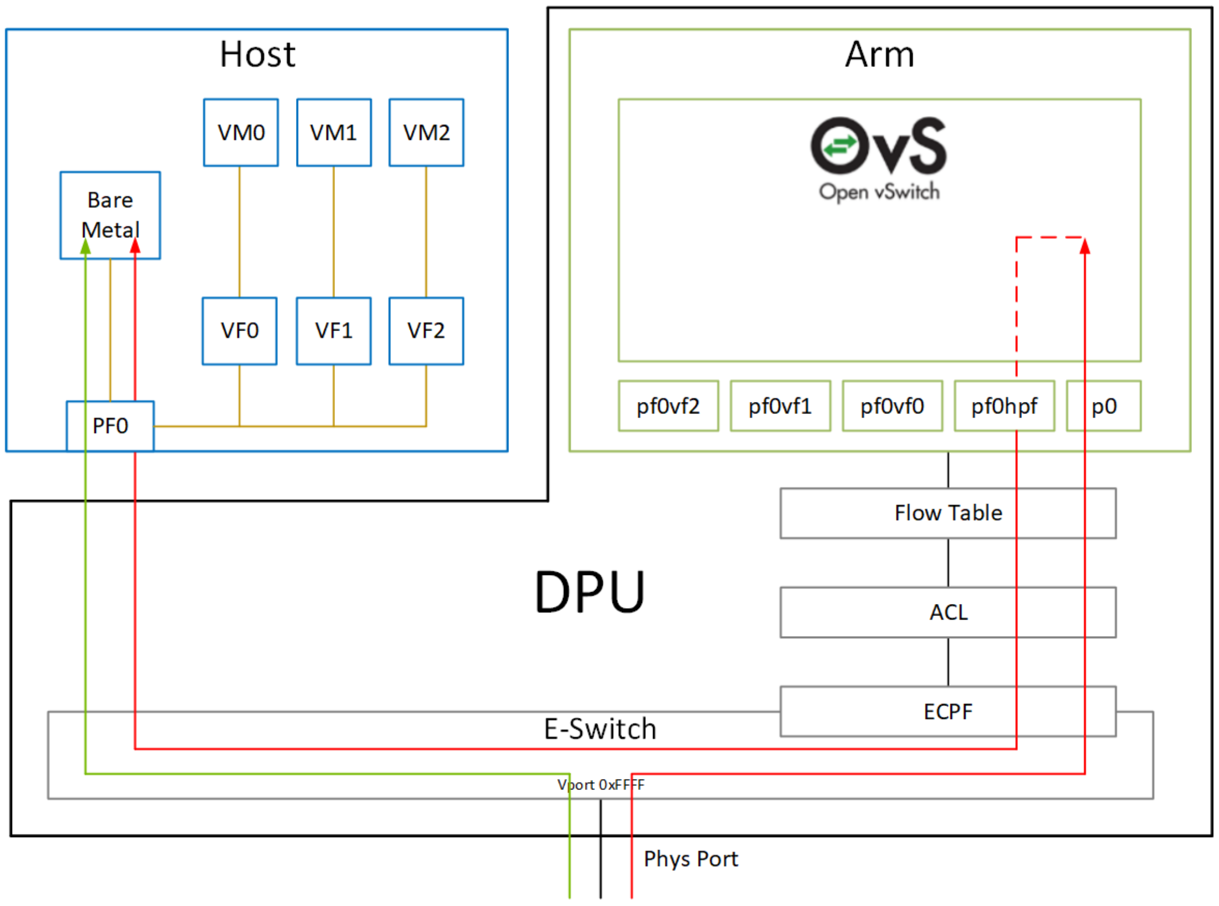
1. Serve as the tunnel to pass traffic for the virtual switch or application running on the Arm cores to the relevant PF or VF on the host side.
2. Serve as the channel to configure the embedded switch with rules to the corresponding represented function.

Those representors are used as the virtual ports being connected to OVS or any other virtual switch running on the Arm cores.

When operating in DPU mode, we see 2 representors for each one of the DPU's network ports: one for the uplink, and another one for the host side PF (the PF representor created even if the PF is not probed on the host side). For each one of the VFs created on the host side a corresponding representor would be created on the Arm side. The naming convention for the representors is as follows:

- Uplink representors: `p<port_number>`
- PF representors: `pf<port_number>hpf`
- VF representors: `pf<port_number>vf<function_number>`

The following diagram shows the mapping of between the PCIe functions exposed on the host side and the representors. For the sake of simplicity, a single port model (duplicated for the second port) is shown.



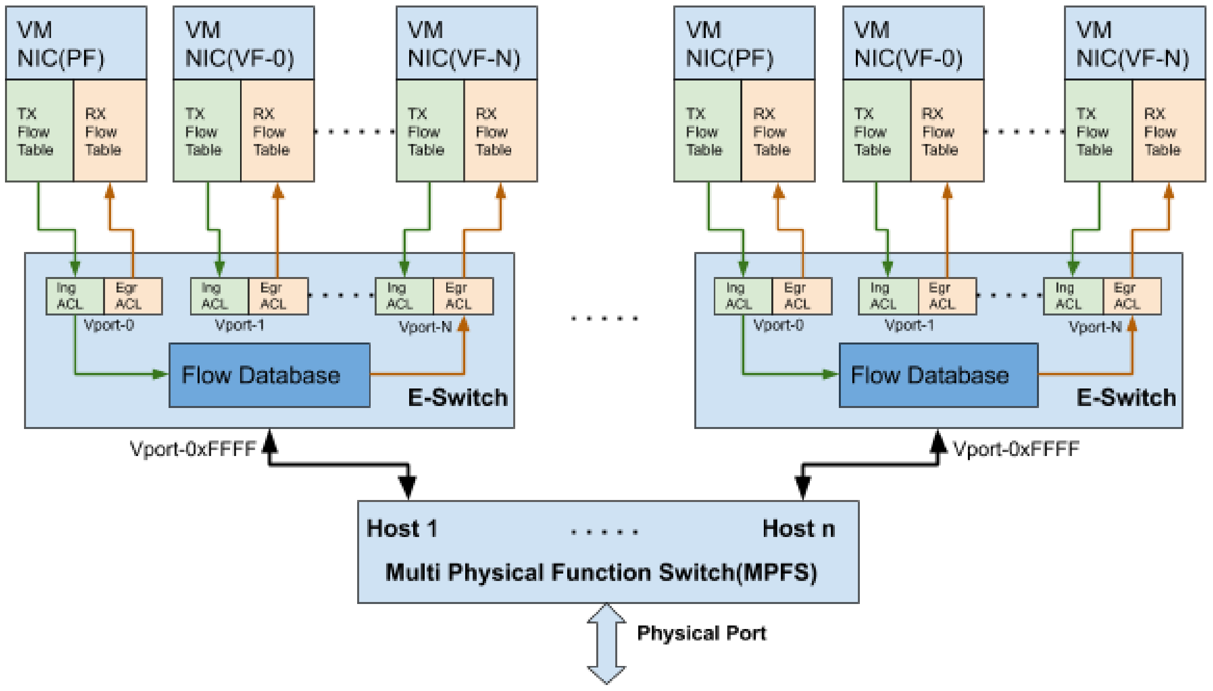
The red arrow demonstrates a packet flow through the representors, while the green arrow demonstrates the packet flow when steering rules are offloaded to the embedded switch. More details on that are available in the switch offload section.

The MTU of host functions (PF/VF) must be smaller than the MTUs of both the uplink and corresponding PF/VF representor. For example, if the host PF MTU is set to 9000, both uplink and PF representor must be set to above 9000.

7.4 Multi-Host

This is only applicable to DPUs running on multi-host model.

In multi-host mode, each host interface can be divided into up to 4 independent PCIe interfaces. All interfaces would share the same physical port, and are managed by the same multi-physical function switch (MPFS). Each host would have its own e-switch and would control its own traffic.



7.4.1 Representors

Similar to Kernel Representors Model, each host here has an uplink representor, PF representor, and VF representors (if SR-IOV is enabled). There are 8 sets of representors (uplink/PF; see example code). For each host to work with OVS offload, the corresponding representors must be added to the OVS bridge.

```
139: p0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b2 brd ff:ff:ff:ff:ff:ff
140: p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b3 brd ff:ff:ff:ff:ff:ff
141: p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b4 brd ff:ff:ff:ff:ff:ff
142: p3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b5 brd ff:ff:ff:ff:ff:ff
143: p4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b6 brd ff:ff:ff:ff:ff:ff
144: p5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b7 brd ff:ff:ff:ff:ff:ff
145: p6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b8 brd ff:ff:ff:ff:ff:ff
146: p7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:b9 brd ff:ff:ff:ff:ff:ff
147: pf0hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP group default qlen 1000
    link/ether 86:c5:8a:b7:7c:84 brd ff:ff:ff:ff:ff:ff
148: pf1hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 6e:ea:1b:84:88:49 brd ff:ff:ff:ff:ff:ff
149: pf2hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 92:ec:99:cb:d7:23 brd ff:ff:ff:ff:ff:ff
150: pf3hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0e:0d:8e:03:2e:27 brd ff:ff:ff:ff:ff:ff
151: pf4hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 5e:42:af:05:67:93 brd ff:ff:ff:ff:ff:ff
152: pf5hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 96:e4:69:4c:b7:7f brd ff:ff:ff:ff:ff:ff
153: pf6hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 5e:67:33:c0:35:05 brd ff:ff:ff:ff:ff:ff
154: pf7hpf: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 12:29:7d:56:07:3e brd ff:ff:ff:ff:ff:ff
```

The following is an example of adding all representors to OVS:

```
    Bridge armBr-3
        Port armBr-3
            Interface armBr-3
                type: internal
        Port p3
            Interface p3
        Port pf3hpf
```

```
                Interface pf3hpf
        Bridge armBr-2
            Port p2
                Interface p2
            Port pf2hpf
                Interface pf2hpf
            Port armBr-2
                Interface armBr-2
                    type: internal
        Bridge armBr-5
            Port p5
                Interface p5
            Port pf5hpf
                Interface pf5hpf
            Port armBr-5
                Interface armBr-5
                    type: internal
        Bridge armBr-7
            Port pf7hpf
                Interface pf7hpf
            Port armBr-7
                Interface armBr-7
                    type: internal
            Port p7
                Interface p7
        Bridge armBr-0
            Port p0
                Interface p0
            Port armBr-0
                Interface armBr-0
                    type: internal
            Port pf0hpf
                Interface pf0hpf
        Bridge armBr-4
            Port p4
                Interface p4
            Port pf4hpf
                Interface pf4hpf
            Port armBr-4
                Interface armBr-4
                    type: internal
        Bridge armBr-1
            Port armBr-1
                Interface armBr-1
                    type: internal
            Port p1
                Interface p1
            Port pf1hpf
                Interface pf1hpf
        Bridge armBr-6
            Port armBr-6
                Interface armBr-6
                    type: internal
            Port p6
                Interface p6
            Port pf6hpf
                Interface pf6hpf
        ovs_version: "2.13.1"
```

For now, users can get the representor-to-host PF mapping by comparing the MAC address queried from host control on the Arm-side and PF MAC on the host-side. In the following example, the user knows p0 is the uplink representor for p6p1 as the MAC address is the same.

From Arm:

```
# cat /sys/class/net/p0/smart_nic/pf/config
MAC         : 0c:42:a1:70:1d:9a
MaxTxRate   : 0
State       : Up
```

From host:

```
# ip addr show p6p1
3: p6p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 0c:42:a1:70:1d:9a brd ff:ff:ff:ff:ff:ff
```

The implicit mapping is as follows:

- PF0, PF1 = host controller 1
- PF2, PF3 = host controller 2
- PF4, PF5 = host controller 3
- PF6, PF7 = host controller 4

The maximum SF or VF count across all hosts is limited to 488 in total. The user can divide 488 VFs/SFs to single or multiple controllers as desired.

7.5 Virtual Switch on DPU

For general information on OVS offload using ASAP² direct, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP² Direct.

ASAP^2 is only supported in Embedded (DPU) mode.

NVIDIA® BlueField® supports ASAP^2 technology. It utilizes the representors mentioned in the previous section. BlueField SW package includes OVS installation which already supports ASAP^2. The virtual switch running on the Arm cores allows us to pass all the traffic to and from the host functions through the Arm cores while performing all the operations supported by OVS. ASAP^2 allows us to offload the datapath by programming the NIC embedded switch and avoiding the need to pass every packet through the Arm cores. The control plane remains the same as working with standard OVS.

OVS bridges are created by default upon first boot of the DPU after BFB installation.

If manual configuration of the default settings for the OVS bridge is desired, run:

```
systemctl start openvswitch-switch.service
ovs-vsctl add-port ovsbr1 p0
ovs-vsctl add-port ovsbr1 pf0hpf
ovs-vsctl add-port ovsbr2 p1
ovs-vsctl add-port ovsbr2 pf1hpf
```

To verify successful bridging:

```
$ ovs-vsctl show
9f635bd1-a9fd-4f30-9bdc-b3fa21f8940a
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port p1
            Interface p1
        Port pf1sf0
            Interface en3f1pf1sf0
        Port pf1hpf
            Interface pf1hpf
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port p0
            Interface p0
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port pf0sf0
            Interface en3f0pf0sf0
    ovs_version: "2.14.1"
```

The host is now connected to the network.

TC-offload is not supported for IPv6 fragment packets. To make IPv6 fragment packets pass through OVS, the MTU of a specific port must be set to equal to or larger than the fragmented packet size. IPv4 fragment packets can be TC-offloaded as their packet size is not checked by OVS.

7.5.1 Verifying Host Connection on Linux

When the DPU is connected to another DPU on another machine, manually assign IP addresses with the same subnet to both ends of the connection.

1. Assuming the link is connected to p3p1 on the other host, run:

   ```
   $ ifconfig p3p1 192.168.200.1/24 up
   ```

2. On the host which the DPU is connected to, run:

   ```
   $ ifconfig p4p2 192.168.200.2/24 up
   ```

3. Have one ping the other. This is an example of the DPU pinging the host:

   ```
   $ ping 192.168.200.1
   ```

7.5.2 Verifying Connection from Host to BlueField

There are two SFs configured on the BlueFIeld-2 device, `enp3s0f0s0` and `enp3s0f1s0`, and their representors are part of the built-in bridge. These interfaces will get IP addresses from the DHCP server if it is present. Otherwise it is possible to configure IP address from the host. It is possible to access BlueField via the SF netdev interfaces.

For example:

1. Verify the default OVS configuration. Run:

   ```
   # ovs-vsctl show
   5668f9a6-6b93-49cf-a72a-14fd64b4c82b
       Bridge ovsbr1
           Port pf0hpf
               Interface pf0hpf
           Port ovsbr1
               Interface ovsbr1
                   type: internal
           Port p0
               Interface p0
           Port en3f0pf0sf0
               Interface en3f0pf0sf0
       Bridge ovsbr2
           Port en3f1pf1sf0
               Interface en3f1pf1sf0
           Port ovsbr2
               Interface ovsbr2
                   type: internal
           Port pf1hpf
               Interface pf1hpf
           Port p1
               Interface p1
       ovs_version: "2.14.1"
   ```

2. Verify whether the SF netdev received an IP address from the DHCP server. If not, assign a static IP. Run:

```
# ifconfig enp3s0f0s0
enp3s0f0s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.200.125  netmask 255.255.255.0  broadcast 192.168.200.255
        inet6 fe80::8e:bcff:fe36:19bc  prefixlen 64  scopeid 0x20<link>
        ether 02:8e:bc:36:19:bc  txqueuelen 1000  (Ethernet)
        RX packets 3730  bytes 1217558 (1.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 2220 (2.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. Verify the connection of the configured IP address. Run:

```
# ping 192.168.200.25 -c 5
PING 192.168.200.25 (192.168.200.25) 56(84) bytes of data.
64 bytes from 192.168.200.25: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 192.168.200.25: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 192.168.200.25: icmp_seq=3 ttl=64 time=0.232 ms
64 bytes from 192.168.200.25: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 192.168.200.25: icmp_seq=5 ttl=64 time=0.168 ms

--- 192.168.200.25 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.168/0.195/0.232/0.031 ms
```

7.5.3 Verifying Host Connection on Windows

Set IP address on the Windows side for the RShim or Physical network adapter, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> New-NetIPAddress -InterfaceAlias "Ethernet 16" -IPAddress "192.168.100.1" -PrefixLength
22
```

To get the interface name, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> Get-NetAdapter
```

Output should give us the interface name that matches the description (e.g. NVIDIA BlueField Management Network Adapter).

```
Ethernet 2          NVIDIA ConnectX-4 Lx Ethernet Adapter        6 Not Present  24-8A-07-0D-E8-1D
Ethernet 6          NVIDIA ConnectX-4 Lx Ethernet Ad...#2       23 Not Present  24-8A-07-0D-E8-1C
Ethernet 16         NVIDIA BlueField Management Netw...#2       15 Up           CA-FE-01-CA-FE-02
```

Once IP address is set, Have one ping the other.

```
C:\Windows\system32>ping 192.168.100.2


Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=148ms TTL=64
Reply from 192.168.100.2: bytes=32 time=152ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
```

7.5.4 Enabling OVS HW Offloading

OVS HW offloading is set by default by the `/sbin/mlnx_bf_configure` script upon first boot after installation.

1. Enable TC offload on the relevant interfaces. Run:

```
$ ethtool -K <PF> hw-tc-offload on
```

2. Enable the HW offload: run the following commands (after enabling the HW offload):

   ```
   $ ovs-vsctl set Open_vSwitch . Other_config:hw-offload=true
   ```

3. Restarting OVS is required for the configuration to apply:
   - For Ubuntu:

     ```
     $ systemctl restart openvswitch-switch
     ```

   - For CentOS/RHEL:

     ```
     $ systemctl restart openvswitch
     ```

To show OVS configuration:

```
$ ovs-dpctl show
system@ovs-system:
  lookups: hit:0 missed:0 lost:0
  flows: 0
  masks: hit:0 total:0 hit/pkt:0.00
  port 0: ovs-system (internal)
  port 1: armbr1 (internal)
  port 2: p0
  port 3: pf0hpf
  port 4: pf0vf0
  port 5: pf0vf1
  port 6: pf0vf2
```

At this point OVS would automatically try to offload all the rules.

To see all the rules that are added to the OVS datapath:

```
$ ovs-appctl dpctl/dump-flows
```

To see the rules that are offloaded to the HW:

```
$ ovs-appctl dpctl/dump-flows type=offloaded
```

7.5.5 Enabling OVS-DPDK Hardware Offload

1. Remove previously configured OVS bridges. Run:

   ```
   ovs-vsctl del-br <bridge-name>
   ```

   Issue the command `ovs-vsctl show` to see already configured OVS bridges.
2. Enable the Open vSwitch service. Run:

   ```
   systemctl start openvswitch
   ```

3. Configure huge pages:

   ```
   echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

4. Configure DPDK socket memory and limit. Run:

   ```
   # ovs-vsctl set Open_vSwitch . other_config:dpdk-socket-limit=2048
   ```

```
# ovs-vsctl set Open_vSwitch . other_config:dpdk-socket-mem=2048
```

5. Enable hardware offload (disabled by default). Run:

   ```
   ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
   ovs-vsctl --no-wait set Open_vSwitch . other_config:hw-offload=true
   ```

6. Configure the DPDK whitelist. Run:

   ```
   ovs-vsctl set Open_vSwitch . other_config:dpdk-extra="-a
   0000:03:00.0,representor=[0,65535],dv_flow_en=1,dv_xmeta_en=1,sys_mem_en=1"
   ```

7. Create OVS-DPDK bridge. Run:

   ```
   ovs-vsctl add-br br0-ovs -- set Bridge br0-ovs datapath_type=netdev -- br-set-external-id br0-ovs bridge-id
   br0-ovs -- set bridge br0-ovs fail-mode=standalone
   ```

8. Add PF to OVS. Run:

   ```
   ovs-vsctl add-port br0-ovs p0 -- set Interface p0  type=dpdk options:dpdk-devargs=0000:03:00.0
   ```

9. Add representor to OVS. Run:

   ```
   ovs-vsctl add-port br0-ovs pf0vf0 -- set Interface pf0vf0 type=dpdk options:dpdk-
   devargs=0000:03:00.0,representor=[0]
   ovs-vsctl add-port br0-ovs pf0hpf -- set Interface pf0hpf type=dpdk options:dpdk-
   devargs=0000:03:00.0,representor=[65535]
   ```

10. Restart the Open vSwitch service. This step is required for HW offload changes to take effect.
    - For CentOS, run:

      ```
      systemctl restart openvswitch
      ```

    - For Debian/Ubuntu, run:

      ```
      systemctl restart openvswitch-switch
      ```

For a reference setup configuration for BlueField-2 devices, refer to the article "Configuring OVS-DPDK Offload with BlueField-2".

7.5.6 Configuring DPDK and Running TestPMD

1. Configure hugepages. Run:

   ```
   echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

2. Run testpmd.
   - For Ubuntu/Debian:

     ```
     env LD_LIBRARY_PATH=/opt/mellanox/dpdk/lib/aarch64-linux-gnu /opt/mellanox/dpdk/bin/dpdk-testpmd -a
     03:00.0,representor=[0,65535] --socket-mem=1024 -- --total-num-mbufs=131000 -i
     ```

   - For CentOS:

```
env LD_LIBRARY_PATH=/opt/mellanox/dpdk/lib64/ /opt/mellanox/dpdk/bin/dpdk-testpmd -a
03:00.0,representor=[0,65535] --socket-mem=1024 -- --total-num-mbufs=131000 -i
```

For a detailed procedure with port display, refer to the article "Configuring DPDK and Running testpmd on BlueField-2".

7.5.7 Flow Statistics and Aging

The aging timeout of OVS is given in milliseconds and can be configured by running the following command:

```
$ ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

7.5.8 Connection Tracking Offload

This feature enables tracking connections and storing information about the state of these connections. When used with OVS, the DPU can offload connection tracking, so that traffic of established connections bypasses the kernel and goes directly to hardware.

Both source NAT (SNAT) and destination NAT (DNAT) are supported with connection tracking offload.

7.5.8.1 Configuring Connection Tracking Offload

This section provides an example of configuring OVS to offload all IP connections of host PF0.

1. Enable OVS HW offloading.
2. Create OVS connection tracking bridge. Run:

   ```
   $ ovs-vsctl add-br ctBr
   ```
3. Add p0 and pf0hpf to the bridge. Run:

   ```
   $ ovs-vsctl add-port ctBr p0
   $ ovs-vsctl add-port ctBr pf0hpf
   ```
4. Configure ARP packets to behave normally. Packets which do not comply are routed to table1. Run:

   ```
   $ ovs-ofctl add-flow ctBr "table=0,arp,action=normal"
   $ ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"
   ```
5. Configure RoCEv2 packets to behave normally. RoCEv2 packets follow UDP port 4791 and a different source port in each direction of the connection. RoCE traffic is not supported by CT. In order to run RoCE from the host add the following line before `ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"`:

   ```
   $ ovs-ofctl add-flow ctBr table=0,udp,tp_dst=4791,action=normal
   ```
   This rule allows RoCEv2 UDP packets to skip connection tracking rules.
6. Configure the new established flows to be admitted to the connection tracking bridge and to then behave normally. Run:

```
$ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk+new,action=ct(commit),normal"
```

7. Set already established flows to behave normally. Run:

```
$ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk+est,action=normal"
```

7.5.8.2 Connection Tracking With NAT

This section provides an example of configuring OVS to offload all IP connections of host PF0, and performing source network address translation (SNAT). The server host sends traffic via source IP from 2.2.2.1 to 1.1.1.2 on another host. Arm performs SNAT and changes the source IP to 1.1.1.16. Note that static ARP or route table must be configured to find that route.

1. Configure untracked IP packets to do nat. Run:

```
ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1,nat)"
```

2. Configure new established flows to do SNAT, and change source IP to 1.1.1.16. Run:

```
ovs-ofctl add-flow ctBr "table=1,in_port=pf0hpf,ip,ct_state=+trk+new,action=ct(commit,nat(src=1.1.1.16)),
p0"
```

3. Configure already established flows act normal. Run:

```
ovs-ofctl add-flow ctBr "table=1,ip,ct_state=+trk+est,action=normal"
```

Conntrack shows the connection with SNAT applied. Run `conntrack -L` for Ubuntu 22.04 kernel or `cat /proc/net/nf_conntrack` for older kernel versions. Example output:

```
ipv4     2 tcp      6 src=2.2.2.1 dst=1.1.1.2 sport=34541 dport=5001 src=1.1.1.2 dst=1.1.1.16 sport=5001
dport=34541 [OFFLOAD] mark=0 zone=1 use=3
```

7.5.8.3 Querying Connection Tracking Offload Status

Start traffic on PF0 from the server host (e.g., iperf) with an external network. Note that only established connections can be offloaded. TCP should have already finished the handshake, UDP should have gotten the reply.

ICMP is not currently supported.

To check if specific connections are offloaded from Arm, run `conntrack -L` for Ubuntu 22.04 kernel or `cat /proc/net/nf_conntrack` for older kernel versions.

The following is example output of offloaded TCP connection:

```
ipv4     2 tcp      6 src=1.1.1.2 dst=1.1.1.3 sport=51888 dport=5001 src=1.1.1.3 dst=1.1.1.2 sport=5001 dport=51888
[HW_OFFLOAD] mark=0 zone=0 use=3
```

7.5.8.4 Performance Tune Based on Traffic Pattern

Offloaded flows (including connection tracking) are added to virtual switch FDB flow tables. FDB tables have a set of flow groups. Each flow group saves the same traffic pattern flows. For example, for connection tracking offloaded flow, TCP and UDP are different traffic patterns which end up in two different flow groups.

A flow group has a limited size to save flow entries. By default, the driver has 4 big FDB flow groups. Each of these big flow groups can save at most 4000000/(4+1)=800k different 5-tuple flow entries. For scenarios with more than 4 traffic patterns, the driver provides a module parameter (`num_of_groups`) to allow customization and performance tune.

> The size of each big flow groups can be calculated according to formula: size = 4000000/(num_of_groups+1)

To change the number of big FDB flow groups, run:

```
$ echo <num_of_groups> > /sys/module/mlx5_core/parameters/num_of_groups
```

The change takes effect immediately if there is no flow inside the FDB table (no traffic running and all offloaded flows are aged out), and it can be dynamically changed without reloading the driver.

If there are residual offloaded flows when changing this parameter, then the new configuration only takes effect after all flows age out.

7.5.8.5 Connection Tracking Aging

Aside from the aging of OVS, connection tracking offload has its own aging mechanism with a default aging time of 30 seconds.

7.5.8.6 Maximum Tracked Connections

> The maximum number for tracked offloaded connections is limited to 1M by default.

The OS has a default setting of maximum tracked connections which may be configured by running:

```
$ /sbin/sysctl -w net.netfilter.nf_conntrack_max=1000000
```

This changes the maximum tracked connections (both offloaded and non-offloaded) setting to 1 million.

The following option specifies the limit on the number of offloaded connections. For example:

```
# devlink dev param set pci/${pci_dev} name ct_max_offloaded_conns value $max cmode runtime
```

This value is set to 1 million by default from BlueFiled. Users may choose a different number by using the `devlink` command.

Make sure `net.netfilter.nf_conntrack_tcp_be_liberal=1` when using connection tracking.

7.5.9 Offloading VLANs

OVS enables VF traffic to be tagged by the virtual switch.

For the BlueField DPU, the OVS can add VLAN tag (VLAN push) to all the packets sent by a network interface running on the host (either PF or VF) and strip the VLAN tag (VLAN pop) from the traffic going from the wire to that interface. Here we operate in Virtual Switch Tagging (VST) mode. This means that the host/VM interface is unaware of the VLAN tagging. Those rules can also be offloaded to the HW embedded switch.

To configure OVS to push/pop VLAN you need to add the tag=$TAG section for the OVS command line that adds the representor ports. So if you want to tag all the traffic of VF0 with VLAN ID 52, you should use the following command when adding its representor to the bridge:

```
$ ovs-vsctl add-port armbr1 pf0vf0 tag=52
```

If the virtual port is already connected to the bridge prior to configuring VLAN, you would need to remove it first:

```
$ ovs-vsctl del-port pf0vf0
```

In this scenario all the traffic being sent by VF 0 will have the same VLAN tag. We could set a VLAN tag by flow when using the TC interface, this is explained in section "Using TC Interface to Configure Offload Rules".

7.5.10 VXLAN Tunneling Offload

VXLAN tunnels are created on the Arm side and attached to the OVS. VXLAN decapsulation/encapsulation behavior is similar to normal VXLAN behavior, including over `hw_offload=true`.

To allow VXLAN encapsulation, the uplink representor (`p0`) should have an MTU value at least 50 bytes greater than that of the host PF/VF. Please refer to "Configuring Uplink MTU" for more information.

7.5.10.1 Configuring VXLAN Tunnel

1. Consider `p0` to be the local VXLAN tunnel interface (or VTEP).

   To be consistent with the examples below, it is assumed that `p0` is configured with a 1.1.1.1 IPv4 address.

2. Remove `p0` from any OVS bridge.
3. Build a VXLAN tunnel over OVS arm-ovs. Run:

```
ovs-vsctl add-br arm-ovs -- add-port arm-ovs vxlan11 -- set interface vxlan11 type=vxlan
options:local_ip=1.1.1.1 options:remote_ip=1.1.1.2 options:key=100
options:dst_port=4789
```

4. Connect any host representor (e.g., `pf0hpf`) for which VXLAN is desired to the same arm-ovs bridge.
5. Configure the MTU of the VTEP (`p0`) used by VXLAN to at least 50 bytes larger than the host representor's MTU.

At this point, the host is unaware of any VXLAN operations done by the DPU's OVS. If the remote end of the VXLAN tunnel is properly set, any network traffic traversing arm-ovs undergoes VXLAN encap/decap.

7.5.10.2 Querying OVS VXLAN hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
in_port(2),eth(src=ae:fd:f3:31:7e:7b,dst=a2:fb:09:85:84:48),eth_type(0x0800), packets:1, bytes:98, used:0.900s,
actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,tp_dst=4789,flags(key))),3
tunnel(tun_id=0x64,src=1.1.1.2,dst=1.1.1.1,tp_dst=4789,flags(+key)),in_port(3),eth(src=a2:fb:09:85:84:48,dst=ae:fd:
f3:31:7e:7b),eth_type(0x0800), packets:75, bytes:7350, used:0.900s, actions:2
```

For the host PF, in order for VXLAN to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

2. The MTU of the end points (`pf0hpf` in the example above) of the VXLAN tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the VXLAN headers. For example, you can set the MTU of P0 to 2000.

7.5.11 GRE Tunneling Offload

GRE tunnels are created on the Arm side and attached to the OVS. GRE decapsulation/encapsulation behavior is similar to normal GRE behavior, including over `hw_offload=true`.

To allow GRE encapsulation, the uplink representor (`p0`) should have an MTU value at least 50 bytes greater than that of the host PF/VF.

Please refer to "Configuring Uplink MTU" for more information.

7.5.11.1 Configuring GRE Tunnel

1. Consider `p0` to be the local GRE tunnel interface. `p0` should not be attached to any OVS bridge.

To be consistent with the examples below, it is assumed that `p0` is configured with a 1.1.1.1 IPv4 address and that the remote end of the tunnel is 1.1.1.2.

2. Create an OVS bridge, `br0`, with a GRE tunnel interface, `gre0`. Run:

   ```
   ovs-vsctl add-port br0 gre0 -- set interface gre0 type=gre options:local_ip=1.1.1.1
   options:remote_ip=1.1.1.2 options:key=100
   ```

3. Add `pf0hpf` to `br0`.

   ```
   ovs-vsctl add-port br0 pf0hpf
   ```

4. At this point, any network traffic sent or received by the host's PF0 undergoes GRE processing inside the BlueField OS.

7.5.11.2 Querying OVS GRE hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
recirc_id(0),in_port(3),eth(src=50:6b:4b:2f:0b:74,dst=de:d0:a3:63:0b:30),eth_type(0x0800),ipv4(frag=no),
packets:878, bytes:122802, used:0.440s,
actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,ttl=64,flags(key))),2
tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,flags(+key)),recirc_id(0),in_port(2),eth(src=de:d0:a3:63:0b:30,dst=50:6b
:4b:2f:0b:74),eth_type(0x0800),ipv4(frag=no), packets:995, bytes:97510, used:0.440s, actions:3
```

For the host PF, in order for GRE to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

2. The MTU of the end points (`pf0hpf` in the example above) of the GRE tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the GRE headers. For example, you can set the MTU of P0 to 2000.

7.5.12 GENEVE Tunneling Offload

GENEVE tunnels are created on the Arm side and attached to the OVS. GENEVE decapsulation/encapsulation behavior is similar to normal GENEVE behavior, including over `hw_offload=true`.

To allow GENEVE encapsulation, the uplink representor (`p0`) must have an MTU value at least 50 bytes greater than that of the host PF/VF.

Please refer to "Configuring Uplink MTU" for more information.

7.5.12.1 Configuring GENEVE Tunnel

1. Consider `p0` to be the local GENEVE tunnel interface. `p0` should not be attached to any OVS bridge.

2. Create an OVS bridge, `br0`, with a GENEVE tunnel interface, `gnv0`. Run:

```
ovs-vsctl add-port br0 gnv0 -- set interface gnv0 type=geneve options:local_ip=1.1.1.1
options:remote_ip=1.1.1.2 options:key=100
```

3. Add `pf0hpf` to `br0`.

```
ovs-vsctl add-port br0 pf0hpf
```

4. At this point, any network traffic sent or received by the host's PF0 undergoes GENEVE processing inside the BlueField OS.

Options are supported for GENEVE. For example, you may add option `0xea55` to tunnel metadata, run:

```
ovs-ofctl add-tlv-map geneve_br "{class=0xffff,type=0x0,len=4}->tun_metadata0"
ovs-ofctl add-flow geneve_br ip,actions="set_field:0xea55->tun_metadata0",normal
```

For the host PF, in order for GENEVE to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

```
$ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
```

2. The MTU of the end points (`pf0hpf` in the example above) of the GENEVE tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the GENEVE headers. For example, you can set the MTU of P0 to 2000.

7.5.13 Using TC Interface to Configure Offload Rules

Offloading rules can also be added directly, and not just through OVS, using the tc utility. To enable TC ingress on all the representors (i.e., uplink, PF, and VF).

```
$ tc qdisc add dev p0 ingress
$ tc qdisc add dev pf0hpf ingress
$ tc qdisc add dev pf0vf0 ingress
```

7.5.13.1 L2 Rules Example

The rule below drops all packets matching the given source and destination MAC addresses.

```
$ tc filter add dev pf0hpf protocol ip parent ffff: \
                               flower \
                                            skip_sw \
                                            dst_mac e4:11:22:11:4a:51 \
                                            src_mac e4:11:22:11:4a:50 \
                                action drop
```

7.5.13.2 VLAN Rules Example

The following rules push VLAN ID 100 to packets sent from VF0 to the wire (and forward it through the uplink representor) and strip the VLAN when sending the packet to the VF.

```
$ tc filter add dev pf0vf0 protocol 802.1Q parent ffff: \
                                    flower \
                                               skip_sw \
                                               dst_mac e4:11:22:11:4a:51 \
                                               src_mac e4:11:22:11:4a:50 \
                                    action vlan push id 100 \
                                    action mirred egress redirect dev p0

$ tc filter add dev p0 protocol 802.1Q parent ffff: \
                                    flower \
                                              skip_sw \
                                              dst_mac e4:11:22:11:4a:51 \
                                              src_mac e4:11:22:11:4a:50 \
                                              vlan_ethtype 0x800 \
                                              vlan_id 100 \
                                              vlan_prio 0 \
                                   action vlan pop \
                                   action mirred egress redirect dev pf0vf0
```

7.5.13.3 VXLAN Encap/Decap Example

```
$ tc filter add dev pf0vf0 protocol 0x806 parent ffff: \
                                    flower \
                                               skip_sw \
                                               dst_mac e4:11:22:11:4a:51 \
                                               src_mac e4:11:22:11:4a:50 \
                                    action tunnel_key set \
                                    src_ip 20.1.12.1 \
                                    dst_ip 20.1.11.1 \
                                    id 100 \
                                    action mirred egress redirect dev vxlan100

$ tc filter add dev vxlan100 protocol 0x806 parent ffff: \
                                    flower \
                                              skip_sw \
                                              dst_mac e4:11:22:11:4a:51 \
                                              src_mac e4:11:22:11:4a:50 \
                                              enc_src_ip 20.1.11.1 \
                                              enc_dst_ip 20.1.12.1 \
                                              enc_key_id 100 \
                                              enc_dst_port 4789 \
                                   action tunnel_key unset \
                                   action mirred egress redirect dev pf0vf0
```

7.5.14 VirtIO Acceleration Through Hardware vDPA

For configuration procedure, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP^2 Direct > VirtIO Acceleration through Hardware vDPA.

7.6 Configuring Uplink MTU

To configure the port MTU while operating in SmartNIC mode, you must restrict the external host port ownership by issuing the following command on the DPU:

```
mlxprivhost -d /dev/mst/<pciconf0 device> r --disable_port_owner
```

Server cold reboot is required for this restriction to take effect.

Once the host is restricted, the port MTU is configured by changing the MTU of the uplink representor (`p0` or `p1`).

7.7 Link Aggregation

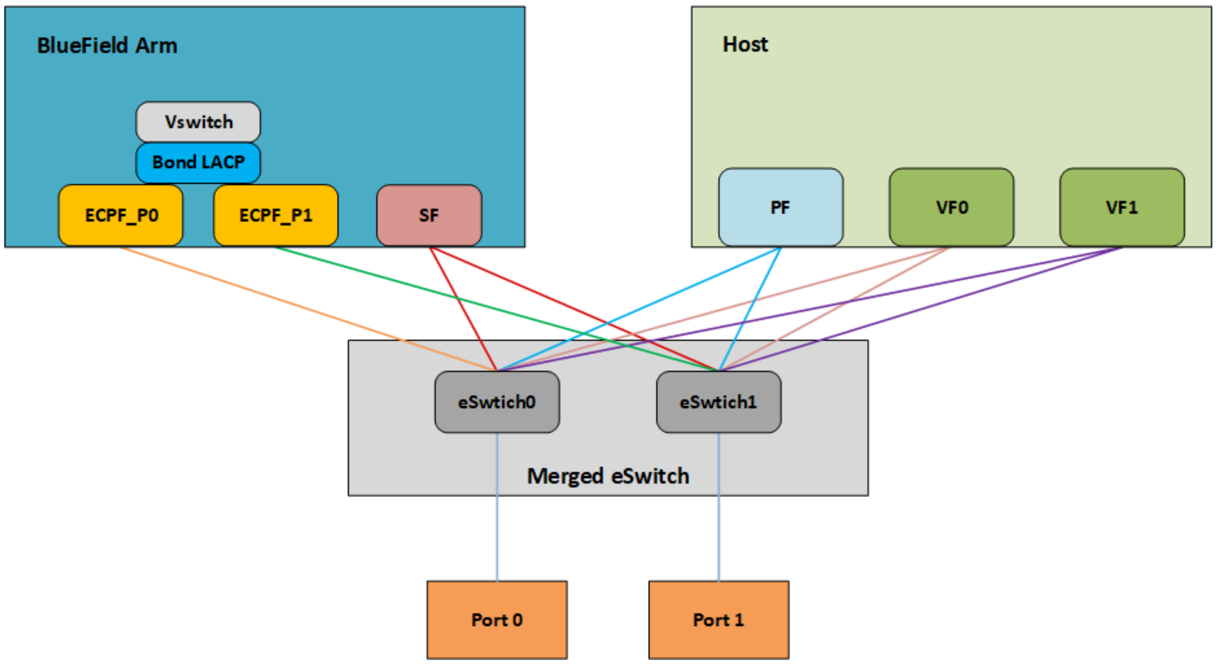
Network bonding enables combining two or more network interfaces into a single interface. It increases the network throughput, bandwidth and provides redundancy if one of the interfaces fails.

NVIDIA® BlueField® DPU has an option to configure network bonding on the Arm side in a manner transparent to the host. Under such configuration, the host would only see a single PF.

This functionality is supported when the DPU is set in embedded function ownership mode for both ports.

While LAG is being configured (starting with step 2 under section "LAG Configuration"), traffic cannot pass through the physical ports.

The diagram below describes this configuration:



7.7.1 LAG Modes

Two LAG modes are supported on BlueField:

- Queue Affinity mode
- Hash mode

7.7.1.1 Queue Affinity Mode

In this mode, packets are distributed according to the QPs.

1. To enable this mode, run:

```
$ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=0
```

Example device name: `mt41686_pciconf0`.

2. Add/edit the following field from `/etc/mellanox/mlnx-bf.conf` as follows:

```
LAG_HASH_MODE="no"
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.7.1.2 Hash Mode

In this mode, packets are distributed to ports according to the hash on packet headers.

For this mode, prerequisite steps 3 and 4 are not required.

1. To enable this mode, run:

```
$ mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=1
```

Example device name: `mt41686_pciconf0`.

2. Add/edit the following field from `/etc/mellanox/mlnx-bf.conf` as follows:

```
LAG_HASH_MODE="yes"
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.7.2 Prerequisites

1. Set the LAG mode to work with.
2. (Optional) Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

Example device name: `mt41686_pciconf0`.

Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

3. Delete any installed Scalable Functions (SFs) on the Arm side.
4. Stop the driver on the host side. Run:

```
$ systemctl stop openibd
```

5. The uplink interfaces (`p0` and `p1`) on the Arm side must be disconnected from any OVS bridge.

7.7.3 LAG Configuration

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond
$ ip link set bond0 down
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

   While LAG is being configured (starting with the next step), traffic cannot pass through the physical ports.

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down
$ ip link set p1 down
$ ip link set p0 master bond0
$ ip link set p1 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p1 up
$ ip link set bond0 up
```

   The following is an example of LAG configuration in Ubuntu:

```
# cat /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback
#p0
auto p0
iface p0 inet manual
        bond-master bond1
#
#p1
auto p1
iface p1 inet manual
        bond-master bond1
#bond1
auto bond1
iface bond1 inet static
        address 192.168.1.1
        netmask 255.255.0.0
        mtu 1500
        bond-mode 2
        bond-slaves p0 p1
        bond-miimon 100
        pre-up (sleep 2 && ifup p0) &
        pre-up (sleep 2 && ifup p1) &
```

   As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

   When in shared RQ mode (enabled by default), the uplink interfaces (`p0` and `p1`) must always stay enabled. Disabling them will break LAG support and VF-to-VF communication on same host.

For OVS configuration, the bond interface is the one that needs to be added to the OVS bridge (interfaces `p0` and `p1` should not be added). The PF representor for the first port (`pf0hpf`) of the LAG must be added to the OVS bridge. The PF representor for the second port (`pf1hpf`) would still be visible, but it should not be added to OVS bridge. Consider the following examples:

```
ovs-vsctl add-br bf-lag
ovs-vsctl add-port bf-lag bond0
ovs-vsctl add-port bf-lag pf0hpf
```

Trying to change bonding configuration in Queue Affinity mode (including bringing the subordinated interface up/down) while the host driver is loaded would cause FW syndrome and failure of the operation. Make sure to unload the host driver before altering DPU bonding configuration to avoid this.

When performing driver reload (`openibd restart`) or reboot, it is required to remove bond configuration and to reapply the configurations after the driver is fully up. Refer to steps 1-4 of "Removing LAG Configuration".

7.7.4 Removing LAG Configuration

1. If Queue Affinity mode LAG is configured (i.e., `LAG_RESOURCE_ALLOCATION=0`):
   a. Delete any installed Scalable Functions (SFs) on the Arm side.
   b. Stop driver (openibd) on the host side. Run:

      ```
      systemctl stop openibd
      ```
2. Delete the LAG OVS bridge on the Arm side. Run:

   ```
   ovs-vsctl del-br bf-lag
   ```
   This allows for later restoration of OVS configuration for non-LAG networking.
3. Stop OVS service. Run:

   ```
   systemctl stop openvswitch-switch.service
   ```
4. Run:

   ```
   ip link set bond0 down
   modprobe -rv bonding
   ```
   As a result, both of the DPU's network interfaces would be available to the host side for networking and SR-IOV.
5. For the host to be able to use the DPU ports, make sure to attach the ECPF and host representor in an OVS bridge on the Arm side. Refer to "Virtual Switch on DPU" for instructions on how to perform this.
6. Revert from `HIDE_PORT2_PF` , on the Arm side. Run:

   ```
   mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=False NUM_OF_PF=2
   ```
7. Restore default LAG settings in the DPU's firmware. Run:

   ```
   mlxconfig -d /dev/mst/<device-name> s LAG_RESOURCE_ALLOCATION=DEVICE_DEFAULT
   ```
8. Delete the following line from `/etc/mellanox/mlnx-bf.conf` on the Arm side:

```
LAG_HASH_MODE=...
```

9. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.7.5 LAG on Multi-host

Only LAG hash mode is supported with BlueField multi-host.

7.7.5.1 LAG Multi-host Prerequisites

1. Enable LAG hash mode.
2. Hide the second PF on the host. Run:

```
$ mlxconfig -d /dev/mst/<device-name> s HIDE_PORT2_PF=True NUM_OF_PF=1
```

3. Make sure NVME emulation is disabled:

```
$ mlxconfig -d /dev/mst/<device-name> s NVME_EMULATION_ENABLE=0
```

   Example device name: `mt41686_pciconf0`.
4. The uplink interfaces (`p0` and `p4`) on the Arm side, representing port0 and port1, must be disconnected from any OVS bridge. As a result, only the first PF of the DPU would be available to the host side for networking and SR-IOV.

7.7.5.2 LAG Configuration on Multi-host

1. Create the bond interface. Run:

```
$ ip link add bond0 type bond
$ ip link set bond0 down
$ ip link set bond0 type bond miimon 100 mode 4 xmit_hash_policy layer3+4
```

2. Subordinate both the uplink representors to the bond interface. Run:

```
$ ip link set p0 down
$ ip link set p4 down
$ ip link set p0 master bond0
$ ip link set p4 master bond0
```

3. Bring the interfaces up. Run:

```
$ ip link set p0 up
$ ip link set p4 up
$ ip link set bond0 up
```

4. For OVS configuration, the bond interface is the one that must be added to the OVS bridge (interfaces p0 and p4 should not be added). The PF representor, `pf0hpf`, must be added to the OVS bridge with the bond interface. The rest of the uplink representors must be added to another OVS bridge along with their PF representors. Consider the following examples:

```
ovs-vsctl add-br br-lag
ovs-vsctl add-port br-lag bond0
ovs-vsctl add-port br-lag pf0hpf
ovs-vsctl add-br br1
```

```
ovs-vsctl add-port br1 p1
ovs-vsctl add-port br1 pf1hpf
ovs-vsctl add-br br2
ovs-vsctl add-port br2 p2
ovs-vsctl add-port br2 pf2hpf
ovs-vsctl add-br br3
ovs-vsctl add-port br3 p3
ovs-vsctl add-port br3 pf3hpf
```

When performing driver reload (`openibd restart`) or reboot, you must remove bond configuration from NetworkManager, and to reapply the configurations after the driver is fully up.

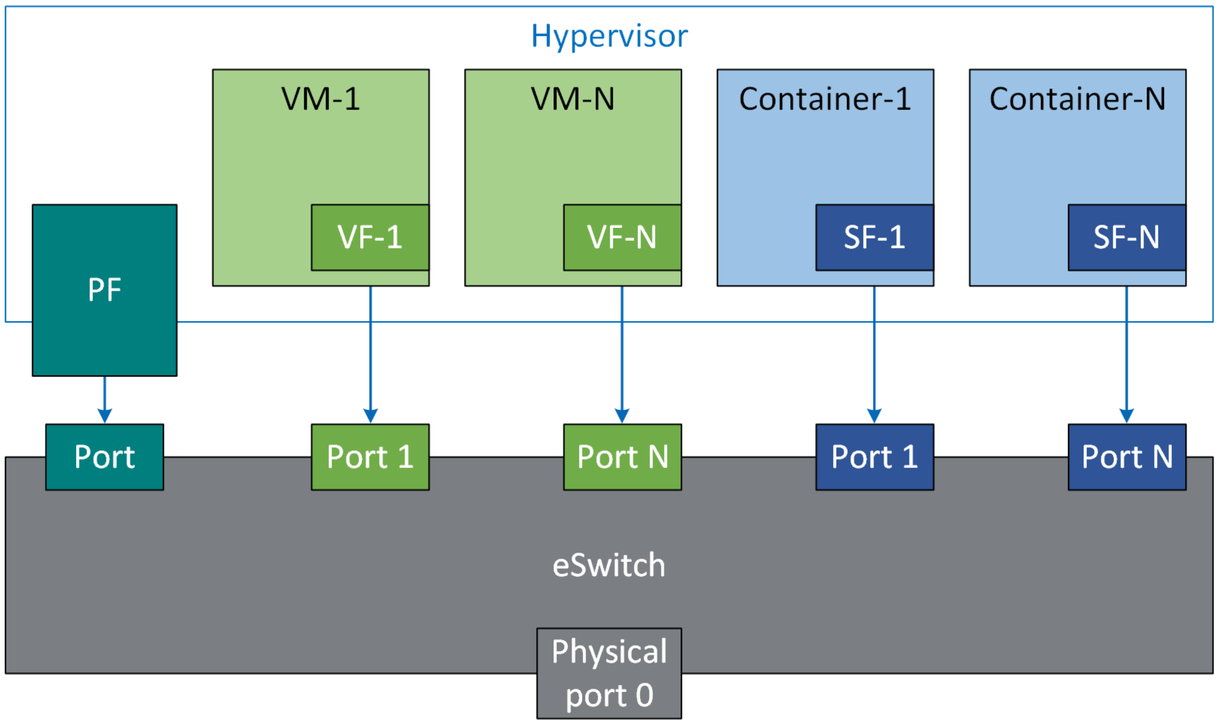
7.7.5.3 Removing LAG Configuration on Multi-host

Refer to section "Removing LAG Configuration".

7.8 Scalable Functions

A scalable function (SF) is a lightweight function that has a parent PCIe function on which it is deployed. An mlx5 SF has its own function capabilities and its own resources. This means that an SF has its own dedicated queues (txq, rxq, cq, eq) which are neither shared nor stolen from the parent PCIe function.

No special support is needed from system BIOS to use SFs. SFs co-exist with PCIe SR-IOV virtual functions. SFs do not require enabling PCIe SR-IOV.



7.8.1 Scalable Function Configuration

The following procedure offers a guide on using scalable functions with upstream Linux kernel.

7.8.1.1 Device Configuration

1. Make sure your firmware version supports SFs (20.30.1004 and above).
2. Enable SF support in device. Run:

```
$ mlxconfig -d 0000:03:00.0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1 PF_TOTAL_SF=236 PF_SF_BAR_SIZE=10
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.8.1.2 Mandatory Kernel Configuration on Host

Support for Linux kernel mlx5 SFs must be enabled as it is disabled by default.

The following two Kconfig flags must be enabled.

- MLX5_ESWITCH
- MLX5_SF

7.8.1.3 Software Control and Commands

SFs use a 4-step process as follows:

- Create
- Configure
- Deploy
- Use

SFs are managed using mlxdevm tool. It is located under directory `/opt/mellanox/iproute2/sbin/mlxdevm`.

1. Display the physical (i.e. uplink) port of the PF. Run:

```
$ devlink port show
pci/0000:03:00.0/65535: type eth netdev p0 flavour physical port 0 splittable false
```

2. Add an SF. Run:

```
$ mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 88
pci/0000:03:00.0/229409: type eth netdev eth0 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

An added SF is still not usable for the end-user application. It can only be used after configuration and activation.

SF number ≥1000 is reserved for the virtio-net controller.

When an SF is added on the external controller (e.g. DPU) users must supply the controller number. In a single host DPU case, there is only one controller starting with controller number 1.

Example of adding an SF for PF0 of external controller 1:

```
$ mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum 0 sfnum 88 controller 1
pci/0000:03:00.0/32768: type eth netdev eth6 flavour pcisf controller 1 pfnum 0 sfnum 88 splittable false
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached
```

3. Show the newly added devlink port by its port index or its representor device.

```
$ mlxdevm port show en3f0pf0sf88
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

Or:

```
$ mlxdevm port show pci/0000:03:00.0/229409
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached trust off
```

4. Set the MAC address of the SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 hw_addr 00:00:00:00:88:88
```

5. Set SF as trusted (optional). Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 trust on
pci/0000:03:00.0/229409: type eth netdev en3f0pf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:88:88 state inactive opstate detached trust on
```

A trusted function has additional privileges like the ability to update steering database.

6. Configure OVS. Run:

```
$ systemctl start openvswitch
$ ovs-vsctl add-br network1
$ ovs-vsctl add-port network1 ens3f0npf0sf88
$ ip link set dev ens3f0npf0sf88 up
```

7. Activate the SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 state active
```

Activating the SF results in creating an auxiliary device and initiating driver load sequence for netdevice, RDMA, and VDPA devices. Once the operational state is marked as attached, a driver is attached to this SF and device loading begins.

An application interested in using the SF netdevice and rdma device must monitor the RDMA and netdevices either through udev monitor or poll the sysfs hierarchy of the SF's auxiliary device.

8. By default, SF is attached to the configuration driver `mlx5_core.sf_cfg`. Users must unbind an SF from the configuration and bind it to the `mlx5_core.sf` driver to make use of it. Run:

```
$ echo mlx5_core.sf.4 > /sys/bus/auxiliary/devices/mlx5_core.sf.4/driver/unbind
$ echo mlx5_core.sf.4 > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
```

9. View the new state of the SF. Run:

```
$ mlxdevm port show en3f0pf0sf88 -jp
{
    "port": {
        "pci/0000:03:00.0/229409": {
            "type": "eth",
            "netdev": "en3f0pf0sf88",
            "flavour": "pcisf",
            "controller": 0,
            "pfnum": 0,
            "sfnum": 88,
            "function": {
                "hw_addr": "00:00:00:00:88:88",
                "state": "active",
                "opstate": "detached",
                "trust": "on"
            }
        }
    }
  }
```

10. View the auxiliary device of the SF. Run:

```
$ cat /sys/bus/auxiliary/devices/mlx5_core.sf.4/sfnum
88
```

There can be hundreds of auxiliary SF devices on the auxiliary bus. Each SF's auxiliary device contains a unique sfnum and PCI information.

11. View the parent PCI device of the SF. Run:

```
$ readlink /sys/bus/auxiliary/devices/mlx5_core.sf.1
../../../devices/pci0000:00/0000:00:00.0/0000:01:00.0/0000:02:00.0/0000:03:00.0/mlx5_core.sf.1
```

12. View the devlink instance of the SF device. Run:

```
$ devlink dev show
$ devlink dev show auxiliary/mlx5_core.sf.4
```

13. View the port and netdevice associated with the SF. Run:

```
$ devlink port show auxiliary/mlx5_core.sf.4/1
auxiliary/mlx5_core.sf.4/1: type eth netdev enp3s0f0s88 flavour virtual port 0 splittable false
```

14. View the RDMA device for the SF. Run:

```
$ rdma dev show
$ ls /sys/bus/auxiliary/devices/mlx5_core.sf.4/infiniband/
```

15. Deactivate SF. Run:

```
$ mlxdevm port function set pci/0000:03:00.0/229409 state inactive
```

Deactivating the SF triggers driver unload in the host system. Once SF is deactivated, its operational state changes to "detached". An orchestration system should poll for the operational state to be changed to "detached" before deleting the SF. This ensures a graceful hot-unplug.

16. Delete SF. Run:

```
$ mlxdevm port del pci/0000:03:00.0/229409
```

Finally, once the state is "inactive" and the operational state is "detached" the user can safely delete the SF. For faster provisioning, a user can reconfigure and active the SF again without deletion.

7.9 RDMA Stack Support on Host and Arm System

Full RDMA stack is pre-installed on the Arm Linux system. RDMA, whether RoCE or InfiniBand, is supported on BlueField® DPU in the configurations listed below.

7.9.1 Separate Host Mode

RoCE is supported from both the host and Arm system.

InfiniBand is supported on the host system.

7.9.2 Embedded CPU Mode

7.9.2.1 RDMA Support on Host

To use RoCE on a host system's PCIe PF, OVS hardware offloads must be enabled on the Arm system.

RoCE is not supported by connection tracking offload. Please refer to "Configuring Connection Tracking Offload" for a workaround for it.

7.9.2.2 RDMA Support on Arm

RoCE is unsupported on the Arm system on the PCIe PF. However, RoCE is fully supported using scalable function as explained under "Scalable Functions". Scalable functions are created by default, allowing RoCE traffic without further configuration.

InfiniBand is supported on the Arm system on the PCIe PF in this mode.

7.10 Controlling Host PF and VF Parameters

NVIDIA® BlueField® allows control over some of the networking parameters of the PFs and VFs running on the host side.

7.10.1 Setting Host PF and VF Default MAC Address

From the Arm, users may configure the MAC address of the physical function in the host. After sending the command, users must reload the NVIDIA driver in the host to see the newly configured MAC address. The MAC address goes back to the default value in the FW after system reboot.

Example:

```
$ echo "c4:8a:07:a5:29:59" > /sys/class/net/p0/smart_nic/pf/mac
$ echo "c4:8a:07:a5:29:61" > /sys/class/net/p0/smart_nic/vf0/mac
```

7.10.2 Setting Host PF and VF Link State

vPort state can be configured to Up, Down, or Follow. For example:

```
$ echo "Follow" > /sys/class/net/p0/smart_nic/pf/vport_state
```

7.10.3 Querying Configuration

To query the current configuration, run:

```
$ cat /sys/class/net/p0/smart_nic/pf/config
MAC         : e4:8b:01:a5:79:5e
MaxTxRate   : 0
State       : Follow
```

Zero signifies that the rate limit is unlimited.

7.10.4 Disabling Host Networking PFs

It is possible to not expose ConnectX networking functions to the host for users interested in using storage or VirtIO functions only. When this feature is enabled, the host PF representors (i.e. `pf0hpf` and `pf1hpf`) will not be seen on the Arm.

- Without a PF on the host, it is not possible to enable SR-IOV, so VF representors will not be seen on the Arm either
- Without PFs on the host, there can be no SFs on it

To disable host networking PFs, run:

```
mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=0
```

To reactivate host networking PFs:

- For single-port DPUs, run:

  ```
  mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=1
  ```

- For dual-port DPUs, run:

  ```
  mlxconfig -d /dev/mst/mt41686_pciconf0 s NUM_OF_PF=2
  ```

When there are no networking functions exposed on the host, the reactivation command must be run from the Arm.

Perform a BlueField system reboot for the mlxconfig settings to take effect. Refer to the "NVIDIA BlueField Reset and Reboot Procedures" troubleshooting page for instructions.

7.11 DPDK on BlueField DPU

Please refer to "NVIDIA BlueField Board Support Package" in the DPDK documentation.

7.12 BlueField SNAP

NVIDIA® BlueField® SNAP (Software-defined Network Accelerated Processing) technology enables hardware-accelerated virtualization of NVMe storage. BlueField SNAP presents networked storage as a local NVMe SSD, emulating an NVMe drive on the PCIe bus. The host OS/Hypervisor makes use of its standard NVMe-driver unaware that the communication is terminated, not by a physical drive, but by the BlueField SNAP. Any logic may be applied to the data via the BlueField SNAP framework and transmitted over the network, on either Ethernet or InfiniBand protocol, to a storage target.

BlueField SNAP combines unique hardware-accelerated storage virtualization with the advanced networking and programmability capabilities of the DPU. BlueField SNAP together with the DPU enable a world of applications addressing storage and networking efficiency and performance.

To enable BlueField SNAP, please refer to the NVIDIA BlueField-3 SNAP for NVMe and Virtio-blk documentation.

7.13 BlueField SR-IOV

The BlueField SR-IOV solution is based on asymmetric VF and enables per-ECPF and per PF control over number of VF allocation.

ECPF VFs are intended to be used in switchdev mode. Like SFs and host VFs, ECPF VFs have a representor. Representor naming for ECPF VFs start after the host VFs. For example, if the host has 32 VFs enabled, then the host VF representors are named `pf0vf0` - `pf0vf31` , and the Arm representors continue at `pf0vf32` onward.

To enable BlueField SR-IOV, apply the following configuration in the BlueField OS:

```
mlxconfig -d 03:00.0 -y s PF_NUM_OF_VF_VALID=1
```

Once `PF_NUM_OF_VF_VALID` is set, the `NUM_OF_VFS` mlxconfig option is not relevant and the user must set `PF_NUM_OF_VF` for each host and EC function. It is recommended for the number of VFs for each ECPF and each host PF be the same.

The BlueField should now support setting asymmetric VF configuration per port.

The following are examples for configuring the number of VFs per port:

1. In the BlueField, issue the following commands to configure 32 VFs per port:

   ```
   dpu> mlxconfig -d 03:00.0 -y s PF_NUM_OF_VF=32
   dpu> mlxconfig -d 03:00.1 -y s PF_NUM_OF_VF=32
   ```

The BlueField ECPF driver in the BlueField's Arm OS limits the number of VFs it supports to 32 per port.

2. In the host OS, issue the following commands to configure up to 126 VFs per port:

```
host> mlxconfig -d 03:00.0 -y s PF_NUM_OF_VF=126
host> mlxconfig -d 03:00.1 -y s PF_NUM_OF_VF=126
```

3. Perform a BlueField system reboot for the `mlxconfig` settings to take effect.
4. Create ECPF VFs:

```
echo 1 > /sys/class/net/p0/device/sriov_numvfs
```

BlueField SR-IOV VFs do not support the following legacy SRIOV functionalities:

- Virtual switch tagging (VF VLAN)
- Spoof check
- VF trust
- VF rate

7.14 Compression Acceleration

NVIDIA® BlueField® DPU supports high-speed compression acceleration. This feature allows the host to offload multiple compression/decompression jobs to the DPU.

Compress-class operations are supported in parallel to the net, vDPA, and RegEx class operations.

7.14.1 Configuring Compression Acceleration

The compression application can run either from the host or Arm.

For more information, please refer to:

- The DPDK community documentation about compression
- The mlx5 support documentation

7.15 Public Key Acceleration

NVIDIA BlueField DPU incorporates several public key acceleration (PKA) engines to offload the processor of the Arm host, providing high-performance computation of PK algorithms. BlueField's PKA is useful for a wide range of security applications. It can assist with SSL acceleration, or a secure high-performance PK signature generator/checker and certificate related operations.

BlueField's PKA software libraries implement a simple, complete framework for crypto public key infrastructure (PKI) acceleration. It provides direct access to hardware resources from the user space and makes available a number of arithmetic operations—some basic (e.g., addition and multiplication), and some complex (e.g., modular exponentiation and modular inversion)—and high-level operations such as RSA, Diffie-Hallman, Elliptic Curve Cryptography, and the Federal Digital Signature Algorithm (DSA as documented in FIPS-186) public-private key systems.

7.15.1 PKA Prerequisites

- The BlueField PKA software is intended for BlueField products with HW accelerated crypto capabilities. To verify whether your BlueField chip has crypto capabilities, look for CPU flags `aes`, `sha1`, and `sha2` in the DPU OS. For example:

```
# lscpu
...
Flags: fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
```

- BlueField bootloader must enable SMMU support to benefit from the full hardware and software capabilities. SMMU support may be enabled in UEFI menu through system configuration options.

7.15.2 PKA Use Cases

Some of the use cases for the BlueField PKA involve integrating OpenSSL software applications with BlueField's PKA hardware. The BlueField PKA dynamic engine for OpenSSL allows applications integrated with OpenSSL (e.g., StrongSwan) to accomplish a variety of security-related goals and to accelerate the cryptographic processing with the BlueField PKA hardware. OpenSSL versions ≥1.0.0, ≤1.1.1, and 3.0.2 are supported.

With CentOS 7.6, only OpenSSL 1.1 (not 1.0) works with PKA engine and keygen. Use `openssl11` with PKA engine and keygen.

The engine supports the following operations:

- RSA
- DH
- DSA
- ECDSA
- ECDH
- Random number generation that is cryptographically secure.

Up to 4096-bit keys for RSA, DH, and DSA operations are supported. Elliptic Curve Cryptography support of (nist) prime curves for 160, 192, 224, 256, 384 and 521 bits.

For example, to sign a file using BlueField's PKA engine:

```
$ openssl dgst -engine pka -sha256 -sign <privatekey> -out <signature> <filename>
```

To verify the signature, execute:

```
$ openssl dgst -engine pka -sha256 -verify <publickey> -signature <signature> <filename>
```

For further details on BlueField PKA, please refer to "PKA Driver Design and Implementation Architecture Document" and/or "PKA Programming Guide". Directions and instructions on how to integrate the BlueField PKA software libraries are provided in the README files on the Mellanox PKA GitHub.

7.16 IPsec Functionality

7.16.1 Transparent IPsec Encryption and Decryption

BlueField DPU can offload IPsec operations transparently from the host CPU. This means that the host does not need to be aware that network traffic is encrypted before hitting the wire or decrypted after coming off the wire. IPsec operations can be run on the DPU in software on the Arm cores or in the accelerator block.

7.16.2 IPsec Hardware Offload: Crypto Offload

IPsec hardware crypto offload, also known as IPsec inline offload or IPsec aware offload, enables the user to offload IPsec crypto encryption and decryption operations to the hardware, leaving the encapsulation/decapsulation task to the software.

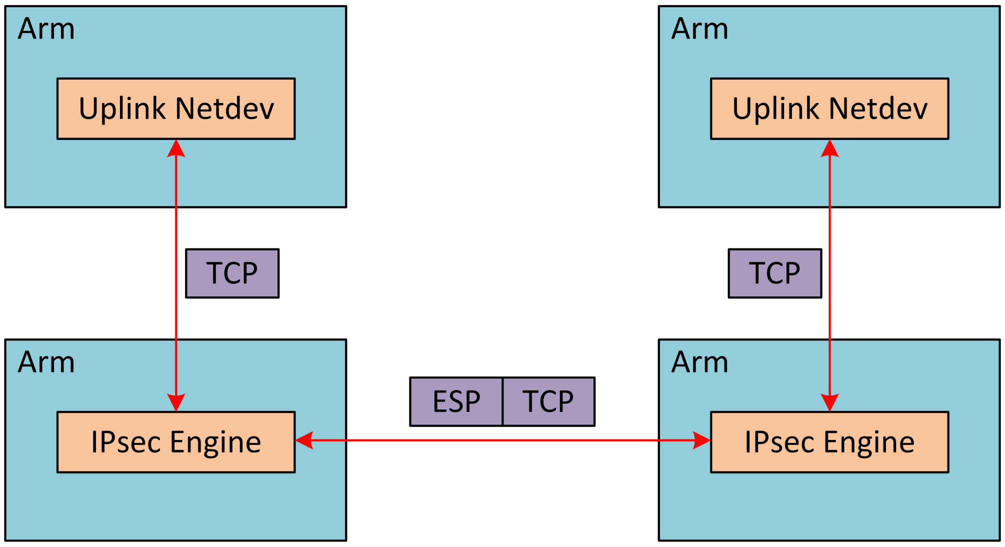
Please refer to the MLNX_OFED documentation under Features Overview and Configuration > Ethernet Network > IPsec Crypto Offload for more information on enabling and configuring this feature.

Please note that to use IPsec crypto offload with OVS, you must disable hardware offloads.

7.16.3 IPsec Hardware Offload: Packet Offload

IPSec packet offload is only supported on Ubuntu BlueField kernel 5.15

IPsec packet offload offloads both IPsec crypto and IPsec encapsulation to the hardware. IPsec packet offload is configured on the Arm via the uplink netdev. The following figure illustrates IPsec packet offload operation in hardware.



7.16.3.1 Enabling IPsec Packet Offload

Explicitly enable IPsec packet offload on the Arm cores before setting up offload-aware IPsec tunnels .

> If an OVS VXLAN tunnel configuration already exists, stop `openvswitch` service prior to performing the steps below and restart the service afterwards.

Explicitly enable IPsec full offload on the Arm cores.

1. Set `IPSEC_FULL_OFFLOAD="yes"` in `/etc/mellanox/mlnx-bf.conf`.
2. Restart IB driver (rebooting also works). Run:

   ```
   /etc/init.d/openibd restart
   ```

   > If `mlx-regex` is running:
   > a. Disable `mlx-regex`:
   >
   > ```
   > systemctl stop mlx-regex
   > ```
   > b. Restart IB driver according to the command above.
   > c. Re-enable `mlx-regex` after the restart has finished:
   >
   > ```
   > systemctl restart mlx-regex
   > ```

> To revert IPsec full offload mode, redo the procedure from step 1, only difference is to set `IPSEC_FULL_OFFLOAD="no"` in `/etc/mellanox/mlnx-bf.conf`.

> To use IPsec packet packet with strongSwan, refer to section "IPsec Packet Offload strongSwan Support".

To configure IPsec rules, please follow the instructions in MLNX_OFED documentation under Features Overview and Configuration > Ethernet Network > IPsec Crypto Offload > Configuring Security Associations for IPsec Offloads but, use "offload packet" to achieve IPsec Packet offload.

7.16.3.2 Configuring IPsec Rules with iproute2

> If you are working directly with the `ip xfrm` tool, you must use the `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec packet offload support.

The following example configures IPsec packet offload rules with local address 192.168.1.64 and remote address 192.168.1.65:

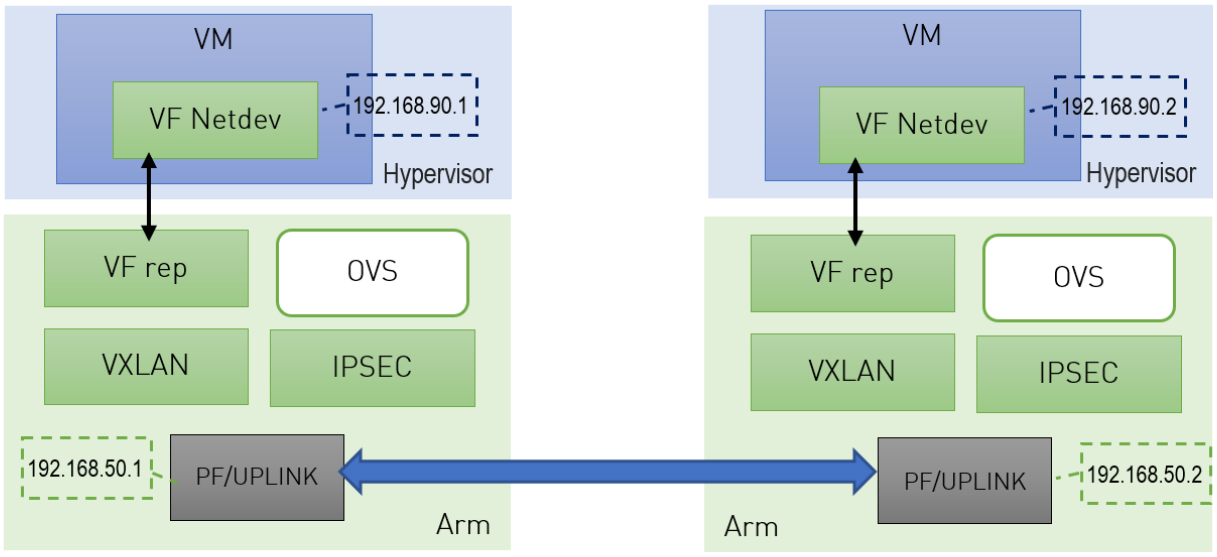
```
ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24 proto esp spi 0x4834535d reqid 0x4834535d mode transport
aead 'rfc4106(gcm(aes))' 0xc57f6f084ebf8c6a71dd9a053c2e03b94c658a9bf00dd25780e73948931d10d08058a27c 128 offload
packet dev p0 dir out sel src 192.168.1.64 dst 192.168.1.65
ip xfrm state add src 192.168.1.65/24 dst 192.168.1.64/24 proto esp spi 0x2be60844 reqid 0x2be60844 mode transport
aead 'rfc4106(gcm(aes))' 0xacca06b66489011d3c1c21f1a36d925cf7449d3aeaa6fe534446c3a8f8bd5f5fdc266589 128 offload
packet dev p0 dir in sel src 192.168.1.65 dst 192.168.1.64
sudo ip xfrm policy add src 192.168.1.64 dst 192.168.1.65 offload packet dev p0 dir out tmpl src 192.168.1.64/24
dst 192.168.1.65/24 proto esp reqid 0x4834535d mode transport
sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 offload packet dev p0 dir in tmpl src 192.168.1.65/24 dst
192.168.1.64/24 proto esp reqid 0x2be60844 mode transport
```

The numbers used by the `spi`, `reqid`, or `aead` algorithms are random. These same numbers are also used in the configuration of peer Arm. Do not confuse these numbers with source and destination IPs. The connection may fail if they are not consistent.

7.16.3.3 IPsec Packet Offload strongSwan Support

BlueField DPU supports configuring IPsec rules using strongSwan 5.9.10—appears as 5.9.10bf in the BFB which is based on upstream 5.9.10 version—which supports new fields in the `swanctl.conf` file
.

The following figure illustrates an example with two BlueField DPUs, Left and Right, operating with a secured VXLAN channel.



Support for strongSwan IPsec packet HW offload requires using VXLAN together with IPSec as shown here.

1. Follow the procedure under section "Enabling IPsec Packet Offload".
2. Follow the procedure under section "VXLAN Tunneling Offload" to configure VXLAN on Arm.

   Make sure the MTU of the PF used by VXLAN is at least 50 bytes larger than VXLAN-REP MTU.

3. Enable tc offloading. Run:

   ```
   ethtool -K <PF> hw-tc-offload on
   ```

Do not add the PF itself using "ovs-vsctl add-port" to the OVS.

7.16.3.3.1 Setting IPSec Packet Offload Using strongSwan

strongSwan configures IPSec HW packet offload using a new value added to its configuration file `swanctl.conf` (as of strongSwan version 5.9.10).

The file should be placed under "sysconfdir" which by default can be found at `/etc/swanctl/swanctl.conf`.

The terms Left (BFL) and Right (BFR) are used to identify the two nodes that communicate (corresponding with the figure under section "IPsec Packet Offload strongSwan Support").

In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```
connections {
   BFL-BFR {
      local_addrs  = 192.168.50.1
      remote_addrs = 192.168.50.2

      local {
        auth = psk
        id = host1
      }
      remote {
        auth = psk
        id = host2
      }
        children {
         bf-out {
            local_ts = 192.168.50.1/24 [udp]
            remote_ts = 192.168.50.2/24 [udp/4789]
            esp_proposals = aes128gcm128-x25519-esn
            mode = transport
            policies_fwd_out = yes
            hw_offload = packet
         }
         bf-in {
            local_ts = 192.168.50.1/24 [udp/4789]
            remote_ts = 192.168.50.2/24 [udp]
            esp_proposals = aes128gcm128-x25519-esn
            mode = transport
            policies_fwd_out = yes
            hw_offload = packet
         }
      }
      version = 2
      mobike = no
      reauth_time = 0
      proposals = aes128-sha256-x25519
   }
}

secrets {
   ike-BF {
      id-host1 = host1
      id-host2 = host2
      secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
   }
}
```

BFB installation will place two example swanctl.conf files for both Left and Right nodes (BFL.swanctl.conf and BFR.swanctl.conf respectively) in the strongSwan conf.d directory. Please move one of them manually to the other machine and edit it according to your configuration.

Note that:

- "`hw_offload = packet"` is responsible for configuring IPsec packet offload
- Packet offload support has been added to the existing `hw_offload` field and preserves backward compatibility.

For your reference:

Value	Description
`no`	Do not configure HW offload
`crypto`	Configure crypto HW offload if supported by the kernel and hardware, fail if not supported
`yes`	Same as crypto (considered legacy)
`packet`	Configure packet HW offload if supported by the kernel and hardware, fail if not supported
`auto`	Configure packet HW offload if supported by the kernel and hardware, do not fail (perform fallback to crypto or no as necessary)

Whenever the value of `hw_offload` is changed, strongSwan configuration must be reloaded.

- `[udp/4789]` is crucial for instructing strongSwan to IPSec only VXLAN communication

Packet HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

Field	Limitation
reauth_time	Ignored if set
rekey_time	Do not use. Ignored if set.
rekey_bytes	Do not use. Not supported and will fail if it is set.
rekey_packets	Use for rekeying

7.16.3.3.2 Running strongSwan Example

Notes:

- IPsec daemons are started by systemd `strongswan.service`, users must avoid using `strongswan-starter.service` as it is a legacy service and using both services at the same time leads to anomalous behavior
- Use `systemctl [start | stop | restart]` to control IPsec daemons through `strongswan.service`. For example, to restart, the command `systemctl restart strongswan.service` will effectively do the same thing as `ipsec restart`.

Do not use `ipsec` script to restart/stop/start.

If you are using the `ipsec` script, then, in order to restart or start the daemons, `openssl.cnf.orig` must be copied to openssl.cnf before performing `ipsec restart` or `ipsec start`. Then `openssl.cnf.mlnx` can be copied to `openssl.cnf` after restart or start. Failing to do so can result in errors since `openssl.cnf.mlnx` allows IPsec PK and RNG hardware offload via the OpenSSL plugin.

- On Ubuntu/Debian/Yocto, `openssl.cnf*` can be found under `/etc/ssl/`
- On CentOS, `openssl.cnf*` can be found under `/etc/pki/tls/`

- The strongSwan package installs `openssl.cnf` config files to enable hardware offload of PK and RNG operations via the OpenSSL plugin
- The OpenSSL dynamic engine is used to carry out the offload to hardware. OpenSSL dynamic engine ID is "pka".

Procedure:

1. Perform the following on Left and Right devices (corresponding with the figure under section "IPsec Packet Offload strongSwan Support").

```
# systemctl start strongswan.service
# swanctl --load-all
```

The following should appear.

```
Starting strongSwan 5.9.10bf IPsec [starter]...
no files found matching '/etc/ipsec.d/*.conf'
# deprecated keyword 'plutodebug' in config setup
# deprecated keyword 'virtual_private' in config setup
loaded ike secret 'ike-BF'
no authorities found, 0 unloaded
no pools found, 0 unloaded
loaded connection 'BFL-BFR'
successfully loaded 1 connections, 0 unloaded
```

2. Perform the actual connection on one side only (client, Left in this case).

```
# swanctl -i --child bf-in bf-out
```

The following should appear.

```
[IKE] initiating IKE_SA BFL-BFR[1] to 192.168.50.2
[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG)
N(REDIR_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (240 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (273 bytes)
[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(FRAG_SUP) N(HASH_ALG)
N(CHDLESS_SUP) N(MULT_AUTH) ]
[CFG] selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/CURVE_25519
[IKE] received 1 cert requests for an unknown ca
[IKE] authentication of 'host1' (myself) with pre-shared key
[IKE] establishing CHILD_SA bf{1}
[ENC] generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH N(USE_TRANSP) SA TSi TSr N(MULT_AUTH)
N(EAP_ONLY) N(MSG_ID_SYN_SUP) ]
[NET] sending packet: from 192.168.50.1[500] to 192.168.50.2[500] (256 bytes)
[NET] received packet: from 192.168.50.2[500] to 192.168.50.1[500] (224 bytes)
[ENC] parsed IKE_AUTH response 1 [ IDr AUTH N(USE_TRANSP) SA TSi TSr N(AUTH_LFT) ]
[IKE] authentication of 'host2' with pre-shared key successful
[IKE] IKE_SA BFL-BFR[1] established between 192.168.50.1[host1]...192.168.50.2[host2]
[IKE] scheduling reauthentication in 10027s
[IKE] maximum IKE_SA lifetime 11107s
[CFG] selected proposal: ESP:AES_GCM_16_128/NO_EXT_SEQ
[IKE] CHILD_SA bf{1} established with SPIs ce543905_i c60e98a2_o and TS 192.168.50.1/32 === 192.168.50.2/32
initiate completed successfully
```

You may now send encrypted data over the HOST VF interface (192.168.70.[1|2]) configured for VXLAN.

7.16.3.3.3 Building strongSwan

Do this only if you want to build your own BFB and would like to rebuild strongSwan.

1. Install dependencies mentioned here. libgmp-dev is missing from that list, so make sure to install that as well.
2. Git clone https://github.com/Mellanox/strongswan.git.
3. Git checkout BF-5.9.10. This branch is based on the official strongSwan 5.9.10 branch with added packaging and support for DOCA IPsec plugin (check the NVIDIA DOCA IPsec Security Gateway Application Guide for more information regarding the strongSwan DOCA plugin).
4. Run `autogen.sh` within the strongSwan repo.
5. Run the following:

```
configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc --enable-systemd
make
make install
```

Note:

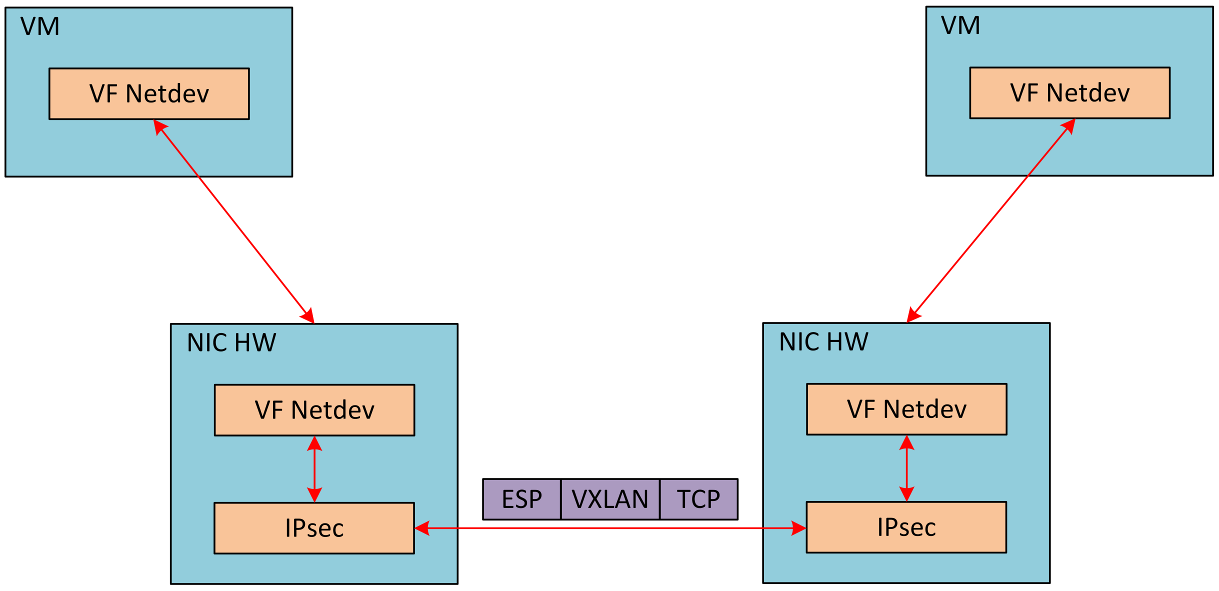
- `--enable-systemd` enables the systemd service for strongSwan present inside the GitHub repo (see step 3) at `init/systemd-starter/strongswan.service.in`.
- When building strongSwan on your own, the `openssl.cnf.mlnx` file, required for PK and RNG HW offload via OpenSSL plugin, is not installed. It must be copied over manually from github repo inside the openssl-conf directory. See section "Running Strongswan Example" for important notes.

  The `openssl.cnf.mlnx` file references PKA engine shared objects. libpka (version 1.3 or later) and openssl (version 1.1.1) must be installed for this to work.

7.16.3.4 IPsec Packet Offload and OVS Offload

IPsec packet offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

The following figure illustrates the interaction between IPsec packet offload and OVS VXLAN offload.



OVS offload and IPsec IPv6 do not work together.

7.16.4 OVS IPsec

To start the service, run:

```
systemctl start openvswitch-ipsec.service
```

Refer to section "Enabling IPsec Packet Offload" for information to prepare the IPsec packet offload environment.

7.16.4.1 Configuring IPsec Tunnel

For the sake of example, if you want to build an IPsec tunnel between two hosts with the following external IP addresses:

- `host1` - 1.1.1.1
- `host2` - 1.1.1.2

You have to first make sure `host1` and `host2` can ping each other via these external IPs.

This example will set up some variables on both hosts, set `ip1` and `ip2`:

```
# ip1=1.1.1.1
# ip2=1.1.1.2
REP=eth5
PF=p0
```

1. Set up OVS bridges in both hosts.
    a. On `Arm_1`:

    ```
    ovs-vsctl add-br ovs-br
    ovs-vsctl add-port ovs-br $REP
    ```

```
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

b. On `Arm_2`:

```
ovs-vsctl add-br ovs-br
ovs-vsctl add-port ovs-br $REP
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

> `Configuring other_config:hw-offload=true` sets IPsec packet offload. Setting it to `false` sets software IPsec. Make sure that IPsec devlink's mode is set back to `none` for software IPsec.

2. Set up IPsec tunnel. Three authentication methods are possible. Follow the steps relevant for the method that works best for your environment.

> Do not try to use more than 1 authentication method.

> After the IPsec tunnel is set up, strongSwan configuration will be automatically done.

3. Make sure the MTU of the PF used by tunnel is at least 50 bytes larger than VXLAN-REP MTU.
   a. Disable host PF as the port owner from Arm (see section "Zero-trust Mode"). Run:

   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```

   b. The MTU of the end points (`pf0hpf` in the example above) of the tunnel must be smaller than the MTU of the tunnel interfaces (`p0`) to account for the size of the tunnel headers. For example, you can set the MTU of P0 to 2000.

7.16.4.1.1 Authentication Methods

7.16.4.1.1.1 Using Pre-shared Key

> The following example uses `tun type=gre` and `dst_port=1723`. Depending on your configuration, `tun type` can be `vxlan` or `geneve` with `dst_port` 4789 or 6081 respectively.

> The following example uses `ovs-br` as the bridge name. However, this value can be any string you have chosen to create the bridge previously.

1. On `Arm_1`, run:

```
# ovs-vsctl add-port ovs-br tun -- \
            set interface tun type=gre \
                          options:local_ip=$ip1 \
                          options:remote_ip=$ip2 \
                          options:key=100 \
                          options:dst_port=1723 \
                          options:psk=swordfish
```

2. On `Arm_2`, run:

```
# ovs-vsctl add-port ovs-br tun -- \
            set interface tun type=gre \
                          options:local_ip=$ip2 \
                          options:remote_ip=$ip1 \
                          options:key=100 \
                          options:dst_port=1723 \
                          options:psk=swordfish
```

7.16.4.1.1.2 Using Self-signed Certificate

1. Generate self-signed certificates in both `host1` and `host2`, then copy the certificate of `host1` to `host2`, and the certificate of `host2` to `host1`.
2. Move both `host1-cert.pem` and `host2-cert.pem` to `/etc/swanctl/x509/`, if on Ubuntu, or `/etc/strongswan/swanctl/x509/`, if on CentOS.
3. Move the local private key to `/etc/swanctl/private`, if on Ubuntu, or `/etc/strongswan/swanctl/private`, if on CentOS. For example, for `host1`:

```
mv host1-privkey.pem /etc/swanctl/private
```

4. Set up OVS `other_config` on both sides.
   a. On `Arm_1`:

   ```
   # ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host1-cert.pem \
     other_config:private_key=/etc/swanctl/private/host1-privkey.pem
   ```

   b. On `Arm_2`:

   ```
   # ovs-vsctl set Open_vSwitch . other_config:certificate=/etc/swanctl/x509/host2-cert.pem \
     other_config:private_key=/etc/swanctl/private/host2-privkey.pem
   ```

5. Set up the tunnel.
   a. On `Arm_1`:

   ```
   # ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip1 \
     options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \
     options:remote_cert=/etc/swanctl/x509/host2-cert.pem
   # service openvswitch-switch restart
   ```

   b. On `Arm_2`:

   ```
   # ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip2 \
     options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \
     options:remote_cert=/etc/swanctl/x509/host1-cert.pem
   # service openvswitch-switch restart
   ```

7.16.4.1.1.3 Using CA-signed Certificate

1. For this method, you need all the certificates and the requests to be in the same directory during the certificate generating and signing. This example refers to this directory as `certsworkspace`.
   a. On `Arm_1`:

   ```
   # ovs-pki init --force
   # cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkspace
   ```

```
# ovs-pki req -u host1
# ovs-pki sign host1 switch
```

   b. On `Arm_2`:

```
# ovs-pki init --force
# cp /var/lib/openvswitch/pki/controllerca/cacert.pem <path_to>/certsworkspace
# ovs-pki req -u host2
# ovs-pki sign host2 switch
```

2. Move both `host1-cert.pem` and `host2-cert.pem` to `/etc/ swanctl/x509/`, if on Ubuntu, or `/etc/strongswan/swanctl/x509/`, if on CentOS.
3. Move the local private key to `/etc/swanctl/private`, if on Ubuntu, or `/etc/strongswan/swanctl/private`, if on CentOS. For example, for `host1`:

```
mv host1-privkey.pem /etc/swanctl/private
```

4. Copy `cacert.pem` to the `x509ca` directory under `/etc/swanctl/x509ca/`, if on Ubuntu, or `/etc/strongswan/swanctl/x509ca/`, if on CentOS.
5. Set up OVS `other_config` on both sides.
   a. On `Arm_1`:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host1.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host1-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

   b. On `Arm_2`:

```
# ovs-vsctl set Open_vSwitch . \
        other_config:certificate=/etc/strongswan/swanctl/x509/host2.pem \
        other_config:private_key=/etc/strongswan/swanctl/private/host2-privkey.pem \
        other_config:ca_cert=/etc/strongswan/swanctl/x509ca/cacert.pem
```

6. Set up the tunnel:
   a. On `Arm_1`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip1 \
options:remote_ip=$ip2 options:key=100 options:dst_port=4789 \ options:remote_name=host2
 #service openvswitch-switch restart
```

   b. On `Arm_2`:

```
# ovs-vsctl add-port ovs-br vxlanp0 -- set interface vxlanp0 type=vxlan options:local_ip=$ip2 \
options:remote_ip=$ip1 options:key=100 options:dst_port=4789 \ options:remote_name=host1
#service openvswitch-switch restart
```

7.16.4.2 Ensuring IPsec is Configured

Use `/opt/mellanox/iproute2/sbin/ip xfrm state show`. You should be able to see IPsec states with the keyword `in mode packet`.

7.16.4.3 Troubleshooting

For troubleshooting information, refer to Open vSwitch's official documentation.

7.17 fTPM over OP-TEE

fTMP over OP-TEE is supported on BlueField-3 only at beta level.

The Trusted Computing Group (TCG) is responsible for the specifications governing the trusted platform module (TPM). In many systems, the TPM provides integrity measurements, health checks and authentication services.

Attributes of a TPM:

- Support for bulk (symmetric) encryption in the platform
- High quality random numbers
- Cryptographic services
- Protected persistent store for small amounts of data, sticky bits, monotonic counters, and extendible registers
- Protected pseudo-persistent store for unlimited amounts of keys and data
- Extensive choice of authorization methods to access protected keys and data
- Platform identities
- Support for platform privacy
- Signing and verifying digital signatures
- Certifying the properties of keys and data
- Auditing the usage of keys and data

With TPM 2.0., the TCG creates a library specification describing all the commands or features that could be implemented and may be necessary in servers, laptops, or embedded systems. Each platform can select the features needed and the level of security or assurance required. This flexibility allows the newest TPMs to be applied to many embedded applications.

Firmware TPM (fTPM) is implemented in protected software. The code runs on the main CPU so that a separate chip is not required. While running like any other program, the code is in a protected execution environment called a trusted execution environment (TEE) which is separate from the rest of the programs running on the CPU. By doing this, secrets (e.g., private keys perhaps needed by the TPM but should not be accessed by others) can be kept in the TEE creating a more secure environment.

fTPM provides similar functionality to a chip-based TPM, but does not require extra hardware. It complies with the official TCG reference implementation of the TPM 2.0 specification. The source code of this implementation is located here.

fTPM fully supports TPM2 Tools and the TCG TPM2 Software Stack (TSS).

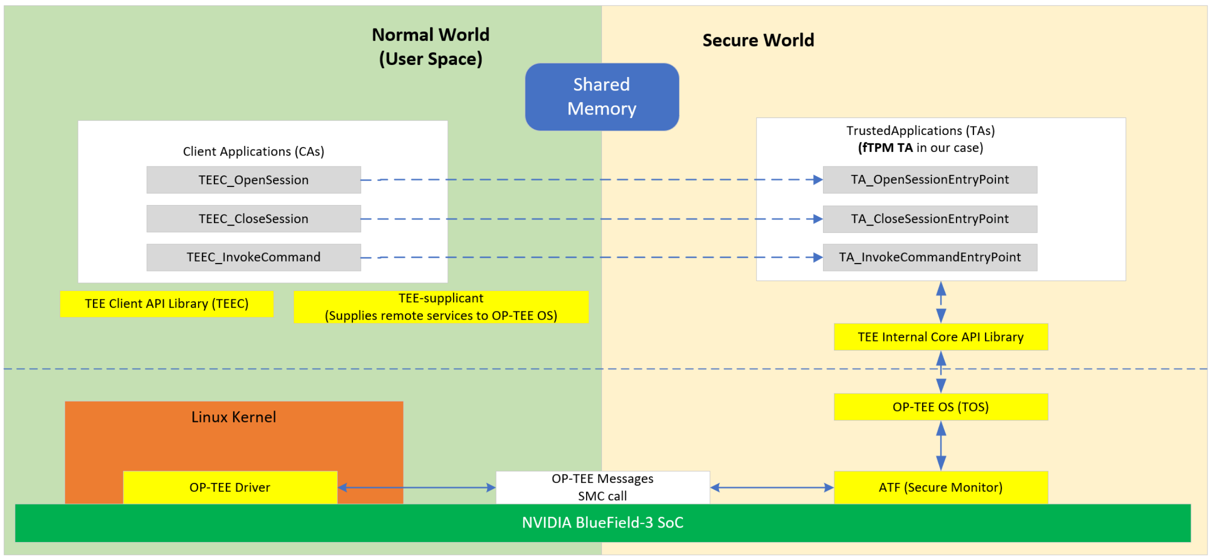
Characteristics of an fTPM:

- Emulated TPM using an isolated hardware environment
- Executes in an open-source trusted execution environment (OP-TEE)
- fTPM trusted application (TA) is part of the OP-TEE binary. This allows early access on bootup, runs only in secure DRAM.

Currently, the only TA supported is fTPM.

- fTPM is not a task waiting to be woken up. It only executes when TPM primitives are forwarded to it from the user space. It is guaranteed shielded execution via the TEE OS and, when invoked via the TEE Dispatcher, runs to completion.

The fTPM TA is the only TA NVIDIA® BlueField®-3 currently supports. Any TA loaded by OP-TEE must be signed (signing done externally) and then authenticated by OP-TEE before being allowed to load and execute.

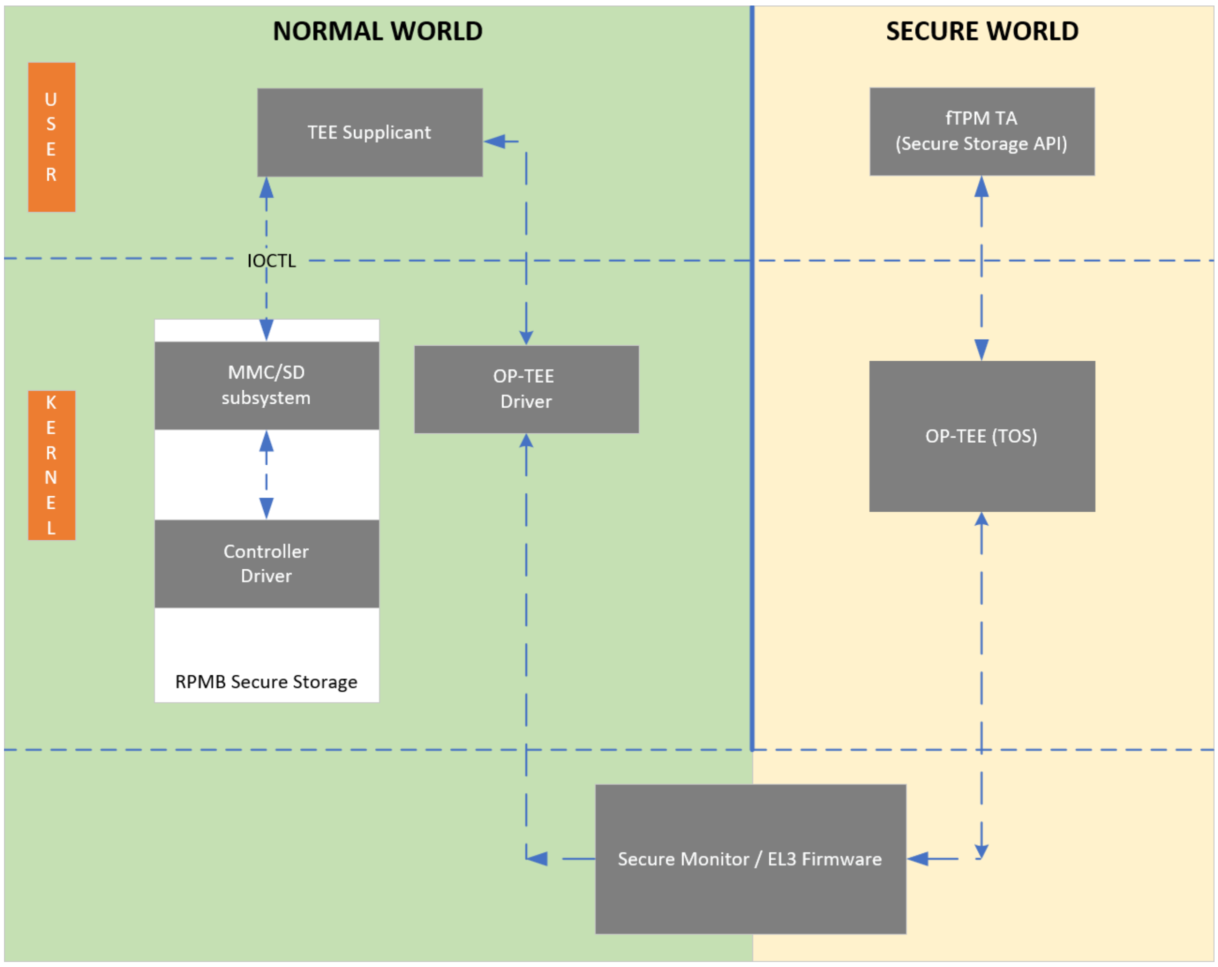


Normal World (User Space)
Secure World
Shared Memory
Client Applications (CAs)
TEEC_OpenSession
TEEC_CloseSession
TEEC_InvokeCommand
TrustedApplications (TAs) (fTPM TA in our case)
TA_OpenSessionEntryPoint
TA_CloseSessionEntryPoint
TA_InvokeCommandEntryPoint
TEE Client API Library (TEEC)
TEE-supplicant (Supplies remote services to OP-TEE OS)
TEE Internal Core API Library
Linux Kernel
OP-TEE Driver
OP-TEE Messages SMC call
OP-TEE OS (TOS)
ATF (Secure Monitor)
NVIDIA BlueField-3 SoC

A replay-protected memory block (RPMB) is provided as a means for a system to store data to the specific memory area in an authenticated and replay-protected manner, making it readable and writable only after a successful authentication read/write accesses. The RPMB is a dedicated partition available on the eMMC, which makes it possible to store and retrieve data with integrity and authenticity support. A signed access to an RPMB is supported by first programming authentication key information to the eMMC memory (shared secret). The RPMB authentication key is programmed into the DPU at manufacturing time.

RPMB features a 4MB partition secure storage for BlueField-3.

There is no eMMC controller driver in OP-TEE. All device operations have to go through the normal world via the TEE-supplicant daemon, which relies on the Linux kernel's ioctl interface to access the device. All writes to the RPMB are atomic, authenticated, and encrypted. The RPMB partition stores data in an authenticated, replay-protected manner, making it a perfect complement to fTPM for storing and protecting data.



7.17.1 Enabling OP-TEE on BlueField-3

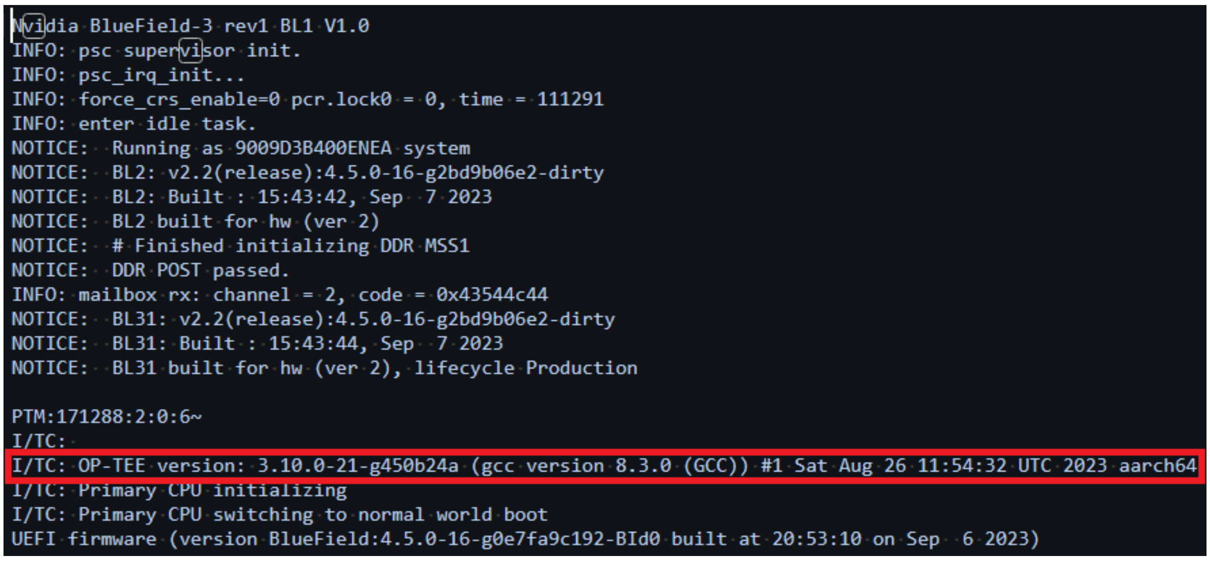
Enable OP-TEE in the UEFI menu:

1. ESC into the UEFI on DPU boot.
2. Navigate to Device Manager > System Configuration.
3. Check "Enable OP-TEE".
4. Save the change and reset/reboot.
5. Upon reboot OP-TEE is enabled.

OP-TEE is essentially dormant (does not have an OS scheduler) and reacts to external inputs.

7.17.2 Verifying BlueField-3 is Running OP-TEE

Users can see the OP-TEE version during BlueField-3 DPU boot:



```
Nvidia BlueField-3 rev1 BL1 V1.0
INFO: psc supervisor init.
INFO: psc_irq_init...
INFO: force_crs_enable=0 pcr.lock0 = 0, time = 111291
INFO: enter idle task.
NOTICE:  Running as 9009D3B400ENEA system
NOTICE:  BL2: v2.2(release):4.5.0-16-g2bd9b06e2-dirty
NOTICE:  BL2: Built : 15:43:42, Sep  7 2023
NOTICE:  BL2 built for hw (ver 2)
NOTICE:  # Finished initializing DDR MSS1
NOTICE:  DDR POST passed.
INFO: mailbox rx: channel = 2, code = 0x43544c44
NOTICE:  BL31: v2.2(release):4.5.0-16-g2bd9b06e2-dirty
NOTICE:  BL31: Built : 15:43:44, Sep  7 2023
NOTICE:  BL31 built for hw (ver 2), lifecycle Production

PTM:171288:2:0:6~
I/TC: 
I/TC: OP-TEE version: 3.10.0-21-g450b24a (gcc version 8.3.0 (GCC)) #1 Sat Aug 26 11:54:32 UTC 2023 aarch64
I/TC: Primary CPU initializing
I/TC: Primary CPU switching to normal world boot
UEFI firmware (version BlueField:4.5.0-16-g0e7fa9c192-BId0 built at 20:53:10 on Sep  6 2023)
```

The following indicators should all be present if fTPM over OP-TEE is enabled:

- Check "dmesg" for the OP-TEE driver initializing

```
root@localhost ~]# dmesg | grep tee
[    5.646578] optee: probing for conduit method.
[    5.653282] optee: revision 3.10 (450b24ac)
[    5.653991] optee: initialized driver
```

- Verify that the following kernel modules are loaded (running):

```
[root@localhost ~]# lsmod | grep tee
tpm_ftpm_tee           16384  0
optee                  49152  1
tee                    49152  3 optee,tpm_ftpm_tee
```

- Verify that the proper devices are created/available (4 in total):

```
[root@localhost ~]# ls -l /dev/tee*
crw------- 1 root root 234,  0 Sep  8 18:24 /dev/tee0
crw------- 1 root root 234, 16 Sep  8 18:24 /dev/teepriv0

[root@localhost ~]# ls -l /dev/tpm*
crw-rw---- 1 tss root  10,   224 Sep  8 18:24 /dev/tpm0
crw-rw---- 1 tss tss  252, 65536 Sep  8 18:24 /dev/tpmrm0
```

- Verify that the required processes are running (3 in total):

```
[root@localhost ~]# ps axu | grep tee
root         707  0.0  0.0  76208  1372 ?        Ssl  14:42   0:00 /usr/sbin/tee-supplicant
root         715  0.0  0.0      0     0 ?        I<   14:42   0:00 [optee_bus_scan]

[root@localhost ~]# ps axu | grep tpm
root         124  0.0  0.0      0     0 ?        I<   18:24   0:00 [tpm_dev_wq]
```

7.18 QoS Configuration

To learn more about port QoS configuration, refer to this community post.

When working in Embedded Host mode, using `mlnx_qos` on both the host and Arm will result with undefined behavior. Users must only use `mlnx_qos` from the Arm. After changing the QoS settings from Arm, users must restart the mlx5 driver on host.

When configuring QoS using DCBX, the `lldpad` service from the DPU side must be disabled if the configurations are not done using tools other than `lldpad`.

This section explains how to configure QoS group and settings using devlink located under `/opt/mellanox/iproute2/sbin/`. It is applicable to host PF/VF and Arm side SFs. The following uses VF as example.

The settings of a QoS group include creating/deleting a QoS group and modifying its `tx_max` and `tx_share` values. The settings of VF QoS include modifying its `tx_max` and `tx_share` values, assigning a VF to a QoS group, and unassigning a VF from a QoS group. This section focuses on the configuration syntax.

Please refer to section "Limit and Bandwidth Share Per VF" in the MLNX_OFED User Manual for detailed explanation on vPort QoS behaviors.

7.18.1 devlink port function rate add

	`devlink port function rate add <DEV>/<GROUP_NAME>` Adds a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format
Example	This command adds a new QoS group named `12_group` under device `pci/0000:03:00.0`: `devlink port function rate add pci/0000:03:00.0/12_group`	
Notes		

7.18.2 devlink port function rate del

	`devlink port function rate del <DEV>/<GROUP_NAME>` Deletes a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format
Example	This command deletes QoS group `12_group` from device `pci/0000:03:00.0`: `devlink port function rate del pci/0000:03:00.0/12_group`	
Notes		

7.18.3 devlink port function rate set tx_max tx_share

	`devlink port function rate set {<DEV>/<GROUP_NAME> | <DEV>/<PORT_INDEX>} tx_max <TX_MAX> [tx_share <TX_SHARE>]` Sets `tx_max` and `tx_share` for QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to operate on
	DEV/PORT_INDEX	Specifies the devlink port to operate on
	TX_MAX	`tx_max` bandwidth in MB/s
	TX_SHARE	tx_share bandwidth in MB/s

Example	This command sets `tx_max` to 2000MB/s and `tx_share` to 500MB/s for the `12_group` QoS group:
	`devlink port function rate set pci/0000:03:00.0/12_group tx_max 2000MBps tx_share 500MBps`
	This command sets `tx_max` to 2000MB/s and `tx_share` to 500MB/s for the VF represented by port index 196609:
	`devlink port function rate set pci/0000:03:00.0/196609 tx_max 200MBps tx_share 50MBps`
	This command displays a mapping between VF devlink ports and netdev names:
	`$ devlink port`
	In the output of this command, VFs are indicated by `flavour pcivf`.	

Notes		

7.18.4 devlink port function rate set parent

	`devlink port function rate set <DEV>/<PORT_INDEX> {parent <PARENT_GROUP_NAME>}` Assigns devlink port to a QoS group.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on
	PARENT_GROUP_NAME	parent group name in string format
Example	This command assigns this function to the QoS group `12_group`: `devlink port function rate set pci/0000:03:00.0/196609 parent 12_group`	
Notes		

7.18.5 devlink port function rate set noparent

	`devlink port function rate set <DEV>/<PORT_INDEX> noparent` Ungroups a devlink port.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on
Example	This command ungroups this function: `devlink port function rate set pci/0000:03:00.0/196609 noparent`	
Notes		

7.18.6 devlink port function rate show

	`devlink port function rate show [<DEV>/<GROUP_NAME> | <DEV>/<PORT_INDEX>]` Displays QoS information QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to display
	DEV/PORT_INDEX	Specifies the devlink port to display

Example	This command displays the QoS info of all QoS groups and devlink ports on the system:
	 `devlink port function rate show`
	`pci/0000:03:00.0/12_group type node tx_max 2000MBps tx_share 500MBps`
	`pci/0000:03:00.0/196609 type leaf tx_max 200MBps tx_share 50MBps parent 12_group`
	 This command displays QoS info of `12_group`:
	 `devlink port function rate show pci/0000:03:00.0/12_group`
	`pci/0000:03:00.0/12_group type node tx_max 2000MBps tx_share 500MBps`	

Notes	If a QoS group name or devlink port are not specified, all QoS groups and devlink ports are displayed.	

7.19 Virtio-net Emulated Devices

For information on virtio-net emulation, please refer to NVIDIA BlueField Virtio-net documentation.

7.20 Shared RQ Mode

When creating 1 send queue (SQ) and 1 receive queue (RQ), each representor consumes ~3MB memory per single channel. Scaling this to the desired 1024 representors (SFs and/or VFs) would require ~3GB worth of memory for single channel. A major chunk of the 3MB is contributed by RQ allocation (receive buffers and SKBs). Therefore, to make efficient use of memory, shared RQ mode is implemented so PF/VF/SF representors share receive queues owned by the uplink representor.

The feature is enabled by default. To disable it:

1. Edit the field `ALLOW_SHARED_RQ` in `/etc/mellanox/mlnx-bf.conf` as follows:

   ```
   ALLOW_SHARED_RQ="no"
   ```

2. Restart the driver. Run:

   ```
   /etc/init.d/openibd restart
   ```

To connect from the host to BlueField in shared RQ mode, please refer to section Verifying Connection from Host to BlueField.

> PF/VF representor to PF/VF communication on the host is not possible.

The following behavior is observed in shared RQ mode:

- It is expected to see a `0` in the `rx_bytes` and `rx_packets` and valid `vport_rx_packets` and `vport_rx_bytes` after running traffic. Example output:

  ```
  # ethtool -S pf0hpf
  NIC statistics:
       rx_packets: 0
       rx_bytes: 0
       tx_packets: 66946
       tx_bytes: 8786869
       vport_rx_packets: 546093
       vport_rx_bytes: 321100036
       vport_tx_packets: 549449
       vport_tx_bytes: 321679548
  ```

- Ethtool usage – in this mode, it is not possible to change/set the ring or coalesce parameters for the RX side using ethtool. Changing channels also only affects the TX side.

8 Troubleshooting and How-Tos

- NVIDIA BlueField Reset and Reboot Procedures
- RShim Troubleshooting and How-Tos
- Connectivity Troubleshooting
- Performance Troubleshooting
- PCIe Troubleshooting and How-Tos
- SR-IOV Troubleshooting
- eSwitch Troubleshooting
- Isolated Mode Troubleshooting and How-Tos
- General Troubleshooting
- Installation Troubleshooting and How-Tos

8.1 NVIDIA BlueField Reset and Reboot Procedures

8.1.1 BlueField System Reboot

This section describes the necessary operations to load new NIC firmware, following NVIDIA® BlueField® NIC firmware update. This procedure deprecates the need for full server power cycle.

The following steps are executed in the BlueField OS:

1. Issue a query command to ascertain whether BlueField system reboot is supported by your environment:

   ```
   mlxfwreset -d 03:00.0 q
   ```

   If the output includes the following lines, proceed to step 2:

   ```
   3: Driver restart and PCI reset                                   -Supported (default)
   ...
   1: Driver is the owner                                            -Supported (default)
   ```

   If it says `Not Supported` instead, then proceed to the instructions under section "BlueField System-level Reset".

2. Issue a BlueField system reboot:

   ```
   mlxfwreset -d 03:00.0 -y -l 3 --sync 1 r
   ```

8.1.2 BlueField System-level Reset

This section describes the necessary system-level reset following firmware configuration changes.

The two methods for performing BlueField system-level reset are described in the following subsection. Each method is designed to support different host platforms, in which host OS/CPUs and PCIe slots may have uniform or separate power control.

In each approach, the procedure can be performed through various methods, according to resource availability and support in the user's environment.

8.1.2.1 System-level Reset for BlueField in DPU Mode with Minimal Host OS Downtime

The following is the high-level flow of the procedure:

1. Graceful shutdown of BlueField Arm cores.
2. Query BlueField state to affirm shutdown reached.

   > In systems with multiple BlueField networking platforms, repeat steps 1 and 2 for all devices before proceeding.

3. Warm reboot the server.

Step by step process:

1. Graceful shutdown of BlueField Arm cores.

   > This operation is expected to finish within 15 seconds.

   Options:
   - From the BlueField OS:

     ```
     shutdown -h now
     ```

     Or:

     ```
     mlxfwreset -d /dev/mst/mt*pciconf0 -l 1 -t 4 --sync 0 r
     ```
   - From the host OS:

     > Not relevant when the BlueField is operating in Zero-Trust Mode.

     ```
     mlxfwreset -d <mst-device> -l 1 -t 4 r
     ```
   - Using the BlueField BMC:

     ```
     ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> power soft
     ```

     Or using Redfish (BlueField-3 and above):

     ```
     curl -k -u root:<password> -H "Content-Type: application/json" -X POST https://<bmc_ip>/redfish/v1/
     Systems/Bluefield/Actions/ComputerSystem.Reset -d '{"ResetType": "GracefulShutdown"}'
     ```
2. Query BlueField state. Options:
   - From the host OS:

     > Not relevant when the BlueField is operating in Zero-Trust Mode.

```
echo DISPLAY_LEVEL 2 > /dev/rshim0/misc
cat /dev/rshim0/misc
```

Expected output:

```
INFO[BL31]: System Off
```

- Utilizing the BlueField BMC:

```
ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> raw 0x32 0xA3
```

Expected output: `06`.

3. Warm reboot the server:
   - From the host OS:

```
mlxfwreset -d <mst-device> -l 4 r
```

> If multiple DPUs are present in the host, this command must run only once. In this case, the MST device can be of any of the DPUs for which the reset is necessary and participated in step 1.

Or:

```
reboot
```

> For external hosts which do not toggle PERST# in their standard reboot command, use the `mlxfwreset` option.

8.1.2.2 System-level Reset for BlueField in DPU Mode where Host is Down Throughout the Process

This procedure is only relevant to server platforms that have separate power control for PCIe slot and CPUs in which the BlueField is provided power while host OS/CPUs may be in shutdown or similar standby state.

The following is the high-level flow of the procedure:

1. Graceful shutdown of host OS or similar CPU standby.
2. Graceful shutdown of BlueField Arm cores.
3. Query BlueField state to affirm shutdown reached.
4. Full BlueField Reset
5. Query BlueField state to affirm operational state reached

> In systems with multiple BlueField networking platforms, repeat steps 1 through 5 for all devices before proceeding.

6. Power on the server.

Step by step process:

1. Graceful shutdown of host OS by any means preferable.
2. Graceful shutdown of BlueField Arm cores.

   This step normally takes up to 15 seconds to complete.

   - From the BlueField OS:

     ```
     shutdown -h now
     ```

   - Utilizing the BlueField BMC:
     - Using IPMI:

       ```
       ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> power soft
       ```

     - Using Redfish (for BlueField-3 and above):

       ```
       curl -k -u root:<password> -H "Content-Type: application/json" -X POST https://<bmc_ip>/
       redfish/v1/Systems/Bluefield/Actions/ComputerSystem.Reset -d '{"ResetType":
       "GracefulShutdown"}'
       ```

3. Query the BlueField's state utilizing the BlueField BMC:

   ```
   ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> raw 0x32 0xA3
   ```

   Expected output: `06` .
4. Perform BlueField hard reset utilizing the BlueField BMC:

   This step takes up to 2 minutes to complete.

   - Using IPMI:

     ```
     ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> power cycle
     ```

   - Using Redfish (for BlueField-3 and above):

     ```
     curl -k -u root:<password> -H "Content-Type: application/json" -X POST https://<bmc_ip>/redfish/v1/
     Systems/Bluefield/Actions/ComputerSystem.Reset -d '{"ResetType" : "PowerCycle"}'
     ```

5. Query BlueField operational state utilizing the BlueField BMC:

   At this point, the BlueField is expected to be operational.

   ```
   ipmitool -C 17 -I lanplus -H <bmc_ip> -U root -P <password> raw 0x32 0xA3
   ```

   Expected output: `05` .
6. Power on/boot up the host OS.

8.1.2.3 System-level Reset for BlueField in NIC Mode

Perform warm reboot of the host OS:

```
mlxfwreset -d <mst-device> -l 4 r
```

Or:

```
reboot
```

For external hosts which do not toggle PERST# in their standard reboot command, use the `mlxfwreset` option.

8.2 RShim Troubleshooting and How-Tos

8.2.1 Another backend already attached

Several generations of BlueField DPUs are equipped with a USB interface in which RShim can be routed, via USB cable, to an external host running Linux and the RShim driver.

In this case, typically following a system reboot, the RShim over USB prevails and the DPU host reports RShim status as "`another backend already attached`". This is correct behavior, since there can only be one RShim backend active at any given time. However, this means that the DPU host does not own RShim access.

To reclaim RShim ownership safely:

1. Stop the RShim driver on the remote Linux. Run:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```

2. Restart RShim on the DPU host. Run:

   ```
   systemctl enable rshim
   systemctl start rshim
   ```

The "`another backend already attached`" scenario can also be attributed to the RShim backend being owned by the BMC in DPUs with integrated BMC. This is elaborated on further down on this page.

8.2.2 RShim driver not loading

Verify whether your DPU features an integrated BMC or not. Run:

```
# sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vvv | grep "Product Name"
```

Example output for DPU with integrated BMC:

```
Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC, Crypto and Secure Boot Enabled, 16GB on-board
DDR, 1GbE OOB management, Tall Bracket, FHHL
```

If your DPU has an integrated BMC, refer to RShim driver not loading on host with integrated BMC.

If your DPU does not have an integrated BMC, refer to RShim driver not loading on host on DPU without integrated BMC.

8.2.2.1 RShim driver not loading on DPU with integrated BMC

8.2.2.1.1 RShim driver not loading on host

1. Access the BMC via the RJ45 management port of the DPU.
2. Delete RShim on the BMC:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```
3. Enable RShim on the host:

   ```
   systemctl enable rshim
   systemctl start rshim
   ```
4. Restart RShim service. Run:

   ```
   sudo systemctl restart rshim
   ```
   If RShim service does not launch automatically, run:

   ```
   sudo systemctl status rshim
   ```
   This command is expected to display "`active (running)`".
5. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        pcie-04:00.2 (ro)
   ```
   This output indicates that the RShim service is ready to use.

8.2.2.1.2 RShim driver not loading on BMC

1. Verify that the RShim service is not running on host. Run:

   ```
   systemctl status rshim
   ```
   If the output is `active`, then it may be presumed that the host has ownership of the RShim.
2. Delete RShim on the host. Run:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```
3. Enable RShim on the BMC. Run:

   ```
   systemctl enable rshim
   systemctl start rshim
   ```
4. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        usb-1.0
   ```

This output indicates that the RShim service is ready to use.

8.2.2.2 RShim driver not loading on host on DPU without integrated BMC

1. Download the suitable DEB/RPM for RShim (management interface for DPU from the host) driver.
2. Reinstall RShim package on the host.
   - For Ubuntu/Debian, run:

     ```
     sudo dpkg --force-all -i rshim-<version>.deb
     ```
   - For RHEL/CentOS, run:

     ```
     sudo rpm -Uhv rshim-<version>.rpm
     ```
3. Restart RShim service. Run:

   ```
   sudo systemctl restart rshim
   ```
   If RShim service does not launch automatically, run:

   ```
   sudo systemctl status rshim
   ```
   This command is expected to display "`active (running)`".
4. Display the current setting. Run:

   ```
   # cat /dev/rshim<N>/misc | grep DEV_NAME
   DEV_NAME        pcie-04:00.2 (ro)
   ```
   This output indicates that the RShim service is ready to use.

8.2.3 Change ownership of RShim from NIC BMC to host

1. Verify that your card has BMC. Run the following on the host:

   ```
   # sudo sudo lspci -s $(sudo lspci -d 15b3: | head -1 | awk '{print $1}') -vvv |grep "Product Name"
   Product Name: BlueField-2 DPU 25GbE Dual-Port SFP56, integrated BMC, Crypto and Secure Boot Enabled, 16GB
   on-board DDR, 1GbE OOB management, Tall Bracket, FHHL
   ```
   The product name is supposed to show "integrated BMC" .
2. Access the BMC via the RJ45 management port of the DPU.
3. Delete RShim on the BMC:

   ```
   systemctl stop rshim
   systemctl disable rshim
   ```
4. Enable RShim on the host:

   ```
   systemctl enable rshim
   systemctl start rshim
   ```
5. Restart RShim service. Run:

```
sudo systemctl restart rshim
```

If RShim service does not launch automatically, run:

```
sudo systemctl status rshim
```

This command is expected to display "`active (running)`".

6. Display the current setting. Run:

```
# cat /dev/rshim<N>/misc | grep DEV_NAME
DEV_NAME        pcie-04:00.2 (ro)
```

This output indicates that the RShim service is ready to use.

8.2.4 How to support multiple DPUs on the host

For more information, refer to section "RShim Multiple Board Support".

8.2.5 BFB installation monitoring

The BFB installation flow can be traced using various interfaces:

- From the host:
  - RShim console (`/dev/rshim0/console`)
  - RShim log buffer (`/dev/rshim0/misc`); also included in bfb-install's output
  - UART console (`/dev/ttyUSB0`)
- From the BMC console:
  - SSH to the BMC and run `obmc-console-client`

    Additional information about BMC interfaces is available in BMC software documentation

- From the DPU:
  - `/root/<OS>.installation.log` available on the DPU OS after installation

8.3 Connectivity Troubleshooting

8.3.1 Connection (ssh, screen console) to the BlueField is lost

The UART cable in the Accessories Kit (OPN: MBF20-DKIT) can be used to connect to the DPU console and identify the stage at which BlueField is hanging.

Follow this procedure:

1. Connect the UART cable to a USB socket, and find it in your USB devices.

```
sudo lsusb
Bus 002 Device 003: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC
```

For more information on the UART connectivity, please refer to the DPU's hardware user guide under Supported Interfaces > Interfaces Detailed Description > NC-SI Management Interface.

It is good practice to connect the other end of the NC-SI cable to a different host than the one on which the BlueField DPU is installed.

2. Install the minicom application.
   - For CentOS/RHEL:

     ```
     sudo yum install minicom -y
     ```

   - For Ubuntu/Debian:

     ```
     sudo apt-get install minicom
     ```

3. Open the minicom application.

   ```
   sudo minicom -s -c on
   ```

4. Go to "Serial port setup"
5. Enter "F" to change "Hardware Flow control" to NO
6. Enter "A" and change to `/dev/ttyUSB0` and press Enter
7. Press ESC.
8. Type on "Save setup as dfl"
9. Exit minicom by pressing Ctrl + a + z.

   ```
   +-----------------------------------------------------------------------+
   | A -    Serial Device      : /dev/ttyUSB0                              |
   |                                                                       |
   | C -   Callin Program      :                                           |
   | D -  Callout Program      :                                           |
   | E -    Bps/Par/Bits       : 115200 8N1                                |
   | F - Hardware Flow Control : No                                        |
   | G - Software Flow Control : No                                        |
   |                                                                       |
   |    Change which setting?                                              |
   +-----------------------------------------------------------------------+
   ```

8.3.2 Driver not loading in host server

What this looks like in dmsg:

```
[275604.216789] mlx5_core 0000:af:00.1: 63.008 Gb/s available PCIe bandwidth, limited by 8 GT/s x8 link at
0000:ae:00.0 (capable of 126.024 Gb/s with 16 GT/s x8 link)
[275624.187596] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
100s
[275644.152994] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
79s
[275664.118404] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
59s
[275684.083806] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
39s
[275704.049211] mlx5_core 0000:af:00.1: wait_fw_init:316:(pid 943): Waiting for FW initialization, timeout abort in
19s
[275723.954752] mlx5_core 0000:af:00.1: mlx5_function_setup:1237:(pid 943): Firmware over 120000 MS in pre-
initializing state, aborting
[275723.968261] mlx5_core 0000:af:00.1: init_one:1813:(pid 943): mlx5_load_one failed with error code -16
[275723.978578] mlx5_core: probe of 0000:af:00.1 failed with error -16
```

The driver on the host server is dependent on the Arm side. If the driver on Arm is up, then the driver on the host server will also be up.

Please verify that:

- The driver is loaded in the BlueField (Arm)
- The Arm is booted into OS
- The Arm is not in UEFI Boot Menu
- The Arm is not hanged

Then:

1. Perform a graceful shutdown and a power cycle on the host server.
2. If the problem persists, reset nvconfig (`sudo mlxconfig -d /dev/mst/<device> -y reset`) and perform a BlueField system reboot.

   If your BlueField is VPI capable, please be aware that this configuration will reset the link type on the network ports to IB. To change the network port's link type to Ethernet, run:

   ```
   sudo mlxconfig -d <device> s LINK_TYPE_P1=2 LINK_TYPE_P2=2
   ```

   This configuration change requires performing a BlueField system reboot.
3. If this problem still persists, please make sure to install the latest bfb image and then restart the driver in host server. Please refer to "Upgrading NVIDIA BlueField DPU Software" for more information.

8.3.3 No connectivity between network interfaces of source host to destination device

Verify that the bridge is configured properly on the Arm side.

The following is an example for default configuration:

```
$ sudo ovs-vsctl show
f6740bfb-0312-4cd8-88c0-a9680430924f
    Bridge ovsbr1
        Port pf0sf0
            Interface pf0sf0
        Port p0
            Interface p0
        Port pf0hpf
            Interface pf0hpf
        Port ovsbr1
            Interface ovsbr1
                type: internal
    Bridge ovsbr2
        Port p1
            Interface p1
        Port pf1sf0
            Interface pf1sf0
        Port pf1hpf
            Interface pf1hpf
        Port ovsbr2
            Interface ovsbr2
                type: internal
    ovs_version: "2.14.1"
```

If no bridge configuration exists, please refer to "Virtual Switch on BlueField".

8.3.4 Uplink in Arm down while uplink in host server up

Please check that the cables are connected properly into the network ports of the DPU and the peer device.

8.4 Performance Troubleshooting

8.4.1 Degradation in performance

Degradation in performance indicates that openvswitch may not be offloaded.

Verify offload state. Run:

```
# ovs-vsctl get Open_vSwitch . other_config:hw-offload
```

- If `hw-offload = true` - Fast Pass is configured (desired result)
- If `hw-offload = false` - Slow Pass is configured

If `hw-offload = false`:

- For RHEL/CentOS, run:

  ```
  # ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
  # systemctl restart openvswitch;
  # systemctl enable openvswitch;
  ```

- Ubuntu/Debian:

  ```
  # ovs-vsctl set Open_vSwitch . other_config:hw-offload=true;
  # /etc/init.d/openvswitch-switch restart
  ```

8.5 PCIe Troubleshooting and How-Tos

8.5.1 Insufficient power on the PCIe slot error

If the error "insufficient power on the PCIe slot" is printed in dmsg, please refer to the Specifications section of your hardware user guide and make sure that you are providing your DPU the correct amount of power.

To verify how much power is supported on your host's PCIe slots, run the command `lspci -vvv | grep PowerLimit`. For example:

```
# lspci -vvv | grep PowerLimit
                        Slot #6, PowerLimit 75.000W; Interlock- NoCompl-
                        Slot #1, PowerLimit 75.000W; Interlock- NoCompl-
                        Slot #4, PowerLimit 75.000W; Interlock- NoCompl-
```

Be aware that this command is not supported by all host vendors/types.

8.5.2 HowTo update PCIe device description

lspci may not present the full description for the NVIDIA PCIe devices connected to your host. For example:

```
# lspci | grep -i Mellanox
a3:00.0 Infiniband controller: Mellanox Technologies Device a2d6 (rev 01)
a3:00.1 Infiniband controller: Mellanox Technologies Device a2d6 (rev 01)
a3:00.2 DMA controller: Mellanox Technologies Device c2d3 (rev 01)
```

Please run the following command:

```
# update-pciids
```

Now you should be able to see the full description for those devices. For example:

```
# lspci | grep -i Mellanox
a3:00.0 Infiniband controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network
controller (rev 01)
a3:00.1 Infiniband controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network
controller (rev 01)
a3:00.2 DMA controller: Mellanox Technologies MT42822 BlueField-2 SoC Management Interface (rev 01)
```

8.5.3 HowTo handle two BlueField DPU devices in the same server

Please refer to section "Multi-board Management Example".

8.6 SR-IOV Troubleshooting

8.6.1 Unable to create VFs

1. Please make sure that SR-IOV is enabled in BIOS.
2. Verify `SRIOV_EN` is true and `NUM_OF_VFS` bigger than 1. Run:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 -e q |grep -i "SRIOV_EN\|num_of_vf"
Configurations:                 Default         Current         Next Boot
*         NUM_OF_VFS            16              16              16
*         SRIOV_EN              True(1)         True(1)         True(1)
```

3. Verify that `GRUB_CMDLINE_LINUX="iommu=pt intel_iommu=on pci=assign-busses"`.

8.6.2 No traffic between VF to external host

1. Please verify creation of representors for VFs inside the Bluefield DPU. Run:

```
# /opt/mellanox/iproute2/sbin/rdma link |grep -i up
...
link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0
...
```

2. Make sure the representors of the VFs are added to the bridge. Run:

```
# ovs-vsctl add-port <bridage_name> pf0vf0
```

3. Verify VF configuration. Run:

```
$ ovs-vsctl show
bb993992-7930-4dd2-bc14-73514854b024
    Bridge ovsbr1
        Port pf0vf0
            Interface pf0vf0
                type: internal
        Port pf0hpf
            Interface pf0hpf
        Port pf0sf0
            Interface pf0sf0
        Port p0
            Interface p0
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port pf1sf0
            Interface pf1sf0
        Port p1
            Interface p1
        Port pf1hpf
            Interface pf1hpf
    ovs_version: "2.14.1"
```

8.7 eSwitch Troubleshooting

8.7.1 Unable to configure legacy mode

To set devlink to "Legacy" mode in BlueField, run:

```
# devlink dev eswitch set pci/0000:03:00.0 mode legacy
# devlink dev eswitch set pci/0000:03:00.1 mode legacy
```

Please verify that:

- No virtual functions are open. To verify if VFs are configured, run:

```
# /opt/mellanox/iproute2/sbin/rdma link | grep -i up
link mlx5_0/2 state ACTIVE physical_state LINK_UP netdev pf0vf0
link mlx5_1/2 state ACTIVE physical_state LINK_UP netdev pf1vf0
```

  If any VFs are configured, destroy them by running:

```
# echo 0 > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
# echo 0 > /sys/class/infiniband/mlx5_1/device/mlx5_num_vfs
```

- If any SFs are configured, delete them by running:

```
/sbin/mlnx-sf -a delete --sfindex <SF Index>
```

You may retrieve the `<SF Index>` of the currently installed SFs by running:

```
# mlnx-sf -a show

SF Index: pci/0000:03:00.0/229408
  Parent PCI dev: 0000:03:00.0
  Representor netdev: en3f0pf0sf0
  Function HWADDR: 02:61:f6:21:32:8c
  Auxiliary device: mlx5_core.sf.2
    netdev: enp3s0f0s0
```

```
      RDMA dev: mlx5_2

  SF Index: pci/0000:03:00.1/294944
    Parent PCI dev: 0000:03:00.1
    Representor netdev: en3f1pf1sf0
    Function HWADDR: 02:30:13:6a:2d:2c
    Auxiliary device: mlx5_core.sf.3
      netdev: enp3s0f1s0
      RDMA dev: mlx5_3
```

Pay attention to the SF Index values. For example:

```
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.0/229408
/sbin/mlnx-sf -a delete --sfindex pci/0000:03:00.1/294944
```

If the error "`Error: mlx5_core: Can't change mode when flows are configured`" is encountered while trying to configure legacy mode, please make sure that

1. Any configured SFs are deleted (see above for commands).
2. Shut down the links of all interfaces, delete any ip xfrm rules, delete any configured OVS flows, and stop openvswitch service. Run:

```
ip link set dev p0 down
ip link set dev p1 down
ip link set dev pf0hpf down
ip link set dev pf1hpf down
ip link set dev vxlan_sys_4789 down

ip x s f ;
ip x p f ;

tc filter del dev p0 ingress
tc filter del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1
tc qdisc del dev p0 ingress
tc qdisc del dev p1 ingress
tc qdisc show dev p0
tc qdisc show dev p1

systemctl stop openvswitch-switch
```

8.7.2 Arm appears as two interfaces

What this looks like:

```
# sudo /opt/mellanox/iproute2/sbin/rdma link
link mlx5_0/1 state ACTIVE physical_state LINK_UP netdev p0
link mlx5_1/1 state ACTIVE physical_state LINK_UP netdev p1
```

- Check if you are working in legacy mode.

```
# devlink dev eswitch show pci/0000:03:00.<0|1>
```

If the following line is printed, this means that you are working in legacy mode:

```
pci/0000:03:00.<0|1>: mode legacy inline-mode none encap enable
```

Please configure the DPU to work in switchdev mode. Run:

```
devlink dev eswitch set pci/0000:03:00.<0|1> mode switchdev
```

- Check if you are working in separated mode:

```
# mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -i cpu
* INTERNAL_CPU_MODEL SEPERATED_HOST(0)
```

Please configure the DPU to work in embedded mode. Run:

```
devlink dev eswitch set pci/0000:03:00.<0|1> mode switchdev
```

8.8 Isolated Mode Troubleshooting and How-Tos

8.8.1 Unable to burn FW from host server

Please verify that you are not in running in isolated mode. Run:

```
$ sudo mlxprivhost -d /dev/mst/mt41686_pciconf0 q
Current device configurations:
------------------------------
level                          : PRIVILEGED
...
```

By default, BlueField operates in privileged mode. Please refer to "Modes of Operation" for more information.

8.9 General Troubleshooting

8.9.1 Server unable to find the DPU

- Ensure that the DPU is placed correctly
- Make sure the DPU slot and the DPU are compatible
- Install the DPU in a different PCI Express slot
- Use the drivers that came with the DPU or download the latest
- Make sure your motherboard has the latest BIOS
- Perform a graceful shutdown then power cycle the server

8.9.2 DPU no longer works

- Reseat the DPU in its slot or a different slot, if necessary
- Try using another cable
- Reinstall the drivers for the network driver files may be damaged or deleted
- Perform a graceful shutdown then power cycle the server

8.9.3 DPU stopped working after installing another BFB

- Try removing and reinstalling all DPUs
- Check that cables are connected properly
- Make sure your motherboard has the latest BIOS

8.9.4 Link indicator light is off

- Try another port on the switch
- Make sure the cable is securely attached
- Check you are using the proper cables that do not exceed the recommended lengths
- Verify that your switch and DPU port are compatible

8.9.5 Link light is on but no communication is established

- Check that the latest driver is loaded
- Check that both the DPU and its link are set to the same speed and duplex settings

8.10 Installation Troubleshooting and How-Tos

8.10.1 BlueField target is stuck inside UEFI menu

Upgrade to the latest stable boot partition images, see "How to upgrade the boot partition (ATF & UEFI) without re-installation".

8.10.2 BFB does not recognize the BlueField board type

If the .bfb file cannot recognize the BlueField board type, it reverts to low core operation. The following message will be printed on your screen:

```
***System type can't be determined***
***Booting as a minimal system***
```

Please contact NVIDIA Support if this occurs.

8.10.3 Unable to load BL2, BL2R, or PSC image

The following errors appear in console if images are corrupted or not signed properly:

Device	Error
BlueField	`ERROR: Failed to load BL2 firmware`
BlueField-2	`ERROR: Failed to load BL2R firmware`
BlueField-3	`Failed to load PSC-BL1` or `PSC VERIFY_BCT timeout`

8.10.4 CentOS fails into "dracut" mode during installation

This is most likely configuration related.

- If installing through the RShim interface, check whether /var/pxe/centos7 is mounted or not. If not, either manually mount it or re-run the setup.sh script.

- Check the Linux boot message to see whether eMMC is found or not. If not, the BlueField driver patch is missing. For local installation via RShim, run the setup.sh script with the absolute path and check if there are any errors. For a corporate PXE server, make sure the BlueField and ConnectX driver disk are patched into the initrd image.

8.10.5 How to find the software versions of the running system

Run the following:

```
/opt/mellanox/scripts/bfvcheck:
root@bluefield:/usr/bin/bfvcheck# ./bfvcheck
Beginning version check...
-RECOMMENDED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
FW: 18.25.1010
-INSTALLED VERSIONS-
ATF: v1.5(release):BL2.0-1-gf9f7cdd
UEFI: 2.0-6004a6b
FW: 18.25.1010
Version checked
```

Also, the version information is printed to the console.

For ATF, a version string is printed as the system boots.

```
"NOTICE:  BL2: v1.3(release):v1.3-554-ga622cde"
```

For UEFI, a version string is printed as the system boots.

```
"UEFI firmware (version 0.99-18d57e3 built at 00:55:30 on Apr 13 2018)"
```

For Yocto, run:

```
$ cat /etc/bluefield_version
2.0.0.10817
```

8.10.6 How to upgrade the host RShim driver

See the readme at `<BF_INST_DIR>/src/drivers/rshim/README`.

8.10.7 How to upgrade the boot partition (ATF & UEFI) without re-installation

1. Boot the target through the RShim interface from a host machine:

   ```
   $ cat <BF_INST_DIR>/sample/install.bfb > /dev/rshim<N>/boot
   ```

2. Log into the BlueField target:

   ```
   $ /opt/mlnx/scripts/bfrec
   ```

8.10.8 How to upgrade ConnectX firmware from Arm side

The `mst`, `mlxburn`, and `flint` tools can be used to update firmware.

For Ubuntu, CentOS and Debian, run the following command from the Arm side:

```
sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl
```

8.10.9 How to configure ConnectX firmware

Configuring ConnectX firmware can be done using the mlxconfig tool.

It is possible to configure privileges of both the internal (Arm) and the external host (for DPUs) from a privileged host. According to the configured privilege, a host may or may not perform certain operations related to the NIC (e.g. determine if a certain host is allowed to read port counters).

For more information and examples please refer to the MFT User Manual which can be found at the following link.

8.10.10 How to use the UEFI boot menu

Press the "Esc" key when prompted after booting (before the countdown timer runs out) to enter the UEFI boot menu and use the arrows to select the menu option.

It could take 1-2 minutes to enter the Boot Manager depending on how many devices are installed or whether the EXPROM is programmed or not.

Once in the boot manager:

- "EFI Network xxx" entries with device path "PciRoot..." are ConnectX interface
- "EFI Network xxx" entries with device path "MAC(..." are for the RShim interface and the BlueField OOB Ethernet interface

Select the interface and press ENTER will start PXE boot.

The following are several useful commands under UEFI shell:

```
Shell> ls FS0:                                        # display file
Shell> ls FS0:\EFI                                    # display file
Shell> cls                                            # clear screen
Shell> ifconfig -l                                    # show interfaces
Shell> ifconfig -s eth0 dhcp                          # request DHCP
Shell> ifconfig -l eth0                               # show one interface
Shell> tftp 192.168.100.1 grub.cfg FS0:\grub.cfg      # tftp download a file
Shell> bcfg boot dump                                 # dump boot variables
Shell> bcfg boot add 0 FS0:\EFI\centos\shim.efi "CentOS" # create an entry
```

8.10.11 How to Use the Kernel Debugger (KGDB)

The default Yocto kernel has `CONFIG_KGDB` and `CONFIG_KGDB_SERIAL_CONSOLE` enabled. This allows the Linux kernel on BlueField to be debugged over the serial port. A single serial port cannot be used both as a console and by KGDB at the same time. It is recommended to use the RShim for console access (`/dev/rshim0/console`) and the UART port (`/dev/ttyAMA0 or /dev/ttyAMA1`) for KGDB. Kernel GDB over console (KGDBOC) does not work over the RShim console. If the RShim console is not available, there are open-source packages such as KGDB demux and agent-proxy which allow a single serial port to be shared.

There are two ways to configure KGDBOC. If the OS is already booted, then write the name of the serial device to the KGDBOC module parameter. For example:

```
$ echo ttyAMA1 > /sys/module/kgdboc/parameters/kgdboc
```

To attach GDB to the kernel, it must be stopped first. One way to do that is to send a "g" to `/proc/sysrq-trigger`.

```
$ echo g > /proc/sysrq-trigger
```

To debug incidents that occur at boot time, kernel boot parameters must be configured. Add "`kgdboc=ttyAMA1,115200 kgdwait`" to the boot arguments to use UART1 for debugging and force it to wait for GDB to attach before booting.

Once the KGDBOC module is configured and the kernel stopped, run the Arm64 GDB on the host machine connected to the serial port, then set the remote target to the serial device on the host side.

```
<BF_INST_DIR>/sdk/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb <BF_INST_DIR>/
sample/vmlinux

(gdb) target remote /dev/ttyUSB3
Remote debugging using /dev/ttyUSB3
arch_kgdb_breakpoint () at /labhome/dwoods/src/bf/linux/arch/arm64/include/asm/kgdb.h:32
32                asm ("brk %0" : : "I" (KGDB_COMPILED_DBG_BRK_IMM));
(gdb)
```

`<BF_INST_DIR>` is the directory where the BlueField software is installed. It is assumed that the SDK has been unpacked in the same directory.

8.10.12 How to enable/disable SMMU

SMMU could affect performance for certain applications. It is disabled by default and can be modified in different ways.

- Enable/disable SMMU in the UEFI System Configuration
- Set it in `bf.cfg` and push it together with the `install.bfb` (see section "Installing Popular Linux Distributions on BlueField")
- In BlueField Linux, create a file with one line with `SYS_ENABLE_SMMU=TRUE`, then run `bfcfg`.

The configuration change will take effect after reboot. The configuration value is stored in a persistent UEFI variable. It is not modified by OS installation.

See section "UEFI System Configuration" for information on how to access the UEFI System Configuration menu.

8.10.13 How to change the default console of the install image

On UART0:

```
$ echo "console=ttyAMA0 earlycon=pl011,0x01000000 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
        <BF_INST_DIR>/sample/ install.bfb
```

On UART1:

```
$ echo "console=ttyAMA1 earlycon=pl011,0x01000000 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
        <BF_INST_DIR>/sample/install.bfb
```

On RShim:

```
$ echo "console=hvc0 initrd=initramfs" > bootarg
$ <BF_INST_DIR>/bin/mlx-mkbfb --boot-args bootarg \
        <BF_INST_DIR>/sample/install.bfb
```

8.10.14 How to change the default network configuration during BFB installation

On Ubuntu OS, the default network configuration for `tmfifo_net0` and `oob_net0` interfaces is set by the cloud-init service upon first boot after BFB installation.

The default content of `/var/lib/cloud/seed/nocloud-net/network-config` as follows:

```
# cat /var/lib/cloud/seed/nocloud-net/network-config
version: 2
renderer: NetworkManager
ethernets:
  tmfifo_net0:
   dhcp4: false
   addresses:
    - 192.168.100.2/30
   nameservers:
    addresses: [ 192.168.100.1 ]
   routes:
   - to: 0.0.0.0/0
    via: 192.168.100.1
    metric: 1025
  oob_net0:
   dhcp4: true
```

This content can be modified during BFB installation using `bf.cfg`. For example:

```
# cat bf.cfg
bfb_modify_os()
{
     sed -i -e '/oob_net0/,+1d' /mnt/var/lib/cloud/seed/nocloud-net/network-config
cat >> /mnt/var/lib/cloud/seed/nocloud-net/network-config << EOF
  oob_net0:
   dhcp4: false
   addresses:
    - 10.0.0.1/24
EOF
}

# bfb-install -c bf.cfg -r rshim0 -b <BFB>
```

Using the same technique, any configuration file on the BlueField DPU side can be updated during the BFB installation process.

8.10.15 Sanitizing DPU eMMC and SSD Storage

During the BFB installation process, DPU storage can be securely sanitized either using the `shred` or the `mmc` and `nvme` utilities in the `bf.cfg` configuration file as illustrated in the following subsections.

By default, only the installation target storage is formatted using the Linux `mkfs` utility.

8.10.15.1 Using shred Utility

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme

        if [ -e /dev/mmcblk0 ]; then
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
                echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
                nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
                nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
        fi
        SANITIZE_DONE=1
        echo ===================== sanitize.log ===================== | tee /dev/kmsg
        cat /tmp/sanitize.*.log | tee /dev/kmsg
        sync
fi
bfb_modify_os()
{
        echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
        if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
                cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
        fi
}
```

8.10.15.2 Using mmc and nvme Utilities

```
# cat bf.cfg
SANITIZE_DONE=${SANITIZE_DONE:-0}
export SANITIZE_DONE
if [ $SANITIZE_DONE -eq 0 ]; then
        sleep 3m
        /sbin/modprobe nvme

        if [ -e /dev/mmcblk0 ]; then
                echo Sanitizing /dev/mmcblk0 | tee /dev/kmsg
                echo Sanitizing /dev/mmcblk0 > /tmp/sanitize.emmc.log
                mmc sanitize /dev/mmcblk0 >>  /tmp/sanitize.emmc.log 2>&1
        fi
        if [ -e /dev/nvme0n1 ]; then
                echo Sanitizing /dev/nvme0n1 | tee /dev/kmsg
                echo Sanitizing /dev/nvme0n1 > /tmp/sanitize.ssd.log
                nvme sanitize /dev/nvme0n1 -a 2 >> /tmp/sanitize.ssd.log 2>&1
                nvme sanitize-log /dev/nvme0n1 >> /tmp/sanitize.ssd.log 2>&1
        fi
        SANITIZE_DONE=1
        echo ===================== sanitize.log ===================== | tee /dev/kmsg
        cat /tmp/sanitize.*.log | tee /dev/kmsg
        sync
fi
bfb_modify_os()
{
        echo ===================== bfb_modify_os ===================== | tee /dev/kmsg
        if ( /bin/ls -1 /tmp/sanitize.*.log > /dev/null 2>&1 ); then
                cat /tmp/sanitize.*.log > /mnt/root/sanitize.log
        fi
}
```

9 Windows Support

9.1 Network Drivers

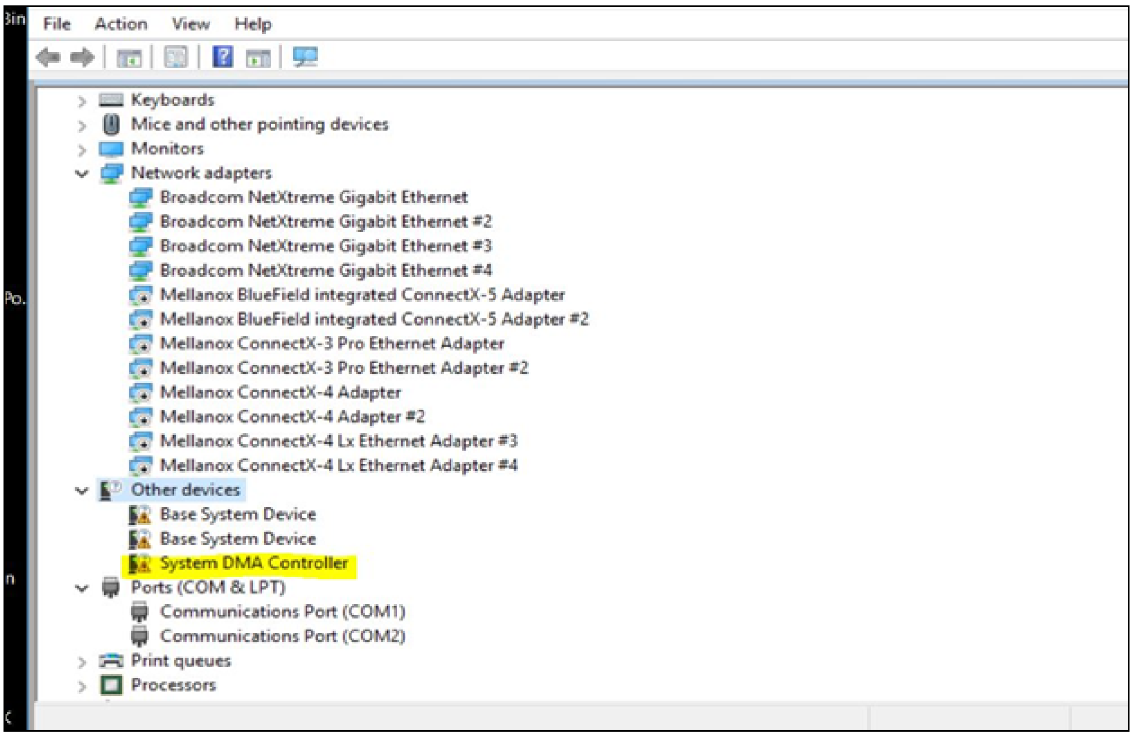
BlueField Windows support from the host-side is facilitated by the WinOF-2 driver. For more information on WinOF-2 (including installation), please refer to the WinOF-2 Documentation.

9.2 RShim Drivers

RShim drivers provide functionalities like resetting the Arm cores, pushing a bootstream image, as well as some networking and console functionalities.

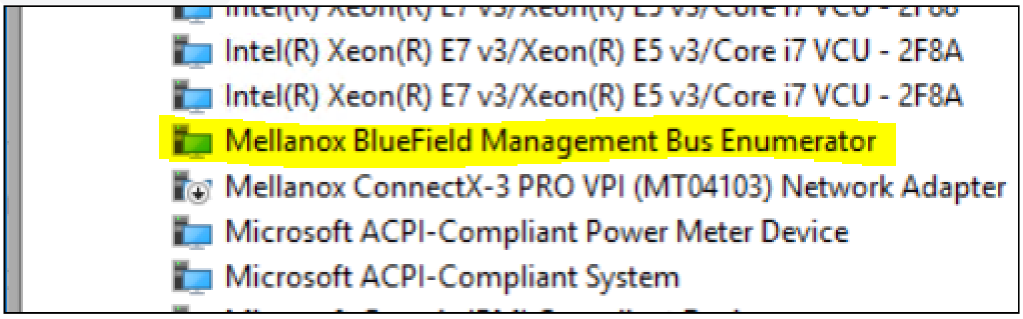
9.3 Verifying RShim Drivers Installation

1. Open the Device Manager when no drivers are installed to make sure a new PCIe device is available as below.

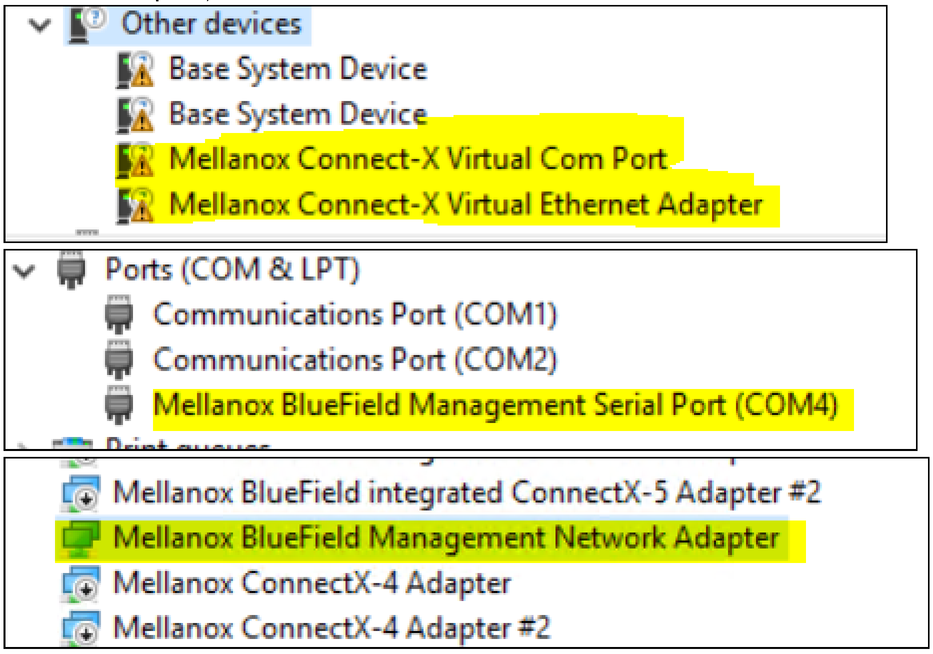


Keyboards
Mice and other pointing devices
Monitors
Network adapters
Broadcom NetXtreme Gigabit Ethernet
Broadcom NetXtreme Gigabit Ethernet #2
Broadcom NetXtreme Gigabit Ethernet #3
Broadcom NetXtreme Gigabit Ethernet #4
Mellanox BlueField integrated ConnectX-5 Adapter
Mellanox BlueField integrated ConnectX-5 Adapter #2
Mellanox ConnectX-3 Pro Ethernet Adapter
Mellanox ConnectX-3 Pro Ethernet Adapter #2
Mellanox ConnectX-4 Adapter
Mellanox ConnectX-4 Adapter #2
Mellanox ConnectX-4 Lx Ethernet Adapter #3
Mellanox ConnectX-4 Lx Ethernet Adapter #4
Other devices
Base System Device
Base System Device
System DMA Controller
Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Print queues
Processors

2. Run the installer to install all 3 drivers (`MlxRshimBus.sys` , `MlxRshimCom.sys` , and `MlxRshimEth.sys`).


Intel(R) Xeon(R) E7 v3/Xeon(R) E5 v3/Core i7 VCU - 2F8A
Intel(R) Xeon(R) E7 v3/Xeon(R) E5 v3/Core i7 VCU - 2F8A
Mellanox BlueField Management Bus Enumerator
Mellanox ConnectX-3 PRO VPI (MT04103) Network Adapter
Microsoft ACPI-Compliant Power Meter Device
Microsoft ACPI-Compliant System

3. Make sure the Bus driver created 2 child devices after the installation (Com port and the Ethernet adapter).


Other devices
Base System Device
Base System Device
Mellanox Connect-X Virtual Com Port
Mellanox Connect-X Virtual Ethernet Adapter

Ports (COM & LPT)
Communications Port (COM1)
Communications Port (COM2)
Mellanox BlueField Management Serial Port (COM4)

Mellanox BlueField integrated ConnectX-5 Adapter #2
Mellanox BlueField Management Network Adapter
Mellanox ConnectX-4 Adapter
Mellanox ConnectX-4 Adapter #2

At this time, PuTTY application or any other network utility can be used to communicate with DPU via Virtual Com Port or Virtual Ethernet Adapter (ssh). The Com Port can be used using the 9600 baud-rate and default settings.

RShim drivers can be connect via PCIe (the drivers we are providing) or via USB (external connection) but not both at the same time. So when the bus driver detects that an external USB is already attached, it will not create the child virtual devices for data access. Access via PCIe is available once the USB connection is removed.

9.4 Accessing BlueField DPU From Host

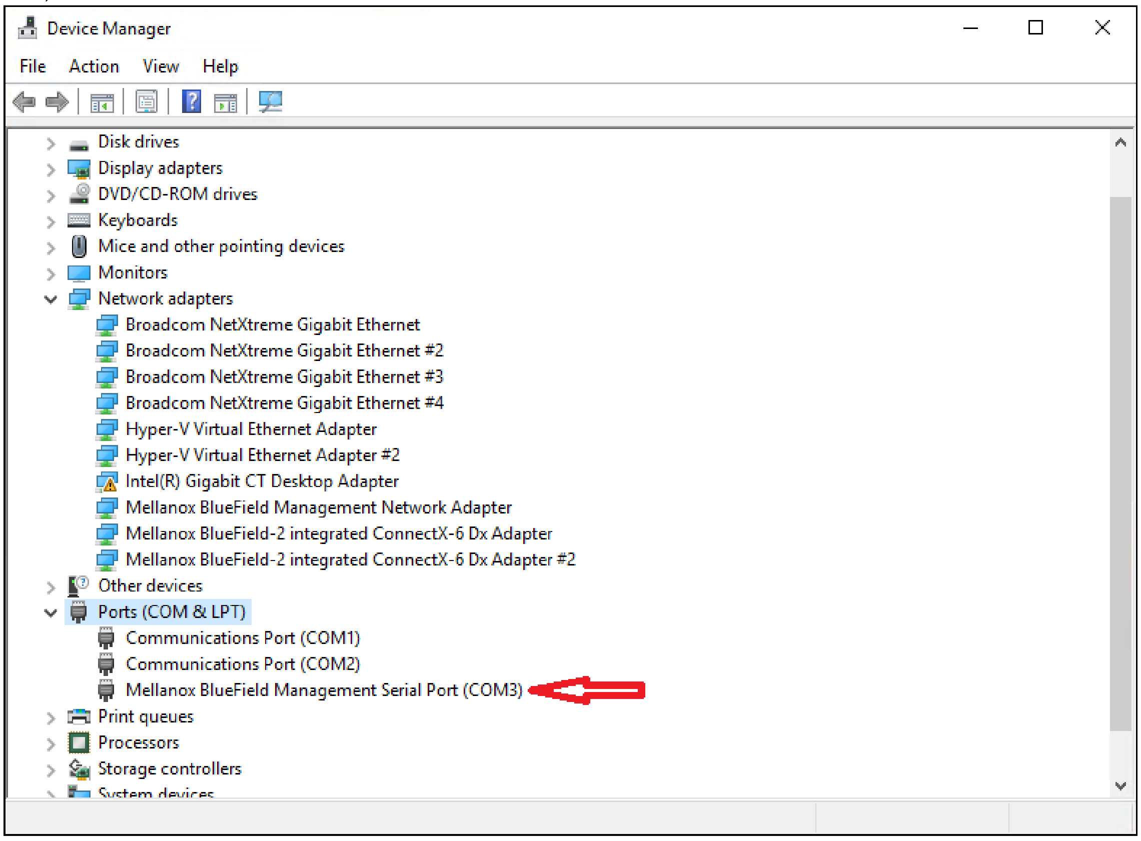
The BlueField DPU can be accessed via PuTTY or any other network utility application to communicate via virtual COM or virtual Ethernet adapter. To use COM:

1. Open Putty.
2. Change connection type to Serial.

3. Run the following command in order to know what to set the "Serial line" field to:

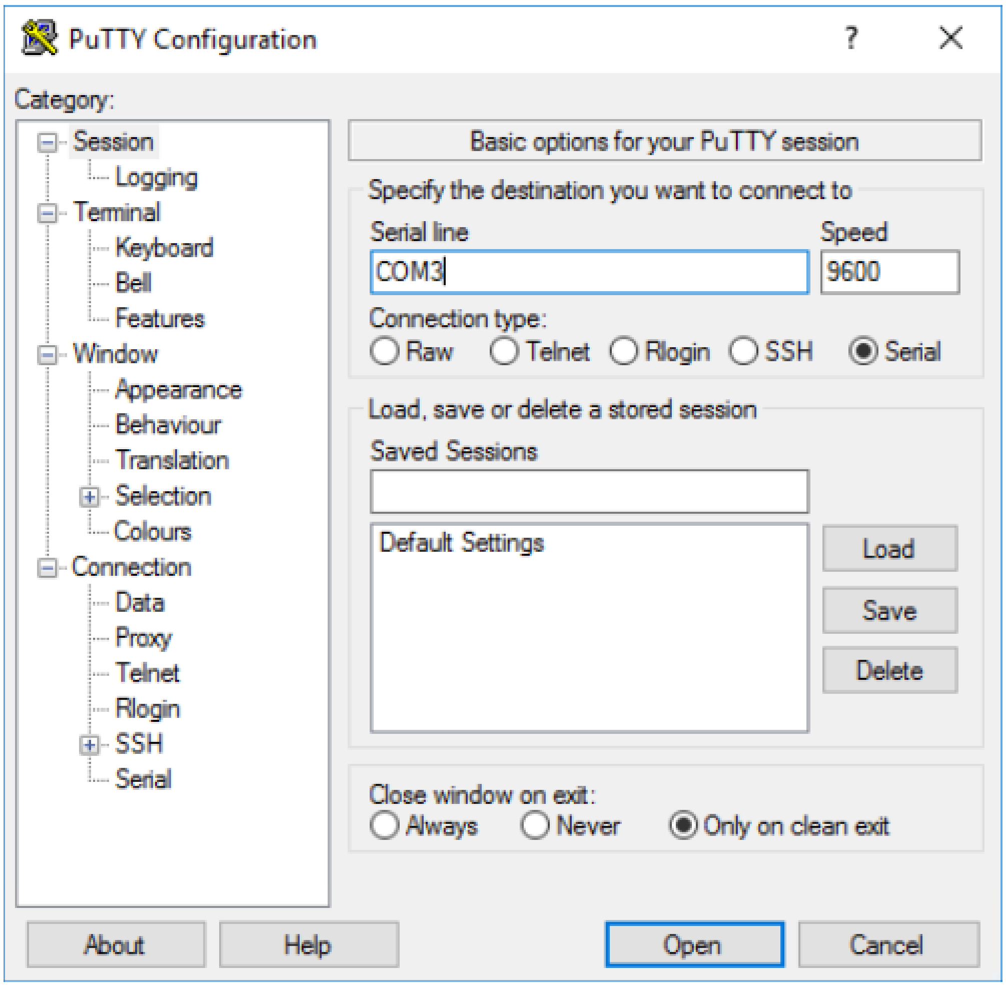
```
C:\Users\username\Desktop> reg query HKLM\HARDWARE\DEVICEMAP\SERIALCOMM | findstr MlxRshim
    \MlxRshim\COM3       REG-SZ      COM3
```

In this case use COM3. This name can also be found via Device Manager under "Ports (Com & LPT)".



```
> Disk drives
> Display adapters
> DVD/CD-ROM drives
> Keyboards
> Mice and other pointing devices
> Monitors
v Network adapters
    Broadcom NetXtreme Gigabit Ethernet
    Broadcom NetXtreme Gigabit Ethernet #2
    Broadcom NetXtreme Gigabit Ethernet #3
    Broadcom NetXtreme Gigabit Ethernet #4
    Hyper-V Virtual Ethernet Adapter
    Hyper-V Virtual Ethernet Adapter #2
    Intel(R) Gigabit CT Desktop Adapter
    Mellanox BlueField Management Network Adapter
    Mellanox BlueField-2 integrated ConnectX-6 Dx Adapter
    Mellanox BlueField-2 integrated ConnectX-6 Dx Adapter #2
> Other devices
v Ports (COM & LPT)
    Communications Port (COM1)
    Communications Port (COM2)
    Mellanox BlueField Management Serial Port (COM3)
> Print queues
> Processors
> Storage controllers
> System devices
```

4. Press Open and hit Enter.



Specify the destination you want to connect to

Serial line: COM3

Speed: 9600

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

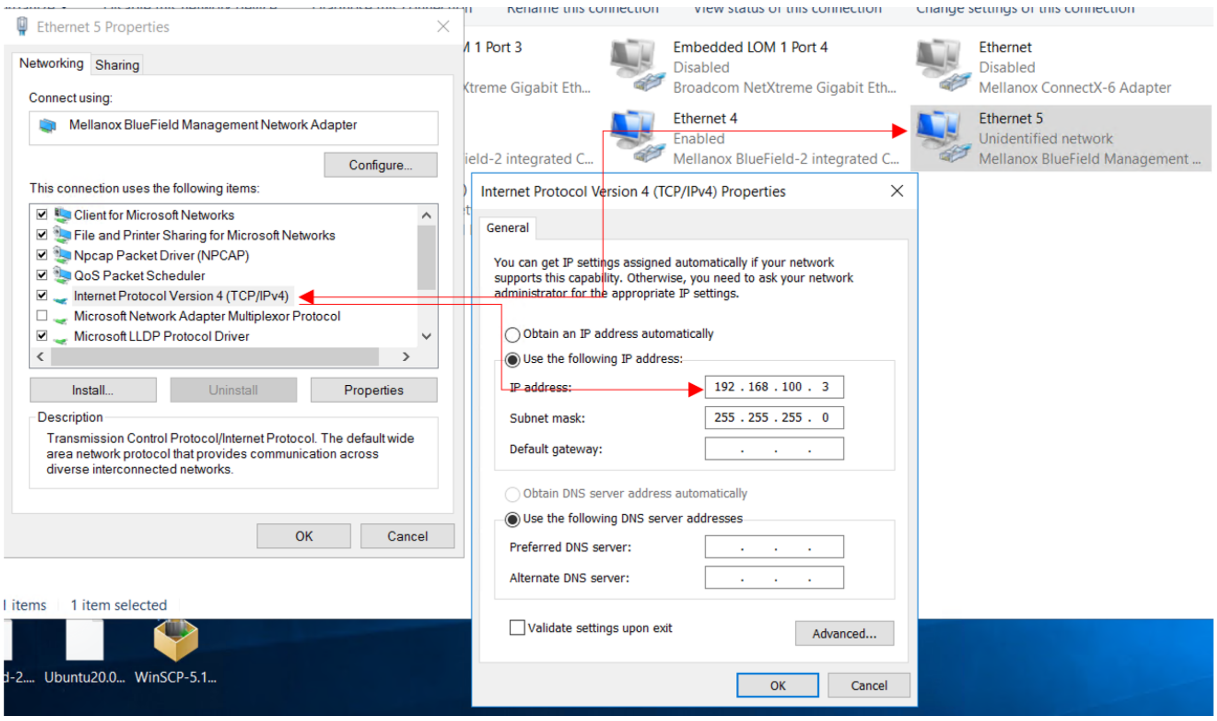
Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

To access via BlueField management network adapter, configure an IP address as shown in the example below and run a ping test to confirm configuration.


Ethernet 5 Properties

Networking | Sharing

Connect using:

Mellanox BlueField Management Network Adapter

This connection uses the following items:

- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] Npcap Packet Driver (NPCAP)
- [✓] QoS Packet Scheduler
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol
- [✓] Microsoft LLDP Protocol Driver

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

Embedded LOM 1 Port 4 — Disabled — Broadcom NetXtreme Gigabit Eth...

Ethernet — Disabled — Mellanox ConnectX-6 Adapter

Ethernet 4 — Enabled — Mellanox BlueField-2 integrated C...

Ethernet 5 — Unidentified network — Mellanox BlueField Management ...

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address:	192 . 168 . 100 . 3
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server:	. . .
Alternate DNS server:	. . .

[ ] Validate settings upon exit

9.5 RShim Ethernet Driver

The device does not support any type of stateful or stateless offloads. This is indicated to the Operating System accordingly when the driver loads. The MAC address is a pre-defined MAC address (CA-FE-01-CA-FE-02). The following registry keys can be used to change basic settings such as MAC address.

Registry Name	Description	Valid Values
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>*JumboPacket	The size, in bytes, of the largest supported Jumbo Packet (an Ethernet frame that is greater than 1514 bytes) that the hardware can support.	1514 (default) - 2048
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>*NetworkAddress	The network address of the device. The format for a MAC address is: XX-XX-XX-XX-XX-XX.	CA-FE-01-CA-FE-02 (default)
HKLM\SYSTEM\CurrentControlSet\Control\Class\{4d36e972-e325-11ce-bfc1-08002be10318}\<nn>\ReceiveBuffers	The number of receive descriptors used by the miniport adapter.	16 – 64 (Default)

Update the MAC address manually using registry key if there are more than one BlueField DPU in the system.

For instructions on how to find interface index in the registry (nn), please refer to section "Finding the Index Value of the Network Interface" in the WinOF-2 User Manual under Features Overview and Configuration > Configuring the Driver Registry Keys.

9.6 MlxRshimBus Driver

This driver does all the read/write work to the hardware registers. User space application can send down IOCTL's to restart the system on chip or to push a new BlueField boot stream image.

9.7 RshimCmd Tool

RshimCmd is a command line tool that enables the user to:

- Restart the DPU.
- Push a boot stream file (`.bfb`). A BFB file is a generated BlueField boot stream file that contains Linux operating system image that runs on the DPU. BFB files can be downloaded from the NVIDIA DOCA SDK webpage.

Usage	`RshimCmd -RestartSmartNic <Option> -BusNum <BusNum>`

Example	`RshimCmd -EnumDevices`
	`RshimCmd -PushImage c:\bin\MlnxBootImage.bfb -BusNum 11`
	`RshimCmd -RestartSmartNic 1 -BusNum 11`

Detailed Usage	`RshimCmd -h`

The BFB image can be either CentOS or Ubuntu. Ubuntu credentials are: `ubuntu` / `ubuntu` and for Centos credentials are: root/centos, IP address of RShim Ethernet component (called tmfifo_net0) on the BlueField side is `192.168.100.2/30` by default. Please set IP address on the Windows side accordingly to be able to communicate via SSH.

9.8 BlueField UEFI System Boot Customizations during Installation

Bluefield's UEFI system boot options and more can be customized during the BFB Installation through the use of configuration parameters in the `bf.cfg` file. For further information on the `bf.cfg` file, refer to the BlueField Documentation.

To include the `bf.cfg` file into the BFB installation, append the file to BFB file as described below:

1. Copy the BFB file to a local folder. For example:

```
Copy <path>\DOCA_1.4.0_BSP_3.9.2_Ubuntu_20.04-5.20220707.bfb c:\bf\MlnxBootImage.bfb
```

2. Append the bf.cfg file into the BFB file.

```
Cd c:\bf
```

```
Copy /b MlnxBootImage.bfb + bf.cfg MlnxBootImage_with_bf_cfg.bfb
```

3. Download the BFB image.

```
RshimCmd -PushImage c:\bf\MlnxBootImage_with_bf_cfg.bfb -BusNum 11
```

As the `bf.cfg` is intended for Linux OSes, it should be created according to Linux rules. For example, the lines of this text file should end in LF and not in CR/LF as accepted in Windows.

All the syntax should be as the accepted by the OS expects. For example, there should be no spaces in the middle of "set" statements: `NET_RSHIM_MAC=00:1a:ca:ff:ff:05`.

9.9 EventLogs and Driver Logging

All driver logging is part of the Mellanox-WinOF2-Kernel trace session that comes with the network drivers installation. The default location to the trace is at `%SystemRoot%\system32\LogFiles\Mlnx\Mellanox-WinOF2-System.etl`.

The following are the Event logs RShim drivers generate:

9.9.1 MlxRShimBus Driver

Event ID	Severity	Message
2	Informational	RShim Bus driver loaded successfully
3	Informational	Device successfully stopped
4	Error	The SmartNIC seems to be stuck as the boot FIFO data is not being drained.
5	Error	Driver startup failed due to failure in creation of the child device.
6	Error	SmartNIC is in a bad state. Please restart SmartNIC and reload bus drivers. Please refer to user manual on how to restart SmartNIC.
7	Warning	SmartNIC is in LiveFish mode
8	Warning	Failed creating child virtual devices as a backend USB device is attached and accessing RShim FIFO. Please refer to user manual for more details.

9.9.2 MlxRShim Serial Driver

Event ID	Severity	Message
2	Informational	RShim serial driver loaded successfully
3	Informational	device successfully stopped

9.9.3 MlxRShim Ethernet Driver

Event ID	Severity	Message
2	Error	MAC address read from registry is not supported. Please set valid unicast address.
3	Informational	Device is successfully stopped
4	Warning	Value read from registry is invalid. Therefore use the default value.
5	Error	SmartNIC seems stuck as transmit packets are not being drained.
6	Informational	RShim Ethernet driver loaded successfully

10 Document Revision History

10.1 Rev 4.7.0 – May 06, 2024

Added:

- Section "UEFI Menu"
- Section "Redfish"
- Section "BlueField SR-IOV"
- Section "NVIDIA BlueField Reset and Reboot Procedures" and updated graceful shutdown guidance with pointers to this section

Updated:

- Section "Software Installation and Upgrade" with `bf-fwbundle-<version>.prod.bfb` information
- Section "BFB Installation"
- Section "Changing Default Credentials Using bf.cfg"
- Section "Configuring NIC Mode on BlueField-3 Using Redfish"

10.2 Rev 4.6.0 – February 08, 2024

Added:

- Page "Default Passwords and Policies"
- Section "VF Msix_num/Queue Requirement"

Updated:

- Section "Customization of BFB Installation Using bf.cfg"
- Section "bf.cfg Parameters"
- Section "Configuring NIC Mode on BlueField-3 Using Redfish"
- Section "Configuring NIC Mode on BlueField-3 from UEFI"
- Section "NIC Mode for BlueField-2"
- Section "Default Ports and OVS Configuration"
- Section "SystemD Service"

10.3 Rev 4.5.0 – December 12, 2023

Added:

- Section "Updating Software Using Redfish"
- Section "Sanitizing DPU eMMC and SSD Storage"
- Section "How to perform graceful shutdown"
- Section "BFB installation monitoring"

Updated:

- Page "Updating DPU Software Packages Using Standard Linux Tools"
- Section "RShim Logging"

- Section "NIC Mode"
- Section "Enabling OVS-DPDK Hardware Offload"
- Section "Enabling IPsec Packet Offload"
- Section "Setting IPSec Packet Offload Using strongSwan"
- Section "Running strongSwan Example"
- Section "Building strongSwan"
- Section "IPsec Packet Offload and OVS Offload"

10.4 Rev 4.2.2 – October 24, 2023

Updated:

- Section "NIC Mode"

10.5 Rev 4.2.0 – August 10, 2023

Updated:

- Step 3 under section "PXE Server Preparations"
- Section "Removing Previously Installed DOCA Runtime Packages"
- Section "NIC Mode"
- Sections "Connection Tracking With NAT" and "Querying Connection Tracking Offload Status" with conntack command for Ubuntu 22.04 kernels
- Section "LAG Configuration"
- Section "SystemD Service"
- Page "QoS Configuration"
- Section "bf.cfg Parameters"

10.6 Rev 4.0.2 – May 08, 2023

Added:

- Page "SoC Management Interface"
- Page "Legal Notices and 3rd Party Licenses"
- Section "Unable to load BL2, BL2R, or PSC image"

Updated:

- Section "Default Ports and OVS Configuration" with new step 2
- Section "BlueField Linux Drivers" with `gpio-mlxbf3`, `mlxbf-ptm`, `pwr-mlxbf`, and `pinctrl-mlxbf`
- Page "Updating DPU Software Packages Using Standard Linux Tools"
- Page "UEFI Secure Boot"
- Section "IPsec Hardware Offload: Full Offload" with Canonical note
- Section "How to upgrade ConnectX firmware from Arm side"
- Section "VirtIO-net PF Device Configuration" by removing `ECPF_ESWITCH_MANAGER` and `ECPF_PAGE_SUPPLIER` from step 4

- Section "Virtio-net SR-IOV VF Device Configuration" by removing `ECPF_ESWITCH_MANAGER` and `ECPF_PAGE_SUPPLIER` from step 7.b
- Section "vDPA over VirtIO Full Emulation"

10.7 Rev 3.9.3 – November 02, 2022

Added:

- Section "DHCP Client Configuration"
- Section "Updating DPU Software Packages Using Standard Linux Tools"
- Section "Creating Transitional Hotplug VirtIO-net PF Device"
- Section "Transitional VirtIO-net VF Device Support"

Updated:

- Section "Upgrading Boot Software" by specifying that the "Reset EFI Variables" action also wipes the BOOT option variables and secure boot keys
- Section "BlueField Linux Drivers"
- Section "Configuring Uplink MTU"
- Section "Disabling Host Networking PFs" by adding instructions for reactivating host networking for single-port DPUs
- Section "Configuring RegEx Acceleration on BlueField-2"
- Section "Virtio-net SR-IOV VF Device Configuration"
- `PXE_DHCP_CLASS_ID` in section "bf.cfg Parameters"

Removed:

- Step 7 in section "Configuring Host Server Side"
- Separated Mode from "Modes of Operation"

10.8 Rev 3.9.2 – August 02, 2022

Added:

- Section "Updating NVConfig Params"
- Page "System Configuration and Services"
- Section "Enrolling New NVIDIA Certificates"
- Section "bf.cfg Parameters"
- Support for OpenSSL version 3.0.2 in section "PKA Use Cases"
- Section "How to change the default network configuration during BFB installation"

Updated:

- Section "Firmware Upgrade"
- Section "Customizations During BFB Installation"
- Section "UEFI System Configuration"
- Page "Host-side Interface Configuration"
- Section "Enrolling Certificates Using Capsule"
- Section "NIC Mode" with supported MLNX_OFED versions
- Section "PKA Use Cases" with support for OpenSSL version 3.0.2

10.9 Rev 3.9 – May 03, 2022

Added:

- Section "GRUB Password Protection"
- New note under step 2 in section "Default Ports and OVS Configuration"
- Section "BlueField Linux Drivers"
- Canonical db certificate to section "Existing DPU Certificates"
- New note under section "Enrolling Certificates Using Capsule"
- New power cycle note under section "Enabling Host Restriction"
- New power cycle note under section "Disabling Host Restriction"
- Section "NIC Mode"
- Section "LAG on Multi-host"
- New power cycle note under section "Disabling Host Networking PFs"
- Section "PKA Prerequisites"
- Section "OVS IPsec"
- Section "Rate Limiting VF Group"
- Note to section "User Frontend"
- Section "Controller Live Update"

Updated:

- Code block in section "Customizations During BFB Installation"
- Section "Building Your Own BFB Installation Image"
- Section "Configuring VXLAN Tunnel"
- Step 2 in section "Prerequisites"
- Section "Enabling IPsec Full Offload"
- Code block under step 1 in section "LAG Configuration"

10.10 Rev 3.8.5 – January 19, 2022

Added:

- Section "Another backend already attached"

Updated:

- Section "Ensure RShim Running on Host"

11 Legal Notices and 3rd Party Licenses

BlueField Software Components	Version	3rd Party Components and Licenses
DOCA SDK	2.7.0	Link
DOCA SDK 3rd Party Notice		Link
DOCA SDK 3rd Party Unify Notice		Link
SoC OS Linux Ubuntu 22.04 Distro	5.15.0-1042-bluefield	Link
SoC OS Linux Ubuntu 20.04 Distro	5.4.0-1084-bluefield	Link
BSP - ATF	4.7.0	Link
BSP - ATF 3rd Party Notice		Link
BSP - ATF 3rd Party Unify Notice		Link
BSP - UEFI (EDK2)	4.7.0	Link
BlueField UEFI (EDK2) 3rd Party Notice		Link
BlueField UEFI (EDK2) 3rd Party Unify Notice		Link
BlueField BMC	24.04	Link
BlueField BMC 3rd Party Notice		Link
BlueField BMC 3rd Party Unify Notice		Link
Virtio Network Controller	1.9.12	Link
Virtio Network Controller 3rd Party Notice		Link
Virtio Network Controller 3rd Party Unify Notice		Link
MLNX LibSnap and virtio-blk	1.6.0-1	Link
MLNX LibSnap and virtio-blk 3rd Party Notice		Link
MLNX SNAP and SPDK	3.8.0-1	Link
MLNX SNAP and SPDK 3rd Party Notice		Link
NVIDIA MLNX_OFED License	24.04-0	Link
NVIDIA MLNX_OFED 3rd Party Unify Notice		Link
NVIDIA MFT License	4.28.0	Link
NVIDIA MFT 3rd Party Notice		Link
NVIDIA MLNX_DPDK	22.11.2404.1.0	Link
NVIDIA MLNX_DPDK 3rd Party Notice		Link
NVIDIA MLNX_DPDK 3rd Party Unify Notice		Link