# Software Installation and Upgrade

# Table of contents

**Info**

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

  **Info**

  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 BF-Bundle and BF-FW-Bundle BFB files. The BFB installation updates BMC software components automatically if BMC credentials (i.e., BMC_USER and BMC_PASSWORD) are provided in bf.cfg.

**Info**

The minimum BMC Firmware version that supports this method of BMC upgrade from BFB image, is 23.07. If your BMC firmware version is lower, follow the NVIDIA BlueField BMC Software documentation to upgrade BMC firmware. The BMC version can be obtained by following instructions here.

**Info**

BMC_REBOOT="no" by default. This will require BMC to be rebooted after BFB installation process finish to apply the new BMC firmware version. BMC reboot will reset the BMC console.

**Info**

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).

# Deploying BlueField Software Using BFB from Host

**Info**

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.

**Note**

This procedure assumes that a NVIDIA® BlueField® networking platform (DPU or SuperNIC) has already been installed in a server according to the instructions detailed in the BlueField device's hardware user guide.

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

| 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 |

| Step | Procedure | Link to Section |
|---|---|---|
| 4 | Install the Ubuntu BFB image | BFB Installation |
| 5 | Verify installation completed successfully | Verify BFB is Installed |
| 6 | Upgrade the firmware on your BlueField | Firmware Upgrade |

## 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`<br>`host# sudo apt-get autoremove` |
| CentOS/RHEL | `host# for f in $(rpm -qa \|grep -i doca ) ; do yum -y remove $f; done`<br>`host# yum autoremove`<br>`host# yum makecache` |

## 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*.<br>2. Unpack the deb repo. Run:<br>`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`<br>3. Perform apt update. Run:<br>`host# sudo apt-get update`<br>4. Run `apt install` for DOCA runtime package.<br>`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*.<br>2. Unpack the RPM repo. Run:<br>`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`<br>3. Enable new yum repos. Run:<br>`host# sudo yum makecache`<br>4. Run `yum install` to install DOCA runtime package. |

| OS | Procedure |
|---|---|
|  | `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*.<br>2. Unpack the RPM repo. Run:<br>`host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`<br>3. Enable new dnf repos. Run:<br>`host# sudo dnf makecache`<br>4. Run `dnf install` to install DOCA runtime.<br>`host# sudo dnf install doca-runtime` |

## 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 BlueField 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.

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

   ```
   sudo systemctl restart rshim
   ```

2. 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.

## Installing Ubuntu on BlueField

## BFB Installation

**Note**

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

**Info**

To upgrade the BMC firmware using BFB, the user must provide the current BMC credentials in the bf.cfg. Refer to "Customizing BlueField Software Deployment Using bf.cfg" for more information.

**Note**

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-level reset.

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.

**Note**

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.

**Warning**

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.

⚠ **Warning**

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

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
```

## 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.

**Note**

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
```

## 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 BlueField. 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>
   ```

   **Note**

**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.

## 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.
```

**Note**

For now, please ignore tool's instruction to reboot

**Note**

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

```
mst start
mst status
```

Example output taken on a multiple BlueField 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 devices 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 is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your BlueField type.

If you have an InfiniBand-and-Ethernet-capable BlueField, 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.

**Note**

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.

## 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
```

**Note**

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
```

**Note**

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.

## 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 devices 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 BlueField devices 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>
```

**Note**

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

```
systemctl restart rshim
```

## 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
```

## DHCP Client Configuration

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

## Ubuntu Dual Boot Support

BlueField 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

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

## Installing Ubuntu OS Image Using Dual Boot

> **Note**
>
> 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 the 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.

**Note**

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

## Upgrading Ubuntu OS Image Using Dual Boot

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

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

2. Reset BlueField to load the new OS image:

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

   BlueField should 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
```

**Note**

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` .

# Deploying BlueField Software Using BFB from BMC

**Info**

It is recommended to upgrade your NVIDIA® BlueField® networking platform (DPU or SuperNIC) to the latest software and firmware versions available to benefit from new features and latest bug fixes.

**Note**

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

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

| 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 |

| Step | Procedure | Direct Link |
|---|---|---|
| 3 | Install the Ubuntu BFB image | BFB Installation |
| 4 | Verify installation completed successfully | Verify BFB is Installed |
| 5 | Upgrade the firmware on your BlueField | Firmware Upgrade |

**Note**

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
```

00000191-70c2-dc13-a19d-f0e2efaf0003

## 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 BlueField with integrated BMC".

## BFB Installation

To update the software on the NVIDIA® BlueField® device, the BlueField 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 devices 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.

To update the software on the NVIDIA® BlueField® device, the BlueField 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 devices 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.

## Changing Default Credentials Using bf.cfg

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).

**Note**

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.

## Installing BFB

The BFB installation procedure consists of the following main stages:

1. Disabling RShim on the server.
2. Initiating the BFB update procedure by transferring the BFB image using one of the following options:
   - Redfish interface – `SimpleUpdate` with SCP, HTTP, or HTTPS
     1. Confirming the identity of the host and BMC—required only for SCP, during first-time setup or after BMC factory reset.
     2. Sending a `SimpleUpdate` request.
   - Direct SCP
3. Tracking the installation's progress and status.

**Note**

While the BlueField Bundle (BFB) contains NIC firmware images, it does not automatically install them. To automatically install the NIC firmware during BFB upgrade, generate the configuration file `bf.cfg` and combine it with the BFB file:

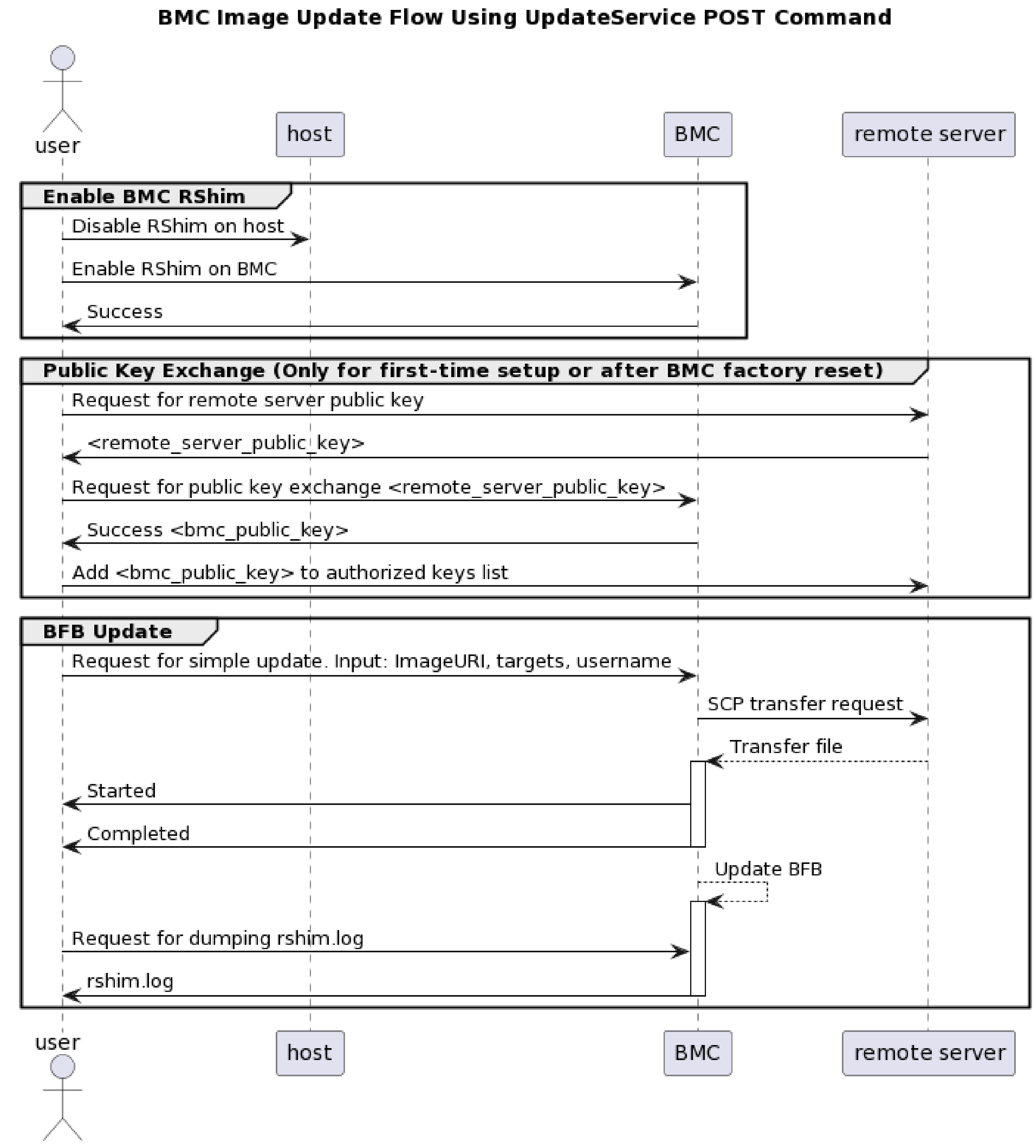
```
# echo WITH_NIC_FW_UPDATE=yes > bf.cfg
# cat <path_to_bfb> bf.cfg > new.bfb
```

## Transferring BFB File

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.

### Redfish Interface

### Installing BFB File Using SCP Protocol


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

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

1. Prepare secure file transfer of BFB:
    1. Gather the public SSH host keys of the server holding the new.bfb file. Run the following against the server holding the new.bfb file ("Remote Server"):

**Info**

OpenSSH is required for this step.

```
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

2. Retrieve the remote server's public key from the response, and send the following Redfish command to the BlueField BMC:

```
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.PublicKeyExc
```

Where:

- password – BlueField BMC password
- 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 **one space** between the two fields (i.e., "<type> <public_key>")
- bmc_ip – BMC IP address

3. 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 remote server holding the BFB file. This enables password-less 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"
        }
    ]
  }
```

2. Initiate image transfer. Run the following Redfish command:

```
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
```

**Note**

This command uses SCP for the image transfer, initiates a soft reset on the BlueField, and then pushes the boot stream. For NVIDIA-supplied BFBs, the eMMC is flashed automatically once the boot stream is pushed. Upon success, a `running` message is received.

**Info**

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

Where:

- `image_uri` – contains both the remote server IP address and the full path to the `.bfb` file on the remote server, with **one slash** between the two fields (i.e., `<remote_server_ip>/<full_path_of_bfb>` ).

  **Info**

  For example, if `<remote_server_ip>` is `10.10.10.10` and `<full_path_of_bfb>` is `/tmp/file.bfb` then `"ImageURI":"10.10.10.10//tmp/file.bfb"`.

- `username` – username on the remote server
- `bmc_ip` – BMC IP address

  Response/error messages:

  - 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."
  }
 ]
}
```

- Success message 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. Run the following Redfish command to track the SCP image's transfer status (percentage is not updated until it reaches 100%):

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

**Note**

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:

- 
    - `bmc_ip` – BMC IP address

- task_id – task ID received by the UpdateService command response

  Examples:

- 
  - 
    - Response/error messages:
      - 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"
```

**Info**

In this case, revoke the remote server key using the following Redfish command:

```
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/NvidiaUpdat
```

Where:

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

Then repeat steps 1 and 2.

- 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"
```

**Info**

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"
```

- Success/status messages:
  - 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 successful completion of SCP BFB transfer:

```
{
      "@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"
```

### Installing BFB File with HTTP Protocol

1. Make sure the BFB file, `new.bfb`, is available on HTTP server
2. Initiate image transfer. Run the following Redfish command:

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

**Note**

This command uses HTTP to download the image, initiates a soft reset on the BlueField, and pushes the boot stream. For NVIDIA-supplied BFBs, the eMMC is flashed automatically once the boot stream is pushed. Upon success, a `running` message is received.

**Info**

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

Where:

-
    - image_uri – contains both the HTTP server address and the exported path to the .bfb file on the server, with **one slash** between the two fields (i.e., <http_server>/<exported_path_of_bfb> ).

      **Info**

      For example, if <http_server> is 10.10.10.10 and <exported_path_of_bfb> is /tmp/new.bfb then "ImageURI":"10.10.10.10//tmp/new.bfb".

    - bmc_ip – BMC IP address

      Response/error messages:

        - 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 the HTTPS server address is wrong or the HTTPS service is not stated, an "Unknown Host" error is expected:

```
  {
    "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
    "Message": "Transfer of image 'new.bfb' to '/dev/rshim0/boot' failed.",
    "MessageArgs": [
      "new.bfb",
      "/dev/rshim0/boot"
    ],
    "MessageId": "Update.1.0.TransferFailed",
    "Resolution": "Unknown Host: Please provide server's public key using
PublicKeyExchange (for SCP download) or  Check and restart server's web service
(for HTTP/HTTPS download)",
    "Severity": "Critical"
  },
```

- If `TransferProtocol` or any other required field are wrong:

```
{
  "@Message.ExtendedInfo": [    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The parameter TransferProtocol for the action
UpdateService.SimpleUpdate is not supported on the target resource.",
      "MessageArgs": [
        "TransferProtocol",
        "UpdateService.SimpleUpdate"
      ],
      "MessageId": "Base.1.16.0.ActionParameterNotSupported",
      "MessageSeverity": "Warning",
      "Resolution": "Remove the parameter supplied and resubmit the request if the
operation failed."
    }
  ]
}
```

- If Targets or any other required field are missing:

```
{
  "Targets@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The create operation failed because the required property Targets
was missing from the request.",
      "MessageArgs": [
        "Targets"
      ],
      "MessageId": "Base.1.16.0.CreateFailedMissingReqProperties",
      "MessageSeverity": "Critical",
      "Resolution": "Correct the body to include the required property with a valid
value and resubmit the request if the operation failed."
    }
  ]
}
```

- Success message 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"
}
```

### Installing BFB File with HTTPS Protocol

1. Make sure the BFB file, new.bfb, is available on HTTPS server
2. Make sure the BMC has certificate to authenticate the HTTPS server. Or install a valid certificate to authenticate:

```
curl -c cjar -b cjar -k -u root:'<password>' -X POST
https://$bmc/redfish/v1/Managers/Bluefield_BMC/Truststore/Certificates -d @CAcert.json
```

3. Initiate image transfer. Run the following Redfish command:

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

**Note**

This command uses HTTPS for the image download, initiates a soft reset on the BlueField, and then pushes the boot stream. For NVIDIA-supplied BFBs, the eMMC is flashed automatically once the boot stream is pushed. Upon success, a `running` message is received.

**Info**

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

Where:

- 
    - `image_uri` – contains both the HTTPS server address and the exported path to the `.bfb` file on the server, with **one slash** between the two fields (i.e.,

<https_server>/<exported_path_of_bfb> ).

> **Info**
>
> For example, if <https_server> is urm.nvidia.com and <exported_path_of_bfb> is artifactory/sw-mlnx-bluefield-generic/Ubuntu22.04/new.bfb then "ImageURI":"10.126.206.42/artifactory/sw-mlnx-bluefield-generic/Ubuntu22.04/new.bfb".

- bmc_ip – BMC IP address

  Response / error messages:

-
  -
    - 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 the HTTPS server address is wrong or the HTTPS service is not stated, an "Unknown Host" error is expected:

```
  {
    "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
    "Message": "Transfer of image 'new.bfb' to '/dev/rshim0/boot' failed.",
    "MessageArgs": [
      "new.bfb",
      "/dev/rshim0/boot"
    ],
    "MessageId": "Update.1.0.TransferFailed",
    "Resolution": "Unknown Host: Please provide server's public key using
PublicKeyExchange (for SCP download) or  Check and restart server's web service
(for HTTP/HTTPS download)",
    "Severity": "Critical"
  },
```

- If `TransferProtocol` or any other required field are wrong:

```
{
  "@Message.ExtendedInfo": [    {
      "@odata.type": "#Message.v1_1_1.Message",
      "Message": "The parameter TransferProtocol for the action
UpdateService.SimpleUpdate is not supported on the target resource.",
      "MessageArgs": [
        "TransferProtocol",
        "UpdateService.SimpleUpdate"
      ],
      "MessageId": "Base.1.16.0.ActionParameterNotSupported",
      "MessageSeverity": "Warning",
      "Resolution": "Remove the parameter supplied and resubmit the request if the
operation failed."
    }
  ]
}
```

- 
    - 
        - If Targets or any other required field are missing:

```
{
 "Targets@Message.ExtendedInfo": [
   {
     "@odata.type": "#Message.v1_1_1.Message",
     "Message": "The create operation failed because the required property Targets was missing from the request.",
     "MessageArgs": [
       "Targets"
     ],
     "MessageId": "Base.1.16.0.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 HTTPS server fails to authenticate the current installed certificate:

```
    {
     "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
     "Message": "Transfer of image 'new.bfb' to '/dev/rshim0/boot' failed.",
     "MessageArgs": [
       "new.bfb",
       "/dev/rshim0/boot"
     ],
     "MessageId": "Update.1.0.TransferFailed",
     "Resolution": "Bad Certificate: Please check the remote server certification, correct and replace the current installed one",
     "Severity": "Critical"
   },
```

- 
    - 
        - Success message 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"
          }
          ```

**Tracking Image Transfer Status and Progress for HTTP/HTTPS Protocols**

The following section is relevant for HTTP/HTTPS protocols which received a success message of a valid `SimpleUpdate` request and a `running` task state.

Run the following Redfish command to track image transfer status and progress:

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

Example:

```
    {
      "@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
      "Message": "Image 'new.bfb' is being transferred to '/dev/rshim0/boot'.",
      "MessageArgs": [
        "new.bfb",
        "/dev/rshim0/boot"
      ],
      "MessageId": "Update.1.0.TransferringToComponent",
      "Resolution": "Transfer started",
      "Severity": "OK"
    },
```

```
...

"PercentComplete": 60,
"StartTime": "2024-06-10T19:39:03+00:00",
"TaskMonitor": "/redfish/v1/TaskService/Tasks/1/Monitor",
"TaskState": "Running",
"TaskStatus": "OK"
```

### Direct SCP

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

If `bf.cfg` is required as part of the boot process, run:

```
cat <path_to_bfb> bf.cfg > new.bfb
scp <path to new.bfb> root@<bmc_ip>:/dev/rshim0/boot
```

## Tracking Installation Progress and Status

After image transfer is complete, users may follow the installation progress and status with the help of a dump of current the RShim miscellaneous messages log.

1. Initiate request for dump download:

   ```
   sudo curl -k -u root:'<password>' -d '{"DiagnosticDataType": "Manager"}' -X POST
   https://<ip_address>/redfish/v1/Managers/Bluefield_BMC/LogServices/Dump/Actions/LogService.Co
   ```

   Where:

   - `<ip-address>` – BMC IP address

- <password> – BMC password

2. Use the received task ID to poll for dump completion:

```
sudo curl -k -u root:'<password>' -H 'Content-Type: application/json' -X GET
https://<ip_address>/redfish/v1/TaskService/Tasks/<task_id>
```

   Where:

   - <ip-address> – BMC IP address
   - <password> – BMC password
   - <task_id> – Task ID received from the first command

3. Once dump is complete, download and review the dump:

```
sudo curl -k -u root:'<password>' -H 'Content-Type: application/json' -X GET
https://<ip_address>/redfish/v1/Managers/Bluefield_BMC/LogServices/Dump/Entries/<entry_id>/at
--output </path/to/tar/log_dump.tar.xz>
```

   Where:

   - <ip-address> – BMC IP address
   - <password> – BMC password
   - <entry_id> – The entry ID of the dump in redfish/v1/Managers/Bluefield_BMC/LogServices/Dump/Entries
   - </path/to/tar/log_dump.tar.xz> – path to download the log dump log_dump.tar.xz

4. Untar the file to review the logs. For example:

```
tar xvfJ log_dump.tar.xz
```

5. The log is contained in the `rshim.log` file. The log displays `Reboot`, `finished`, `DPU is ready`, or `In Enhanced NIC mode` when BFB installation completes.

> **Note**
>
> If the downloaded log file does not contain any of these strings, keep downloading the log file until they appear.

6. When installation is complete, you may crosscheck the new BFB version against the version provided to verify a successful upgrade:

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

Example response:

```
"@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/DPU_OS",
 "@odata.type": "#SoftwareInventory.v1_4_0.SoftwareInventory",
 "Description": "Host image",
 "Id": "DPU_OS",
 "Members@odata.count": 1,
 "Name": "Software Inventory",
 "RelatedItem": [
   {
     "@odata.id": "/redfish/v1/Systems/Bluefield/Bios"
   }
 ],
 "SoftwareId": "",
 "Status": {
   "Conditions": [],
   "Health": "OK",
   "HealthRollup": "OK",
   "State": "Enabled"
 },
 "Updateable": true,
"Version": "DOCA_2.2.0_BSP_4.2.1_Ubuntu_22.04-8.23-07"
```

**Note**

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

## 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.

**Note**

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
```

## 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 BlueField. 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>
   ```

   **Note**

   **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.

## 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.
```

**Note**

For now, please ignore tool's instruction to reboot

**Note**

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

```
mst start
mst status
```

Example output taken on a multiple BlueField 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 devices 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 is Ethernet only. Please refer to section "Supported Platforms and Interoperability" under the Release Notes to learn your BlueField type.

   If you have an InfiniBand-and-Ethernet-capable BlueField, 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:

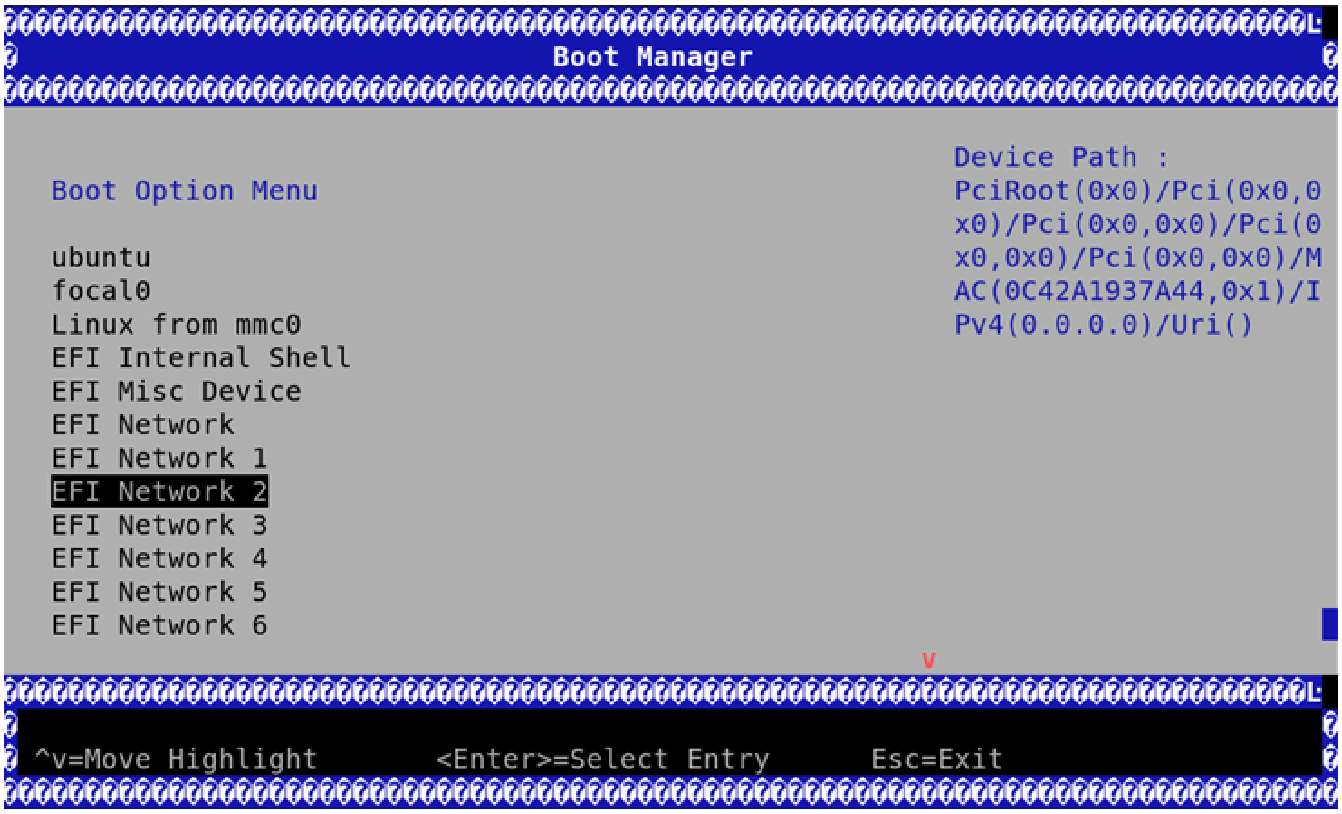
```
sudo mlxconfig -d <MST device> -y s LINK_TYPE_P1=2 LINK_TYPE_P2=2
```

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

# Deploying BlueField Software Using PXE

The following steps detail the PXE deployment 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

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

**Note**

To set up a PXE server, please refer to the documentation provided by the distribution vendor. For example, to install Ubuntu 20.04 or later, see official Ubuntu 20.04 documentation.

## Deploying BlueField Software Using BFB with PXE

**Info**

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.

**Note**

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

The following are the steps to prepare a PXE server to deploy a BFB bundle:

1. Provide the image of the BFB file. Run:

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

For example:

```
# mlx-mkbfb -x DOCA_2.7.0_BSP_4.7.0_Ubuntu_22.04-<version>.bfb
```

**Note**

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. Create a boot entry in the PXE server. 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
}
```

bfks is a BASH script that runs alongside 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";
}
```

# Deploying BlueField Software Using ISO with PXE

BlueField software (including Ubuntu OS), NIC firmware, and BMC software can be deployed using an ISO image similarly to the standard Ubuntu deployment method using ISO. The BlueField ISO image is based on the standard Ubuntu ISO image for Arm64 with an updated kernel and added DOCA packages.

## PXE Server Setup

Mount the ISO:

```
$ mount bf-bundle-2.7.0085-1-2024-06-14-22-36-50.iso /mnt
$ cp /mnt/casper/vmlinuz /var/lib/tftpboot/boot/
$  cp /mnt/casper/initrd /var/lib/tftpboot/boot/
```

Example of grub.cfg:

```
menuentry "Install BF OS" {
   linux /boot/vmlinuz autoinstall fsck.mode=skip no-snapd  console=hvc0 console=ttyAMA0
earlycon=pl011,0x13010000 net.ifnames=0 biosdevname=0 iommu.passthrough=1 ip=dhcp
url=http://<HTTP server IP>/jammy/ISO/bf-bundle-2.7.0085-1-2024-06-14-22-36-50.iso bfnet=eth0:dhcp
bfks=http://<HTTP server IP>/jammy/ISO/bfks
   initrd /boot/initrd
}
```

The bf.cfg file can be used to customize the installation procedure. To create bf.cfg on the BlueField to be used for the installation use the bfks parameter to point to the script located on HTTP server that will create bf.cfg file:

bfks example:

```
cat > /etc/bf.cfg << 'EOF'
BMC_PASSWORD="..."
```

```
EOF
```

Standard automatic Ubuntu installation using `autoinstall.yaml` is also supported. See Introduction to autoinstall - Ubuntu installation documentation.

Example of `autoinstall.yaml` that can be used to customize the installation and modify `bf.cfg`:

Example of a `grub.cfg` with `autoinstall.yaml`:

```
menuentry "Install BF OS" {
  linux /boot/vmlinuz autoinstall fsck.mode=skip no-snapd  console=hvc0 console=ttyAMA0
earlycon=pl011,0x13010000 net.ifnames=0 biosdevname=0 iommu.passthrough=1 ip=dhcp
url=http://<HTTP server IP>/jammy/ISO/bf-bundle-2.7.0085-1-2024-06-14-22-36-50.iso force-
ai=http://<HTTP server IP>/jammy/ISO/autoinstall.yaml cloud-config-url=/dev/null
   initrd /boot/initrd
}
```

Example of `autoinstall.yaml`:

```
version: 1


apt:
  preserve_sources_list: false
  conf: |
    Dpkg::Options {
      "--force-confdef";
      "--force-confold";
    };

storage:
  swap:
    size: 0
  grub:
    reorder_uefi: true
  config:
  - id: nvme0n1
    type: disk
    ptable: gpt
    path: /dev/nvme0n1
    name: osdisk
    wipe: superblock-recursive

  - id: nvme0n1-part1
    type: partition
    device: nvme0n1
    number: 1
    size: 50MB
    flag: boot
    grub_device: true

  - id: nvme0n1-part1-fs1
    type: format
    fstype: fat32
    label: efi
    volume: nvme0n1-part1

  - id: nvme0n1-part2
    type: partition
    device: nvme0n1
    number: 2
    size: -1

  - id: nvme0n1-part2-fs1
    type: format
    fstype: ext4
    label: root
    volume: nvme0n1-part2

  - id: nvme0n1-mount
    type: mount
    path: /
    device: nvme0n1-part2-fs1
    options: defaults
    passno: 0
    fstype: auto

  - id: nvme0n1-boot-mount
    type: mount
    path: /boot/efi
    device: nvme0n1-part1-fs1
    options: umask=0077
    passno: 1
```

```
reporting:
  builtin:
    type: print

# Add user-data so that subiquity doesn't complain about us not
# having a identity section
user-data:
  debug:
    verbose: true
  write_files:
    - path: /etc/iptables/rules.v4
      permissions: '0644'
      owner: 'root:root'
      content: |
        *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
 users:
  - name: ubuntu
    lock_passwd: False
    groups: adm, audio, cdrom, dialout, dip, floppy, lxd, netdev, plugdev, sudo, video
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash
    plain_text_passwd: 'ubuntu'
 chpasswd:
  list: |
      ubuntu:ubuntu
    expire: True
  no_ssh_fingerprints: true
  runcmd:
    - [ /usr/sbin/netfilter-persistent, start ]
    - [ /opt/mellanox/doca/services/telemetry/import_doca_telemetry.sh ]
    - [ /usr/bin/bfrshlog, "INFO: DPU is ready" ]

late-commands:
  # write release file
  - |
    cat << EOF > /target/etc/bf-release
    BF_NAME="Mellanox Bluefield"
    BF_PRETTY_NAME="Mellanox Bluefield"
    BF_SWBUILD_TIMESTAMP="2024-06-12-12-47-25"
    BF_SWBUILD_VERSION="2.7.0085-1"
    BF_COMMIT_ID="7fce146"
    BF_PLATFORM="BlueField SoC"
    BF_SERIAL_NUMBER="1332723060006"
    EOF

  # mount cdrom
  - mkdir -p /target/tmp/cdrom
  - mount --bind /cdrom /target/tmp/cdrom || true
  - |
    cat << EOF > /target/etc/apt/sources.list
    deb [check-date=no] file:///tmp/cdrom/ jammy main restricted
    EOF

  # avoid running flash kernel after install kernel
  - mkdir -p /target/run/systemd
  - echo docker > /target/run/systemd/container

  # Install packages
  - curtin in-target -- apt update -y
  - curtin in-target -- apt remove -y --purge `dpkg --list | grep openipmi | awk '{print $2}'`
  - curtin in-target -- /bin/bash -c "DEBIAN_FRONTEND=noninteractive RUN_FW_UPDATER=no apt-get
install --no-install-recommends -y acpid bc binutils bridge-utils build-essential cracklib-runtime dc
docker.io flash-kernel i2c-tools ifenslave iperf3 iptables-persistent iputils-arping iputils-ping iputils-
tracepath kexec-tools libpam-pwquality libssl-dev lldpad lm-sensors net-tools network-manager nfs-
common nvme-cli openssh-server python3.10 python3-pyinotify python3-pip rasdaemon rsync
sbsigntool shim-signed tcpdump watchdog doca-runtime doca-devel containerd kubelet runc nv-
common-apis nvidia-repo-keys linux-bluefield-modules-bluefield linux-image-5.15.0-1042-bluefield"
```

```
  # rewrite sources
  - |
    cat << EOF > /target/etc/apt/sources.list
    deb http://ports.ubuntu.com/ubuntu-ports/ jammy main restricted universe multiverse
    deb http://ports.ubuntu.com/ubuntu-ports/ jammy-updates main restricted universe multiverse
    deb http://ports.ubuntu.com/ubuntu-ports/ jammy-security main restricted universe multiverse
    EOF

  # Allow cloud-init to configure networking
  - find /target/etc/cloud/cloud.cfg.d/ -type f ! -name README ! -name 05_logging.cfg ! -name 90_dpkg.cfg
-delete || true;
  - curtin in-target -- cloud-init clean

  # Post-installation steps
  # Create bf.cfg
  - |
    cat << EOF > /target/etc/bf.cfg
    # 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"


###############################################################################
    # 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 BlueField 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: no)
   WITH_NIC_FW_UPDATE="yes"
   EOF

   # Run post-installation script to update ATF/UEFI, NIC firmware and BMC components
  - curtin in-target -- /bin/bash -c "device=/dev/nvme0n1 /usr/local/sbin/bfiso-post-install.sh || true"

  - curtin in-target -- systemctl disable snapd
```

## PXE Sequence with Redfish

HTTP boot configuration can be done using the BlueField BMC's Redfish interface.

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 "Deploying BlueField Software Using BFB with PXE".

2. Check the current boot override settings by performing a GET on the `ComputerSystem` schema over 1GbE to the BlueField BMC. Look for the `"Boot"` property.

```
curl -k -X GET -u root:<password> https://<BF-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"
        ],
  ....
}
```

**Info**

Boot override enables overriding the first boot source, either once or continuously.

3. The example 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, then boot override is disabled after the first boot, and any related properties are reset to their former values to avoid repetition. If it is set to Continuous, then on every reboot the BlueField keeps performing boot override (HTTPBoot).

5. To perform boot override, perform a PATCH to pending settings URI over the 1GbE to the BlueField BMC.

```
curl -k -X PATCH -d '{"Boot": {"BootSourceOverrideEnabled":"Once",
"BootSourceOverrideMode":"UEFI", "BootSourceOverrideTarget": "UefiHttp",
"HttpBootUri":"http://<HTTP-Server-Ip>/Image.iso"}}' -u root:<password> https://<BF-BMC-
IP>/redfish/v1/Systems/<SystemID>/Settings | python3 -m json.tool
```

For example:

```
curl -k -X GET -u root:<password> https://<BF-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 BlueField using the Redfish Manager schema over the 1GbE to the BlueField BMC:

```
curl -k -u root:<password> -H "Content-Type: application/json" -X POST https://<BF-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 the 1GbE OOB to the BlueField BMC.

**Note**

The `HttpBootUri` property is parsed by the Redfish server and the URI is presented to the BlueField as part of DHCP lease when the BlueField performs the HTTP boot.

# Customizing BlueField Software Deployment Using bf.cfg

bf.cfg is an optional configuration file which may be used to customize the software deployment process on NVIDIA® BlueField® networking platforms (DPU or SuperNIC).

**Note**

To update the BMC components, it is required to provide the BMC_PASSWORD using bf.cfg to the BFB/ISO installation environment.

There are different ways to pass bf.cfg along with the BFB or ISO to customize the installation procedure:

- With BFB from the host:

```
# bfb-install -r <rshim device> -c <path to bf.cfg> -b <BFB>
```

- Using cat command:

```
# cat <BFB> <path to bf.cfg> > /dev/<rshim device>/boot
```

- By appending bf.cfg to the BFB and push it to RShim device on a host or BMC:

```
# cat <BFB> <path to bf.cfg> > <new BFB>
```

- In PXE environment using `bfks` parameter to provide a script that will be downloaded by the installation process and run on the Bluefield side at the beginning of installation:

```
cat > /etc/bf.cfg << 'EOF'
BMC_PASSWORD="..."
EOF
```

- Or using `autoinstall.yaml`. See " Deploying BlueField Software Using ISO with PXE" for details.

## Changing Default Credentials for "ubuntu" User via bf.cfg

**Info**

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`).

**Note**

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.

## Changing UEFI Password Using bf.cfg

To change UEFI password add current UEFI password `UEFI_PASSWORD` and the new UEFI password `NEW_UEFI_PASSWORD` to `bf.cfg`.

## Changing BMC Password Using bf.cfg

To change BMC root password, add current BMC root password `BMC_PASSWORD` and the new BMC root password `NEW_BMC_PASSWORD` to `bf.cfg`.

## Advanced 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 BlueField 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 =====================
}
```

## 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"

# To change UEFI password set UEFI_PASSWORD to its current value and NEW_UEFI_PASSWORD to the
new UEFI password (clear text).
UEFI_PASSWORD=<current UEFI password>
NEW_UEFI_PASSWORD=<new UEFI password>

# UPDATE_DPU_OS - Update/Install BlueField 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.sha512.10000.5EB1FF92FDD89BDAF3395174282C77430656A6DBE
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=""

# NEW_BMC_PASSWORD - can be used to change BMC_PASSWORD to the new one (Default: None)
# Note: current BMC_PASSWORD is required
NEW_BMC_PASSWORD=<new 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 BlueField 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 BlueField Arm 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.
```

# Deploying NVIDIA Converged Accelerator

**Info**

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 BlueField Software Using BFB from Host
- Deploying BlueField Software Using BFB from BMC
- Deploying BlueField Software Using PXE

NVIDIA® CUDA® (GPU driver) must be installed to use the GPU. For information on how to install CUDA on your Converged Accelerator, refer to NVIDIA CUDA Installation Guide for Linux.

## 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® and the GPU operate separately (GPU is owned by the host)
- BlueField-X – the GPU is exposed to BlueField and is no longer visible on the host (GPU is owned by BlueField)

**Note**

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
```

## 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. P erform a BlueField system-level reset for the mlxconfig settings to take effect.

## Standard Mode

To return BlueField 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. P erform a BlueField system-level reset for the mlxconfig settings to take effect.

## 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)
```

## Verifying GPU Ownership

The following are example outputs for when BlueField 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 BlueField:

```
ubuntu@bf:~$ lspci | grep -i nv
06:00.0 3D controller: NVIDIA Corporation GA20B8 (rev a1)
```

## GPU Firmware

## Get GPU Firmware

```
smbpbi: (See SMBPBI spec)

root@bf:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x00 0x80 s
root@bf:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x00 0x5f
root@bf:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x39 0x32 0x2e 0x30
root@bf:~#
root@bf:~#
```

```
root@bf:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x01 0x80 s
root@bf:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x01 0x5f
root@bf:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x2e 0x36 0x42
root@bf:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x02 0x80 s
root@bf:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x02 0x5f
root@bf:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x2e 0x30 0x30 0x2e
root@bf:~# i2cset -y 3 0x4f 0x5c 0x05 0x08 0x03 0x80 s
root@bf:~# i2cget -y 3 0x4f 0x5c ip 5
5: 0x04 0x05 0x08 0x03 0x5f
root@bf:~# i2cget -y 3 0x4f 0x5d ip 5
5: 0x04 0x30 0x31 0x00 0x00
root@bf:~#

39 32 2e 30 30 2e 36 42 2e 30 30 2e 30 31 00 00    92.00.6B.00.01
```

## Updating GPU Firmware

```
root@bf:~# 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@bf:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@bf:~# cat /tmp/gpu_images/progress.txt
TaskState="Running"
TaskStatus="OK"
TaskProgress="50"

root@bf:~# cat /tmp/gpu_images/progress.txt
TaskState=Frimware update succeeded.
TaskStatus=OK
```

```
TaskProgress=100
```

# Installing Repo Package on Host Side

**Note**

This section assumes that an NVIDIA® BlueField® networking platform (DPU or SuperNIC) has already been installed in a server according to the instructions detailed in the BlueField's hardware user guide.

The following procedure instructs users on upgrading DOCA local repo package for host.

## Removing Previously Installed DOCA Runtime Packages

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

| | |
|---|---|
| Deb-based | `$ for f in $( dpkg --list \| grep doca \| awk '{print $2}' ); do echo $f ; apt remove --purge $f -y ; done`<br>`$ /usr/sbin/ofed_uninstall.sh --force`<br>`$ sudo apt-get autoremove` |
| **RPM-based** | `host# for f in $(rpm -qa \| grep -i doca ) ; do yum -y remove $f; done`<br>`host# /usr/sbin/ofed_uninstall.sh --force`<br>`host# yum autoremove`<br>`host# yum makecache` |

Then perform the following steps:

**Note**

The following procedure is valid for RPM-based OS only.

1. Download NVIDIA's RPM-GPG-KEY-Mellanox-SHA256 key:

```
# wget http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox-SHA256
--2018-01-25 13:52:30--  http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox-SHA256
Resolving www.mellanox.com... 72.3.194.0
Connecting to www.mellanox.com|72.3.194.0|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1354 (1.3K) [text/plain]
Saving to: ?RPM-GPG-KEY-Mellanox-SHA256?

100%[=================================================>] 1,354       --.-K/s   in 0s

2018-01-25 13:52:30 (247 MB/s) - ?RPM-GPG-KEY-Mellanox-SHA256? saved [1354/1354]
```

2. Install the key:

```
# sudo rpm --import RPM-GPG-KEY-Mellanox-SHA256
warning: rpmts_HdrFromFdno: Header V3 DSA/SHA1 Signature, key ID 6224c050: NOKEY
Retrieving key from file:///repos/MLNX_OFED//RPM-GPG-KEY-Mellanox
Importing GPG key 0x6224C050:
 Userid: "Mellanox Technologies (Mellanox Technologies - Signing Key v2) "
 From  : /repos/MLNX_OFED//RPM-GPG-KEY-Mellanox-SHA256
Is this ok [y/N]:
```

3. Verify that the key was successfully imported:

```
# rpm -q gpg-pubkey --qf '%{NAME}-%{VERSION}-%{RELEASE}\t%{SUMMARY}\n' | grep Mellanox
gpg-pubkey-a9e4b643-520791ba    gpg(Mellanox Technologies )
```

## 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-2.8.0-204000_24.07_alinux32.x86_64.rpm |
| Anolis | aarch64 | doca-host-2.8.0-204000_24.07_anolis86.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_anolis86.x86_64.rpm |
| BCLinux 21.10 | aarch64 | doca-host-2.8.0-204000_24.07_bclinux2210.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_bclinux2210.x86_64.rpm |
| BCLinux 21.10 SP2 | aarch64 | doca-host-2.8.0-204000_24.07_bclinux2110sp2.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_bclinux2110sp2.x86_64.rpm |
| CTyunOS 2.0 | aarch64 | doca-host-2.8.0-204000_24.07_ctyunos20.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_ctyunos20.x86_64.rpm |
| CTyunOS 23.01 | aarch64 | doca-host-2.8.0-204000_24.07_ctyunos2301.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_ctyunos2301.x86_64.rpm |
| Debian 10.13 | aarch64 | doca-host_2.8.0-204000-24.07-debian1013_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-debian1013_amd64.deb |
| Debian 10.8 | aarch64 | doca-host_2.8.0-204000-24.07-debian108_arm64.deb |

| OS | Arch | Link |
|---|---|---|
| | x86 | doca-host_2.8.0-204000-24.07-debian108_amd64.deb |
| Debian 10.9 | x86 | doca-host_2.8.0-204000-24.07-debian109_amd64.deb |
| Debian 11.3 | aarch64 | doca-host_2.8.0-204000-24.07-debian113_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-debian113_amd64.deb |
| Debian 12.1 | aarch64 | doca-host_2.8.0-204000-24.07-debian121_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-debian121_amd64.deb |
| Debian 12.5 | aarch64 | doca-host_2.8.0-204000-24.07-debian125_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-debian125_amd64.deb |
| EulerOS 20 SP11 | aarch64 | doca-host-2.8.0-204000_24.07_euleros20sp11.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_euleros20sp11.x86_64.rpm |
| EulerOS 20 SP12 | aarch64 | doca-host-2.8.0-204000_24.07_euleros20sp12.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_euleros20sp12.x86_64.rpm |
| Fedora32 | x86 | doca-host-2.8.0-204000_24.07_fc32.x86_64.rpm |
| Kylin 1.0 SP2 | aarch64 | doca-host-2.8.0-204000_24.07_kylin10sp2.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_kylin10sp2.x86_64.rpm |
| Kylin 1.0 SP3 | aarch64 | doca-host-2.8.0-204000_24.07_kylin10sp3.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_kylin10sp3.x86_64.rpm |
| Mariner 2.0 | x86 | doca-host-2.8.0-204000_24.07_mariner20.x86_64.rpm |
| Oracle Linux 7.9 | x86 | doca-host-2.8.0-204000_24.07_ol79.x86_64.rpm |
| Oracle Linux 8.4 | x86 | doca-host-2.8.0-204000_24.07_ol84.x86_64.rpm |
| Oracle Linux 8.6 | x86 | doca-host-2.8.0-204000_24.07_ol86.x86_64.rpm |

| OS | Arch | Link |
|---|---|---|
| Oracle Linux 8.7 | x86 | doca-host-2.8.0-204000_24.07_ol87.x86_64.rpm |
| Oracle Linux 8.8 | x86 | doca-host-2.8.0-204000_24.07_ol88.x86_64.rpm |
| Oracle Linux 9.1 | x86 | doca-host-2.8.0-204000_24.07_ol91.x86_64.rpm |
| Oracle Linux 9.2 | x86 | doca-host-2.8.0-204000_24.07_ol92.x86_64.rpm |
| openEuler 20.03 SP3 | aarch64 | doca-host-2.8.0-204000_24.07_openeuler2003sp3.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_openeuler2003sp3.x86_64.rpm |
| openEuler 22.03 | aarch64 | doca-host-2.8.0-204000_24.07_openeuler2203.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_openeuler2203.x86_64.rpm |
| openEuler 22.03 SP1 | x86 | doca-host-2.8.0-204000_24.07_openeuler2203sp1.x86_64.rpm |
| RHEL/CentOS 8.0 | aarch64 | doca-host-2.8.0-204000_24.07_rhel80.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel80.x86_64.rpm |
| RHEL/CentOS 8.1 | aarch64 | doca-host-2.8.0-204000_24.07_rhel81.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel81.x86_64.rpm |
| RHEL/CentOS 8.2 | aarch64 | doca-host-2.8.0-204000_24.07_rhel82.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel82.x86_64.rpm |
| RHEL/CentOS 8.3 | aarch64 | doca-host-2.8.0-204000_24.07_rhel83.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel83.x86_64.rpm |
| RHEL/CentOS 8.4 | aarch64 | doca-host-2.8.0-204000_24.07_rhel84.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel84.x86_64.rpm |

| OS | Arch | Link |
|---|---|---|
| RHEL/CentOS 8.5 | aarch64 | doca-host-2.8.0-204000_24.07_rhel85.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel85.x86_64.rpm |
| RHEL/Rocky 8.6 | aarch64 | doca-host-2.8.0-204000_24.07_rhel86.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel86.x86_64.rpm |
| RHEL/Rocky 8.7 | aarch64 | doca-host-2.8.0-204000_24.07_rhel87.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel87.x86_64.rpm |
| RHEL/Rocky 8.8 | aarch64 | doca-host-2.8.0-204000_24.07_rhel88.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel88.x86_64.rpm |
| RHEL/Rocky 8.9 | aarch64 | doca-host-2.8.0-204000_24.07_rhel89.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel89.x86_64.rpm |
| RHEL/Rocky 8.10 | aarch64 | doca-host-2.8.0-204000_24.07_rhel810.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel810.x86_64.rpm |
| RHEL/Rocky 9.0 | aarch64 | doca-host-2.8.0-204000_24.07_rhel90.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel90.x86_64.rpm |
| RHEL/Rocky 9.1 | aarch64 | doca-host-2.8.0-204000_24.07_rhel91.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel91.x86_64.rpm |
| RHEL/Rocky 9.2 | aarch64 | doca-host-2.8.0-204000_24.07_rhel92.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel92.x86_64.rpm |
| RHEL/Rocky 9.3 | aarch64 | doca-host-2.8.0-204000_24.07_rhel93.aarch64.rpm |

| OS | Arch | Link |
|---|---|---|
| | x86 | doca-host-2.8.0-204000_24.07_rhel93.x86_64.rpm |
| RHEL/Rocky 9.4 | aarch64 | doca-host-2.8.0-204000_24.07_rhel94.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_rhel94.x86_64.rpm |
| SLES 15 SP2 | aarch64 | doca-host-2.8.0-204000_24.07_sles15sp2.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_sles15sp2.x86_64.rpm |
| SLES 15 SP3 | aarch64 | doca-host-2.8.0-204000_24.07_sles15sp3.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_sles15sp3.x86_64.rpm |
| SLES 15 SP4 | aarch64 | doca-host-2.8.0-204000_24.07_sles15sp4.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_sles15sp4.x86_64.rpm |
| SLES 15 SP5 | aarch64 | doca-host-2.8.0-204000_24.07_sles15sp5.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_sles15sp5.x86_64.rpm |
| SLES 15 SP6 | x86 | doca-host-2.8.0-204000_24.07_sles15sp6.x86_64.rpm |
| TencentOS 3.3 | aarch64 | doca-host-2.8.0-204000_24.07_tencentos33.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_tencentos33.x86_64.rpm |
| Ubuntu 20.04 | aarch64 | doca-host_2.8.0-204000-24.07-ubuntu2004_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-ubuntu2004_amd64.deb |
| Ubuntu 22.04 | aarch64 | doca-host_2.8.0-204000-24.07-ubuntu2204_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-ubuntu2204_amd64.deb |
| Ubuntu 24.04 | aarch64 | doca-host_2.8.0-204000-24.07-ubuntu2404_arm64.deb |
| | x86 | doca-host_2.8.0-204000-24.07-ubuntu2404_amd64.deb |

| OS | Arch | Link |
|---|---|---|
| UOS20.1060 | aarch64 | doca-host-2.8.0-204000_24.07_uos201060.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_uos201060.x86_64.rpm |
| UOS20.1060A | aarch64 | doca-host-2.8.0-204000_24.07_uos201060a.aarch64.rpm |
| | x86 | doca-host-2.8.0-204000_24.07_uos201060a.x86_64.rpm |
| XenServer 8.2 | x86 | doca-host-2.8.0-204000_24.07_xenserver82.x86_64.rpm |

## Installing Local Repo Package for Host Dependencies

1. Install DOCA local repo package for host:

| OS | Procedure |
|---|---|
| Ubuntu | 1. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the host.<br>2. Unpack the deb repo. Run:<br>`host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb`<br>3. Perform apt update. Run:<br>`host# sudo apt-get update`<br>4. Run `apt install` for DOCA runtime, tools, and SDK:<br>`host# sudo apt install -y doca-runtime doca-sdk` |
| CentOS | 1. Download the DOCA SDK and DOCA Runtime packages from Downloading DOCA Runtime Packages section for the x86 host.<br>2. Install the following software dependencies. Run:<br>`host# sudo yum install -y epel-release`<br>3. For CentOS 8.2 only, also run: |

| OS | Procedure |
|---|---|
|  | <pre>host# yum config-manager --set-enabled PowerTools</pre> 4. Unpack the RPM repo. Run: <pre>host# sudo rpm -Uvh doca-host-repo-rhel&lt;version&gt;.x86_64.rpm</pre> 5. Run `yum install` for DOCA runtime, tools, and SDK. <pre>host# sudo yum install -y doca-runtime doca-sdk</pre> |
| RHEL | 1. Open a RedHat account.<br>    1. Log into RedHat website via the developers tab.<br>    2. Create a developer user.<br>2. Run: <pre>host# subscription-manager register --username=&lt;username&gt; --<br>password=PASSWORD</pre> To extract pool ID: <pre>host# subscription-manager list --available --all<br>...<br>Subscription Name:   Red Hat Developer Subscription for Individuals<br>Provides:            Red Hat Developer Tools (for RHEL Server for ARM)<br>                     ...<br>                     Red Hat CodeReady Linux Builder for x86_64<br>...<br>Pool ID:             &lt;pool-id&gt;<br>...</pre> And use the pool ID for the `Subscription Name` and `Provides` that include `Red Hat CodeReady Linux Builder for x86_64`.<br>3. Run: <pre>host# subscription-manager attach --pool=&lt;pool-id&gt;<br>host# subscription-manager repos --enable codeready-builder-for-rhel-8-<br>x86_64-rpms</pre> |

| OS | Procedure |
|---|---|
| | `host# yum makecache`<br>4. Install the DOCA local repo package for host. Run:<br>`host# rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm`<br>`host# sudo yum install -y doca-runtime doca-sdk`<br>5. Sign out from your RHEL account. Run:<br>`host# subscription-manager remove --all`<br>`host# subscription-manager unregister` |

2. Verify that RShim is active.

```
host# sudo systemctl status rshim
```

This command is expected to display `incative (dead)`.

- To launch RShim service, run:

```
host# sudo systemctl start rshim
```

- To allow RShim to launch automatically in future boots, run:

```
host# sudo systemctl enable rshim
```

3. A ssign a dynamic IP to `tmfifo_net0` interface (RShim host interface):

```
host# ifconfig tmfifo_net0 192.168.100.1 netmask 255.255.255.252 up
```

# Installing Popular Linux Distributions on BlueField

## Building Your Own BFB Installation Image

Users wishing to build their own customized NVIDIA® BlueField® networking platform's (DPU or SuperNIC) OS image can use the BFB build environment. See this GitHub webpage for more information.

**Note**

For any customized BlueField OS image to boot on the UEFI secure-boot-enabled BlueField (default BlueField 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.

## Installing Linux Distributions

Contact NVIDIA Enterprise Support for information on the installation of Linux distributions other than Ubuntu.

## 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) | | |
| 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 BlueField 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 | | |

| Driver | Description | BlueField-2 | BlueField-3 |
|---|---|---|---|
|  | 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 |  |  |

# Updating BlueField Software Packages Using Standard Linux Tools

This upgrade procedure enables upgrading DOCA components using standard Linux tools (e.g., `apt update` and `yum update`). This process utilizes native package manager repositories to upgrade BlueField networking platforms (DPUs or SuperNICs) without the need for a full installation.

This process has the following benefits :

- Only updates components that include modifications
  - Configurable – user can select specific components (e.g., UEFI-ATF, NIC-FW)
- Includes upgrade of:
  - DOCA drivers and libraries
  - DOCA reference applications
  - BSP (UEFI/ATF) upgrade while maintaining the configuration
  - NIC firmware upgrade while maintaining the configuration
- Does not:
  - Impact user binaries
  - Upgrade non-Ubuntu OS kernels
  - Upgrade BlueField BMC firmware
- After completion of BlueField upgrade:

- If NIC firmware was not updated, perform BlueField Arm reset (software reset/reboot BlueField )
- If NIC firmware was updated, perform firmware reset (`mlxfwreset`) or perform a graceful shutdown and power cycle

| OS | Action | Instructions |
|---|---|---|
| Ubuntu/Debian | Remove `mlxbf-bootimages` package | `<bf> $ apt remove --purge mlxbf-bootimages* -y` |
| | Install the the GPG key | `<bf> $ apt update`<br>`<bf> $ apt install gnupg2` |
| | Export the desired distribution | Export `DOCA_REPO` with the relevant URL. The following is an example for Ubuntu 22.04:<br>`<bf> $ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.8.0/ubuntu22.04/dpu-arm64"`<br>• Ubuntu 22.04 – https://linux.mellanox.com/public/repo/doca/2.8.0/ubuntu22.04/dpu-arm64<br>• Ubuntu 20.04 – https://linux.mellanox.com/public/repo/doca/2.8.0/ubuntu20.04/dpu-arm64<br>• Debian 12 – https://linux.mellanox.com/public/repo/doca/2.8.0/debian12/dpu-arm64 |
| | Add GPG key to APT trusted keyring | `<bf> $ curl $DOCA_REPO/GPG-KEY-Mellanox.pub \| gpg --dearmor > /etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub` |

| OS | Action | Instructions |
|---|---|---|
| | Add DOCA online repository | `<bf> $ echo "deb [signed-by=/etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub] $DOCA_REPO ./" > /etc/apt/sources.list.d/doca.list` |
| | Update index | `<bf> $ apt update` |
| | Upgrade UEFI/ATF firmware | Run: `<bf> $ apt install mlxbf-bootimages-signed` Then i nitiate upgrade for UEFI/ATF firmware: `<bf> $ apt install mlxbf-scripts` `<bf> $ bfrec` |
| | Upgrade BlueField NIC firmware | The following commands update the firmware package and flash the firmware to the NIC: `<bf> $ apt install mlnx-fw-updater-signed` `<bf> $ sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update` |
| | Remove old metapackages | `<bf> $ apt-get remove doca* mlnx-ofed* kernel-mft* -y` |
| | Install new metapackages | `<bf> $ apt-get install doca-runtime doca-devel -y` |
| | Upgrade system | `<bf> $ apt upgrade` |
| | Apply the new changes, | For the upgrade to take effect, perform BlueField system reboot. |

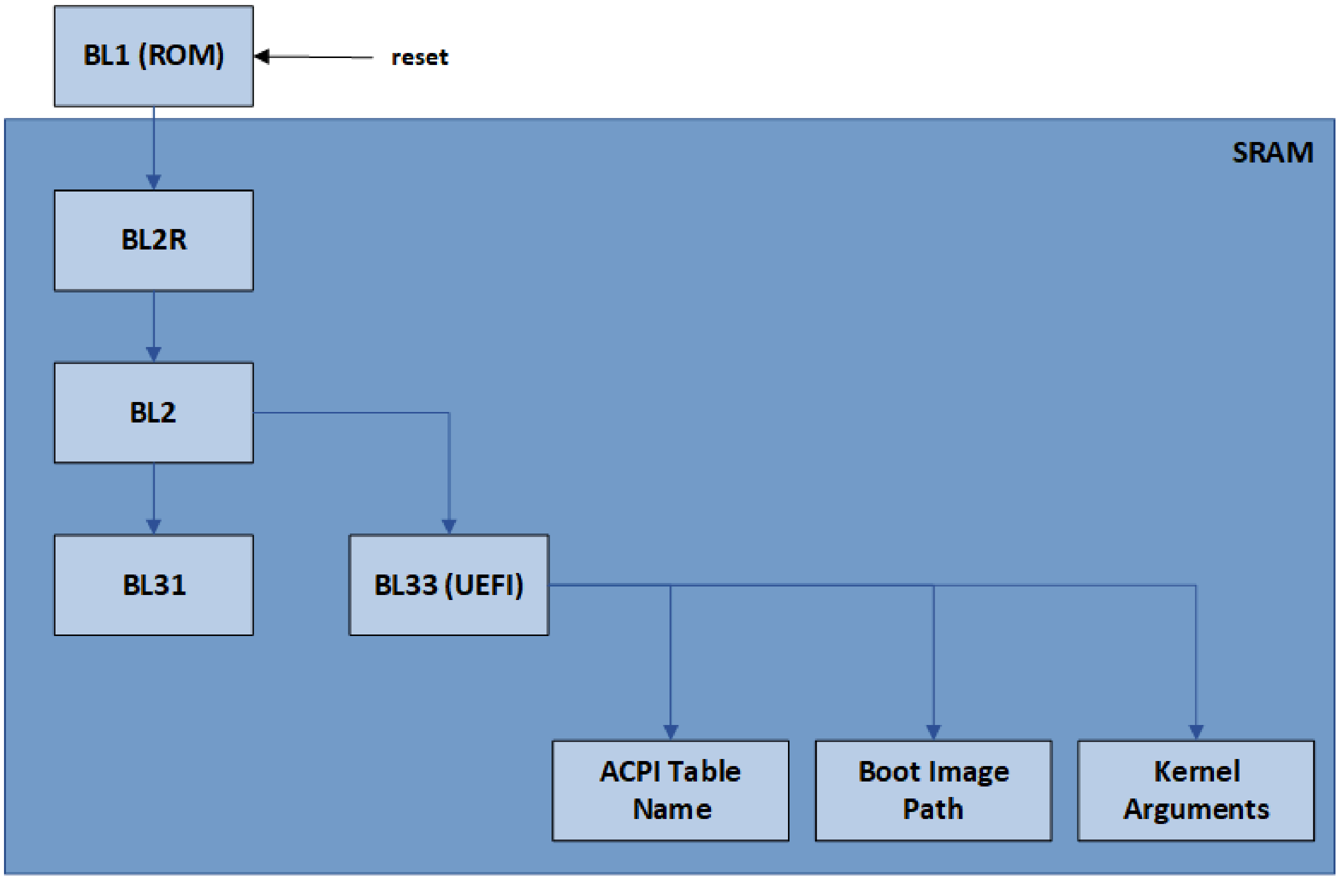
| OS | Action | Instructions |
|---|---|---|
| | NIC firmware, and UEFI/ATF | **Note** This step triggers immediate reboot of the BlueField Arm cores. |
| CentOS/RHEL/Anolis/Rocky | Remove mlxbf-bootimages package | `<bf> $ yum -y remove mlxbf-bootimages*`<br>`<bf> $ yum makecache` |
| | Export the desired distribution | Export DOCA_REPO with the relevant URL. The following is an example for Rocky Linux 8.6:<br>`<bf> $ export DOCA_REPO="https://linux.mellanox.com/public/repo/doca/2.8.0/rhel8.6/dpu-arm64/"`<br>• AnolisOS 8.6 – https://linux.mellanox.com/public/repo/doca/2.8.0/anolis8.6/dpu-arm64/<br>• OpenEuler 20.03 sp1 – https://linux.mellanox.com/public/repo/doca/2.8.0/openeuler20.03sp1/dpu-arm64/<br>• CentOS 7.6 with 4.19 kernel – https://linux.mellanox.com/public/repo/doca/2.8.0/rhel7.6-4.19/dpu-arm64/<br>• CentOS 7.6 with 5.10 kernel – https://linux.mellanox.com/public/repo/doca/2.8.0/rhel7.6-5.10/dpu-arm64/<br>• CentOS 7.6 with 5.4 kernel – https://linux.mellanox.com/public/repo/doca/2.8.0/rhel7.6/dpu-arm64/<br>• Rocky Linux 8.6 – https://linux.mellanox.com/public/repo/doca/2.8.0/rhel8.6/dpu-arm64/ |

| OS | Action | Instructions |
|---|---|---|
| | Add DOCA online repository | `echo "[doca]`<br>`name=DOCA Online Repo`<br>`baseurl=$DOCA_REPO`<br>`enabled=1`<br>`gpgcheck=0`<br>`priority=10`<br>`cost=10" > /etc/yum.repos.d/doca.repo`<br>A file is created under `/etc/yum.repos.d/doca.repo` . |
| | Update index | `<bf> $ yum makecache` |
| | Upgrade UEFI/ATF firmware | Run:<br>`<bf> $ yum install mlxbf-bootimages-signed mlxbf-bfscripts`<br>Then i nitiate the upgrade for UEFI/ATF firmware:<br>`<bf> $ bfrec` |
| | Upgrade BlueField NIC firmware | The following commands update the firmware package and flash the firmware to the NIC:<br>`<bf> $ yum install mlnx-fw-updater-signed`<br>`<bf> $ sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update` |
| | Remove old metapackages | `<bf> $ yum remove doca*  mlnx-ofed* kernel-mft* -y` |
| | Install new metapackages | **Note**<br>Before installing the metapackages, please remove strongSwan and libreSwan packages |

| OS | Action | Instructions |
|---|---|---|
| | | to avoid any conflicts:<br>`<bf> $ yum remove strongswan-bf strongswan-swanctl`<br>`<bf> $ yum remove strongswan-bf strongswan-swanctl libreswan`<br><br>`<bf> $ yum -y install doca-runtime doca-devel` |
| | Upgrade system | `<bf> $ yum upgrade --nobest` |
| | Apply the new changes, NIC firmware, and UEFI/ATF | For the upgrade to take effect, perform BlueField system reboot.<br><br>**Note**<br>This step triggers immediate reboot of the BlueField Arm cores. |

## Upgrading Boot Software

This section describes how to use the NVIDIA® BlueField® networking platform's (DPU or SuperNIC) 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.



## 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 BlueField 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. 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.

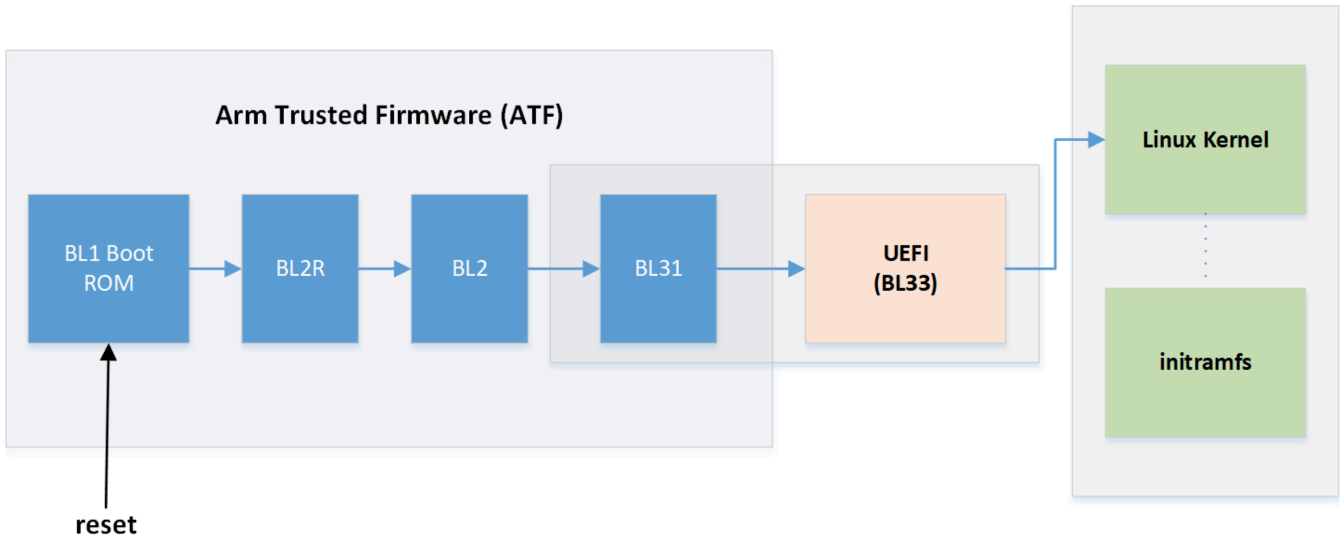
| Filename | Description | ID | Read By |
|---|---|---|---|
| Bl2r-cert | Secure Firmware BL2R (RIoT Core) certificate | 33 | BL1 |
| Bl2r | Secure Firmware BL2R (RIoT Core) | 28 | BL1 |
| bl2-cert | Trusted Boot Firmware BL2 certificate | 6 | BL1/BL2R[a] |
| bl2 | Trusted Boot Firmware BL2 | 1 | BL1/BL2R[a] |
| 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 |

ⓘ **Note**

[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.

## 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.

## 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.

## 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
```

**Note**

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
```

## 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

  ⚠ **Warning**

  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-bootctl 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 driver 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.

## 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.

**Note**

BlueField 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 BlueField to complete the upgrade.

**Note**

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.

## 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.

## 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.

## 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.

## 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)

## 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
```

**Note**

Boot arguments are printed in Hex mode, but you may recognize the boot parameters printed on the side in ASCII format.

## 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

**Note**

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.