



Software Installation and Upgrade

Table of contents

Types and Methods of Updating BlueField Software Image	4
Deploying BlueField Software from Host	12
Deploying BlueField Software from BlueField BMC	51
Deploying BlueField Software Using PXE	52
Deploying BlueField Software Using BFB with PXE	59
Deploying BlueField Software Using ISO with PXE	63
Customizing BlueField Software Deployment	81
Deploying NVIDIA Converged Accelerator	100
Installing Repo Package on Host Side	106
Installing Popular Linux Distributions on BlueField	107
Updating BlueField Software Packages Using Standard Linux Tools	110
BFB FW-Bundle Extraction Process	111

Note

Make sure to update DOCA on the host side before installing the BF-bundle on BlueField.

Info

NVIDIA recommends upgrading to the latest BlueField software and firmware versions.

NVIDIA BlueField comes pre-installed with BlueField software based on Ubuntu 22.04. The DPU's Arm execution environment can be 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 (OVSDDB) 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 deployments and upgrades:

- Full OS image deployment using a BlueField boot stream file (BFB) via RShim interface (also 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 (`apt` , `dpkg` , etc.)

The DPU's BMC software (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 (`BMC_USER` and `BMC_PASSWORD`) are provided in [bf.cfg](#).

i Info

The minimum BMC firmware version required to support BMC upgrades from a BFB image is 23.07. If your BMC firmware version is lower, follow the [NVIDIA BlueField BMC Software](#) documentation to upgrade the BMC firmware. You can find instructions for checking the BMC version in the [software versioning guide](#).

i Info

By default, `BMC_REBOOT="no"`. This means the BMC will need to be manually rebooted after the BFB installation process to apply the new BMC firmware version. Rebooting the BMC will reset the BMC console.

Similarly, `BMC_UPGRADE_RESET="no"` by default. If you change this setting to "yes," the UEFI will automatically reset the CEC and BMC during the first boot after BFB installation, ensuring that the new CEC and BMC firmware versions are applied. Rebooting the BMC will also reset the BMC console.

i Info

Upgrading BlueField software using the BFB bundle automatically performs a NIC firmware update. To apply the new firmware, a host system-level reset or power cycle is required.

A firmware-only BFB, named `bf-fwbundle-<version>.prod.bfb`, is available for BlueField devices for day 2 operations. This BFB updates all firmware components except for the Arm OS and DOCA software.

The firmware-only BFB can use the same set of `bf.cfg` parameters as a standard BFB, with the exception of BlueField-OS related flags (for example, `UPDATE_DPU_OS`).

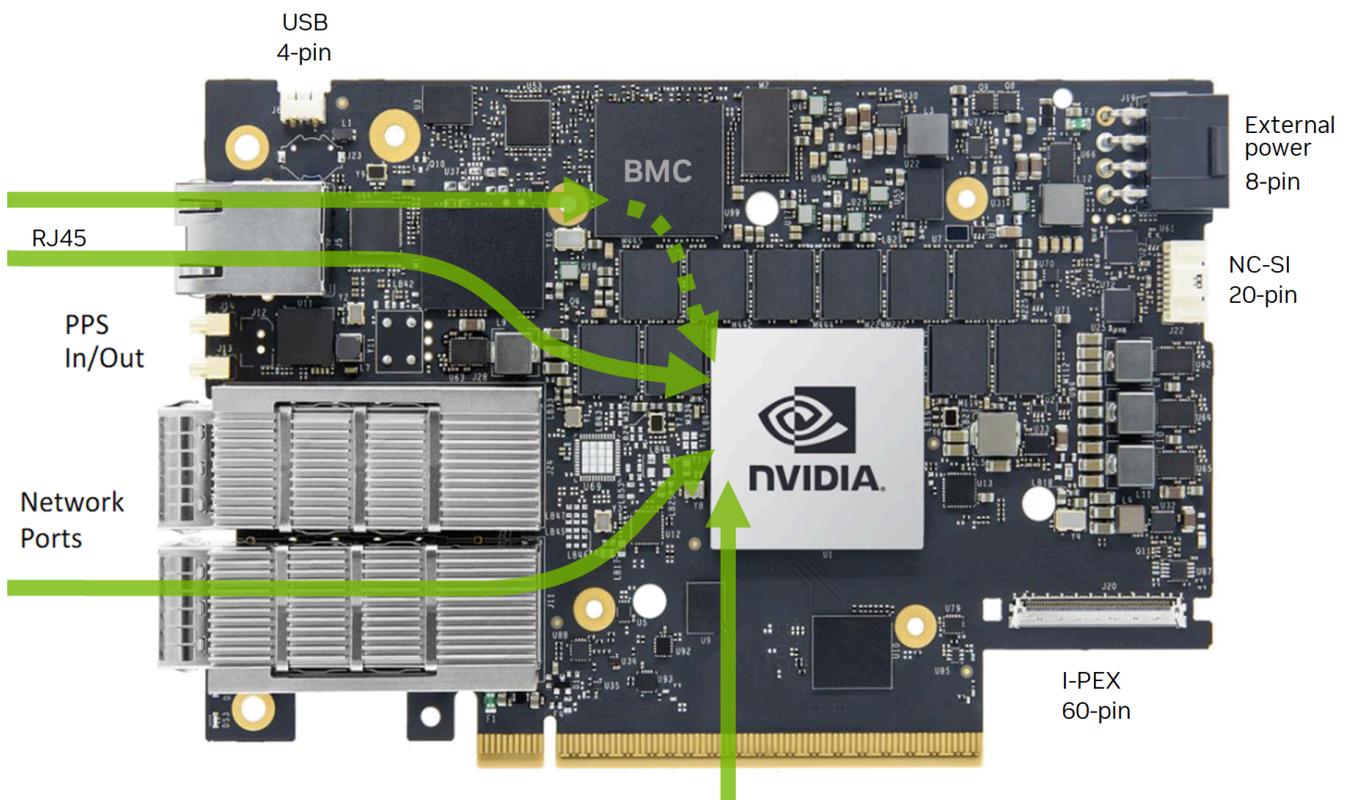
Types and Methods of Updating BlueField Software Image

Interfaces and Methods

NVIDIA® BlueField® has multiple paths to update the running software depending on the BlueField interface used:

- RJ-45 1GbE interface:
 - BlueField BMC using Redfish – Recommended for managing BlueField devices at scale in a data center. See the ["BlueField Provisioning Playbook"](#) and the ["Update and Recovery"](#) pages of the *NVIDIA BlueField BMC Software* documentation for detailed instructions.
 - BlueField Arm (DPU mode only) –
 - PXE/HTTP boot – BlueField, like a standard server, incorporates a UEFI BIOS that can be used to PXE/HTTP boot the device and pull a new image. See ["Deploying BlueField Software Using PXE"](#) for more information.
 - Linux standard tools – done by running `apt update`/`yum install` from within the Arm OS running on BlueField. See section "Upgrading BlueField Using Standard Linux Tools" under ["BF-Bundle Installation and Upgrade"](#) for more information.
- High-speed network ports:
 - BlueField Arm (DPU mode only) – This option is the same as the BlueField Arm scenario above, but it uses the high-speed network ports instead of the RJ-45 1GbE interface.
 - PXE/HTTP boot – BlueField's UEFI BIOS may be used to PXE/HTTP boot the device and pull a new image. See ["Deploying BlueField Software Using PXE"](#) for more information.

- Linux standard tools – done by running `apt update`/`yum install` from within the Arm OS running on BlueField. See section "Upgrading BlueField Using Standard Linux Tools" under "[BF-Bundle Installation and Upgrade](#)" for more information.
- Host server's PCIe interface:
 - From server host-OS – Using the BlueField Management PCIe RShim PF interface. See "[Deploying BlueField Software from Host](#)" for more details.
 - From the server's platform-BMC – PLDM firmware update (PLDM over MCTP over PCIe), a standardized protocol that enables out-of-band firmware upgrades for devices via the server's platform BMC (does not include the DPU OS).



BlueField Image Types

BlueField software images are available in the following formats:

- BFB – BlueField Bundle images – a proprietary format used to update and recover the BlueField device. There are two types of BFB images available:

- BF-Bundle – Includes BlueField firmware components, Arm OS, and DOCA. This bundle contains everything needed to update all possible components of a BlueField device.
- BF-FW-Bundle – Includes only the BlueField firmware components and excludes Arm OS and DOCA. This is typically used for day-2 operations or by customers working in NIC mode who do not require Arm OS or DOCA running on the device. Also comes in the form of a per-SKU binary.
- ISO – Similar to BF-Bundle, this format includes BlueField firmware components, Arm OS, and DOCA packages in an ISO standard format. The BlueField ISO image is based on the standard Ubuntu ISO image for Arm64, but with an updated kernel and added DOCA packages. PXE booting the BlueField device with the ISO image results in the installation of the Arm OS, including DOCA, and the update of BlueField firmware components.
- Repository – An online repository used for updating with standard Linux tools. This method updates DOCA from NVIDIA's repository and Arm OS packages from Canonical's updates repository. BlueField firmware components are also updated. Post-installation steps are required to upgrade BlueField firmware components (i.e., ATF/UEFI, NIC firmware, and BMC components). See "[Updating DPU Software Packages Using Standard Linux Tools](#)" for details.

i Note

Make sure that the BlueField firmware versions are aligned to the same release with the running DOCA version on the BlueField Arm.

The following table summarizes image types, their contents, and from which interfaces they can be loaded:

Interface	Method	Image Type	BlueField Operation Modes which Support this Flow	Includes ATF, UEFI, BMC, CEC, NIC-FW	Includes Arm-OS	Includes DOCA	Normal Service Operation During Update Flow	No
RJ-45 1GbE	BlueField BMC	BF-Bundle	<ul style="list-style-type: none"> • NIC mode 	Yes	Yes	Yes	No	Se cyc

Interface	Method	Image Type	BlueField Operation Modes which Support this Flow	Includes ATF, UEFI, BMC, CEC, NIC-FW	Includes Arm-OS	Includes DOCA	Normal Service Operation During Update Flow	No
	using Redfish	- bfb	<ul style="list-style-type: none"> DPU mode 					recap
		Per-SKU BF-FW-Bundle	<ul style="list-style-type: none"> NIC mode DPU mode 	Yes	No	No	Yes	No ser du up ne: of Ap ser rek cyc
		BF-FW-Bundle - bfb	<ul style="list-style-type: none"> NIC mode DPU mode 	Yes	No	No	No	Se cyc rec ap coi
	BlueField Arm - PXE/HTTP boot	BF-Bundle - bfb	DPU mode	Yes	Yes	Yes	No	Us coi ori file bo Se "De Blu So Us wit de Se cyc rec ap coi

Interface	Method	Image Type	BlueField Operation Modes which Support this Flow	Includes ATF, UEFI, BMC, CEC, NIC-FW	Includes Arm-OS	Includes DOCA	Normal Service Operation During Update Flow	No
		BF-FW-Bundle - bfb	DPU mode	Yes	No	No	No	Us col ori file bo Se "Dr Blu So Us wit de Se cyc rec ap col
		ISO	DPU mode	Yes	Yes	Yes	No	Se cyc rec ap col
	Linux Standard tools	apt/yum update	DPU mode	Yes	No	Yes	No	Lin exi Se cyc rec ap col
High speed network ports	BlueField Arm - PXE/HTTP boot	BF-Bundle - bfb	DPU mode	Yes	Yes	Yes	No	Us col ori file bo Se "Dr

Interface	Method	Image Type	BlueField Operation Modes which Support this Flow	Includes ATF, UEFI, BMC, CEC, NIC-FW	Includes Arm-OS	Includes DOCA	Normal Service Operation During Update Flow	No
								Bl So Us wit de Se cyc rec ap co
		BF-FW-Bundle – bfb	DPU mode	Yes	No	No	No	Us co ori file bo Se "D Bl So Us wit de Se cyc rec ap co
		ISO	DPU mode	Yes	Yes	Yes	No	Se cyc rec ap co
	Linux Standard tools	apt/yum update	DPU mode	Yes	No	Yes	No	Lin exi Se cyc

Interface	Method	Image Type	BlueField Operation Modes which Support this Flow	Includes ATF, UEFI, BMC, CEC, NIC-FW	Includes Arm-OS	Includes DOCA	Normal Service Operation During Update Flow	No
								rec ap coi
Host server's PCIe	From server host-OS	BF-Bundle – bfb	<ul style="list-style-type: none"> NIC mode DPU mode 	Yes	Yes	Yes	No	Se cyc rec ap coi
		BF-FW-Bundle – bfb	<ul style="list-style-type: none"> NIC mode DPU mode 	Yes	No	No	No	Se cyc rec ap coi
		Per-SKU BF-FW-Bundle	<ul style="list-style-type: none"> NIC mode DPU mode 	Yes	No	No	Yes	No ser du up ne. of Ap ser rek cyc

1. BlueField must be restarted to apply changes. When in DPU mode (Arm OS is used), the running Arm OS must be gracefully shutdown prior to power cycle. Instead of a power cycle, server reboot may be performed provided a graceful shutdown of the Arm OS and a manual issue of reset/restart of BlueField BMC using Redfish are performed. [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#)

2. Apply with server reboot if DPU Arm cores are shut down or `INT_CPU_AUTO_SHUTDOWN` is enabled. [1](#) [2](#)

Where to Download BlueField Software Images

Go to [NVIDIA DOCA Download](#) page and download the appropriate image.

Deploying BlueField Software 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](#).

Stage	Procedure	Flow
Preparation ¹	Preparation – install RShim	1. Install DOCA on the host or Install RShim on the host . 2. Verify that RShim is running on the host .
Update options ²	Offline update – install BFB to BlueField ³	3. Install the BFB image (full or firmware BFB). 4. Verify installation completed successfully .
	Deferred update – no-service-interruption update flow ⁴	3. Install the per-SKU BF-FW-Bundle in a deferred flow . 4. Verify that installation completed successfully . 5. Deferred update the DOCA components . 6. Apply the new version .

1. Preparation steps 1 and 2 are common to both offline and deferred update flows. [\[1\]](#)
2. The offline and deferred update flows are mutually exclusive methods. Perform the flow that matches your use case. [\[2\]](#)
3. Recommended for Day 1 operations. [\[3\]](#)
4. Recommended for Day 2 operations. [\[4\]](#)

Prepare Host for Update Flow

To update the BlueField, the host server must have the RShim service running. You can install the RShim service using one of the following two methods.

Option 1: Full DOCA SDK Installation

This method is for users who want the complete DOCA software development kit. The full SDK installation includes the `doca-runtime` package (which provides RShim) along with all other DOCA libraries and tools.

Refer to [DOCA-Host Installation and Upgrade](#) in DOCA documentation for instructions.

Option 2: Minimal RShim Installation (via doca-runtime)

This method is for users who *only* need the RShim service (e.g., for firmware updates) and do not need the full DOCA SDK. The RShim service is included in the `doca-runtime` package.

Verify RShim Device

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`).

Install doca-runtime

Follow the steps for your OS to install the `doca-runtime` package.

OS	Procedure
Ubuntu/Debian	<ol style="list-style-type: none"> 1. Download the DOCA host repo from the NVIDIA DOCA Downloads page . 2. Unpack the deb repo. Run: <pre data-bbox="591 1278 1463 1486"> host# sudo dpkg -i doca-host-repo-ubuntu<version>_amd64.deb </pre> 3. Perform apt update. Run: <pre data-bbox="591 1526 1463 1682"> host# sudo apt-get update </pre> 4. Run <code>apt install</code> for DOCA runtime package. <pre data-bbox="591 1732 1463 1887"> host# sudo apt install doca-runtime </pre>

OS	Procedure
CentOS/RHEL 7.x	<ol style="list-style-type: none"><li data-bbox="553 226 1393 300">1. Download the DOCA host repo from the NVIDIA DOCA Downloads page .<li data-bbox="553 306 992 338">2. Unpack the RPM repo. Run:<pre data-bbox="591 344 1463 548">host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm</pre><li data-bbox="553 554 1003 585">3. Enable new yum repos. Run:<pre data-bbox="591 592 1463 743">host# sudo yum makecache</pre><li data-bbox="553 749 1382 781">4. Run <code>yum install</code> to install DOCA runtime package.<pre data-bbox="591 787 1463 949">host# sudo yum install doca-runtime</pre>
CentOS/RHEL 8.x or Rocky 8.6	<ol style="list-style-type: none"><li data-bbox="553 1008 1393 1081">1. Download the DOCA host repo from the NVIDIA DOCA Downloads page .<li data-bbox="553 1087 992 1119">2. Unpack the RPM repo. Run:<pre data-bbox="591 1125 1463 1329">host# sudo rpm -Uvh doca-host-repo-rhel<version>.x86_64.rpm</pre><li data-bbox="553 1335 992 1367">3. Enable new dnf repos. Run:<pre data-bbox="591 1373 1463 1524">host# sudo dnf makecache</pre><li data-bbox="553 1530 1252 1562">4. Run <code>dnf install</code> to install DOCA runtime.<pre data-bbox="591 1568 1463 1730">host# sudo dnf install doca-runtime</pre>

Ensure RShim Running on Host

After installing *either* the full DOCA SDK (Option 1) or the minimal `doca-runtime` package (Option 2), you must verify that the RShim service is active.

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
```

- `<N>` denotes RShim device number (0, 1, 2, etc).
- If the text `another backend already attached` is displayed, you would not be able to use RShim on the host.
- If the command displays `inactive` or another error, restart RShim service.
Run:

```
sudo systemctl restart rshim
```

- Verify RShim status again. Run:

```
sudo systemctl status rshim
```

2. Run the following to confirm the RShim device is attached. 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.

Offline Updating BlueField Software Using BFB image

This updates the BlueField from the Host OS, in an offline manner, using the BFB bundle image and interrupting the current running services by the BlueField.

BFB Image Types

The BFB image is available in two formats:

- BF-Bundle – includes BlueField firmware and BlueField Arm OS as well as DOCA
- BF-FW-Bundle – includes BlueField firmware only

Select the appropriate image for you.

Info

Both images end with `*.bfb`.

Downloading the BFB Image

To download the BFB image, BF-Bundle or BF-FW-Bundle go to the [NVIDIA DOCA Downloads](#) page.

BFB Installation

This section describes how to use the `bfb-install` utility to install a BFB image.

Prerequisites and Key Considerations

Before starting the installation, be aware of the following:

- Secure boot: All new BlueField-2 and all BlueField-3 devices are secure boot enabled. All software images (ATF/UEFI, Linux Kernel, etc.) must be signed to boot. All formally published software images are signed.
- NIC Mode host reset: After a successful BFB installation in NIC mode, you must perform a power cycle on the host to apply the changes.

Installation Command

The BFB image is installed using the `bfb-install` utility, which is included in the RShim package.

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

This utility pushes the BFB image and any optional `bf.cfg` file to the BlueField, then checks and prints the installation progress.

Monitoring the Installation

- Live progress: To see a live progress bar during the image transfer, install the `pv` Linux tool.

- RShim log: By default, `bfb-install` clears the RShim log at `/dev/rshim<N>/misc` and saves a copy to `/tmp/bfb-install-rshim[N].log`. To prevent the log from being cleared, use the `--keep-log` argument.

Critical Warning: Do Not Restart During Installation

Warning

BFB image installation must complete before restarting the system or the BlueField! Restarting or interrupting the process may result in anomalous behavior (e.g., the BlueField may not be accessible via SSH). If this happens, re-initiate the update process with `bfb-install` to recover the BlueField.

Optional Configurations (bf.cfg)

You can [customize the installation](#) by providing a `bf.cfg` file.

- Updating BMC firmware: To update the BMC firmware using the BFB, you must provide the current BMC credentials and set `BMC_REBOOT=yes` and `CEC_REBOOT=yes` in the `bf.cfg`. This forces an automatic reset of the DPU-BMC after the installation is complete.
- Skipping NIC firmware update: The BFB installation updates the NIC firmware by default, which triggers a reset. If this reset flow is not supported on your setup, you can skip the NIC update by providing `WITH_NIC_FW_UPDATE=no` in the `bf.cfg` file. If you do update the NIC and `bfb-install` alerts you that the reset failed, you must perform a [BlueField System-level Reset](#).

Refer to "[Customizing BlueField Software Deployment](#)" for more information.

Example Installation Output

The following is an example output of a BFB installation, assuming `pv` is 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 Install Completed Successfully

In DPU mode, 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 log into the system.

After the installation of the Ubuntu 22.04 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-2.9.0-<version>_ubuntu-22.04_prod
```

Check the NIC firmware version from the host and make sure the new version is applied:

```
# flint -d /dev/mst/mt41692_pciconf0 q
Image type:          FS4
FW Version:          32.43.0366
FW Version(Running): 32.43.0318
FW Release Date:    12.10.2024
Product Version:    32.43.0318
Rom Info:           type=UEFI Virtio net version=21.4.13
cpu=AMD64, AARCH64
                   type=UEFI Virtio blk version=22.4.14
cpu=AMD64, AARCH64
                   type=UEFI version=14.36.12
cpu=AMD64, AARCH64
                   type=PXE version=3.7.500 cpu=AMD64
Description:        UID                               GuidNumber
Base GUID:          c470bd0300cbe708           38
Base MAC:           c470bdcbe708           38
Image VSD:          N/A
Device VSD:         N/A
PSID:               MT_0000000001
Security Attributes: secure-fw
```

If the version of the NIC firmware is different from the running firmware version as is the case in this example, then a BlueField system-level reset is required.

Info

To verify the version of the installed BMC components, refer to the BMC documentation:

- [Retrieving Golden Image Version Information Using Redfish](#)
- [Fetching Running BMC Firmware Version](#)

- [Fetching Running CEC Firmware Version](#)

In NIC mode, verify the NIC firmware and BMC components versions using Redfish.

Apply New BFB Image

To apply firmware from the BFB image, BlueField must be restarted.

Option 1: Server Power Cycle

This is the most straightforward method. It ensures that all components in the BFB (NIC, Arm OS, ATF, UEFI, BMC, and CEC) are applied.

1. (DPU Mode only) Perform a graceful shutdown of the BlueField Arm OS.
2. Power cycle the server to complete the restart.

Option 2: Server Reboot and Automated BMC Restart

This method uses a one-time configuration to have the BMC and CEC firmware applied automatically during the server reboot cycle.

1. Configure your `bf.cfg` file with BMC credentials and set the `BMC_REBOOT` and `CEC_REBOOT` flags.

Note

For DPU Mode, you must perform a graceful shutdown of the BlueField Arm OS *before* rebooting. Failure to do so will cause the Arm side to skip the restart, and only the NIC firmware will be applied.

2. (DPU Mode Only) Perform a graceful shutdown of the BlueField Arm OS.

3. (DPU Mode Only) Wait for the shutdown to complete.
4. Reboot the server.

Info

During this reboot, the system uses the `bf.cfg` settings to automatically restart the BMC. All firmware components will be applied.

Option 3: Server Reboot and Manual BMC Restart

This method applies the main firmware (NIC, Arm OS, etc.) during the server reboot, but requires a second, manual step to apply the BMC/CEC firmware.

Note

For DPU Mode, you must perform a graceful shutdown of the BlueField Arm OS before rebooting. Failure to do so causes the Arm side to skip the restart, and for only the NIC firmware to be applied.

1. (DPU Mode only) Perform a graceful shutdown of the BlueField Arm OS.
2. (DPU Mode only) Wait for the shutdown to complete.
3. Reboot the server.

Info

At this point, only the NIC, ATF, UEFI, and BlueField Arm OS firmware are applied.

4. Once the server is back up, log into the BlueField BMC (e.g., via Redfish) and issue a restart.

Info

The BMC and CEC firmware are now applied.

Deferred Update BlueField-3

NVIDIA BlueField-3 supports a Deferred Update Flow, which enables administrators to update firmware and DOCA components without immediate service interruption. This capability allows a DPU or SuperNIC to continue servicing workloads while a new firmware bundle and user-space/kernel DOCA components are staged in the background.

The new versions become active only after a reset is applied, minimizing downtime in production environments.

Prerequisites

1. Download the appropriate SKU-specific `fw-bundle-*.bfb` for your DPU from the [DOCA Downloads](#) page.
2. The currently installed firmware must be at least BSP 4.13.0/DOCA 3.2.0 or later.
3. When operating in DPU mode, credentials for DPU-BMC must be specified in `/etc/bf-upgrade.conf` on the Arm OS following the same format as `bf.cfg`. For more details, refer to "[Customizing BlueField Software Deployment](#)".
4. Server booting in UEFI mode. The Deferred Upgrade relays on support from Server UEFI. Please ensure that Device Option ROM in the server UEFI setup, is enabled for the DPU PCIe.

5. Make sure the following DPU UEFI ROM enablers are set to True (Enabled state):

- On the host OS (assuming MFT is installed on host) for servers of ARM architecture:

```
mlxconfig -d /dev/mst/<device> -y set  
EXP_ROM_UEFI_ARM_ENABLE=1
```

- On the host OS (assuming MFT is installed on host), for servers of x86_64/Amd64 architecture:

```
mlxconfig -d /dev/mst/<device> -y set  
EXP_ROM_UEFI_x86_ENABLE=1
```

- On the BlueField Arm OS :

```
mlxconfig -d /dev/mst/<device> -y set  
EXP_ROM_UEFI_ARM_ENABLE=1
```

i Info

mlxconfig is provided by MFT installation. MFT is part of DOCA installation.

6. (Optional) To enable a coordinated BlueField reboot with the host reboot on servers with UEFI boot mode, perform the following configuration from the BlueField Arm OS :

```
mlxconfig -d /dev/mst/<device> set INT_CPU_AUTO_SHUTDOWN=1
```

Note

This must be configured in advance as it requires a [BlueField System-level Reset](#) to take effect.

7. Make sure that RShim is running on the host.

Step 1: Deferred Firmware Update

Firmware updates are delivered using the `bfb-install` tool. Each BlueField SKU requires a specific firmware bundle bfb image per-OPN available on the [NVIDIA DOCA Downloads](#) page.

The BlueField-3 firmware image includes:

- NIC firmware
- ATF/UEFI
- BMC firmware
- CEC firmware

The installation will make use of `bfb-install` utility:

```
# bfb-install --help
Usage: ./bfb-install [options]
Options:
...
```

```
-r, --rshim <device>           Rshim device, format [<ip>:  
<port>:]rshim<N>.  
-d, --deferred                 Deferred activation (local rshim  
only: formerly runtime)
```

To install the bfb image in a deferred flow, run on the host side:

```
# bfb-install --deferred -r rshim0 -b <sku-fw-bundle*>.bfb
```

Example:

```
bfb-install --deferred -r rshim0 -b bf-fwbundle-3.2.0-94-900-  
9D3B6-00CV-AA0_25.10_prod.bfb
```

i Note

In DPU mode, updating BMC and CEC images requires providing DPU BMC credentials in `/etc/bf-upgrade.conf` on the Arm OS.

Example:

Verify successful image update. The following is an output example for NIC Mode. DPU Mode has similar output for the update part.

```
Checking if local host has root access...  
Convert bf-fwbundle-3.2.0_25.10-prod.bfb to flat format for  
deferred upgrade  
INFO: Extracting BFB
```

```
INFO: Extracting BFB's initramfs
1969252 blocks
INFO: Extracting initramfs for repackaging
1969252 blocks
Found PSID MT_0000000884 OPN 900-9D3B6-00CV-A_Ax FW version
32.47.0402 in ./opt/mellanox/mlnx-fw-
updater/firmware/mlxfwmanager_sriov_dis_aarch64_41692
Extracting NIC Firmware Binary for PSID MT_0000000884 OPN 900-
9D3B6-00CV-A_Ax...
INFO: Rebuilding BFB in flat format
BFB: /tmp/tmp.RdMnWrJ80M/bfb/bf-fwbundle-
3.2.0/MT_0000000884/flat.bfb
Checking if rshim driver is running locally...
Pushing bfb + cfg
 143MiB 0:00:16 [8.74MiB/s] [          <=>
]
Collecting BlueField booting status. Press Ctrl+C to stop..
INFO[PSC]: PSC BL1 START
INFO[BL2]: start
INFO[BL2]: boot mode (emmc)
INFO[BL2]: VDD_CPU: 851 mV
INFO[BL2]: VDDQ: 1120 mV
INFO[BL2]: DDR POST passed
INFO[BL2]: UEFI loaded
INFO[BL31]: start
INFO[BL31]: lifecycle GA Secured
INFO[BL31]: runtime
INFO[BL31]: MB ping success
INFO[UEFI]: eMMC init
INFO[UEFI]: eMMC probed
INFO[UEFI]: UPVS valid
INFO[UEFI]: PCIe enum start
INFO[UEFI]: PCIe enum end
INFO[UEFI]: UEFI Secure Boot (enabled)
INFO[UEFI]: Redfish enabled
INFO[UEFI]: DPU-BMC RF credentials found
INFO[UEFI]: exit Boot Service
INFO[MISC]: Linux up
INFO[MISC]: DPU is ready
INFO[MISC]: Extracting BFB /tmp/bfb.MLm5D5/upgrade.bfb
```

```
INFO[MISC]: Found bf.cfg
INFO[MISC]: Detected Flat fwbundle BFB
INFO[MISC]: Staging BMC firmware
INFO[MISC]: Updating CEC firmware
INFO[MISC]: Updating DPU Golden Image
INFO[MISC]: Updating NIC firmware Golden Image
INFO[MISC]: NIC firmware update done: 32.47.0402. NIC Firmware reset or Host power cycle is required to
activate the new NIC Firmware.
INFO[MISC]: Runtime upgrade finished
```

i Note

More details of the update progress can be seen from the Arm console.

i Note

Do not reset the device after firmware update. Continue to Step 2.

Step 2: Deferred Update of DOCA Components

i Note

This step is only relevant when BlueField is operating in DPU mode.

For DEB-based Systems

1. Export the desired repository:

```
export DOCA_REPO="<URL>"
```

Info

- GA:

```
https://linux.mellanox.com/public/repo/doca/latest/ubuntu22.04/dpu-arm64
```

- Latest 2.9 LTS:

```
https://linux.mellanox.com/public/repo/doca/latest-2.9-LTS/ubuntu22.04/dpu-arm64
```

- Latest 2.5 LTS:

```
https://linux.mellanox.com/public/repo/doca/latest-2.5-LTS/ubuntu22.04/dpu-arm64
```

2. Add GPG key:

```
curl $DOCA_REPO/GPG-KEY-Mellanox.pub | gpg --dearmor > /etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub
```

3. Add DOCA repo:

```
echo "deb [signed-by=/etc/apt/trusted.gpg.d/GPG-KEY-Mellanox.pub] $DOCA_REPO ./" > /etc/apt/sources.list.d/doca.list
```

4. Update the APT repository indexes:

```
apt update
```

5. Upgrade system packages:

```
apt upgrade
```

For RPM-based Systems

1. Export the desired repository:

```
export DOCA_REPO="URL"
```

Info

- GA:

```
https://linux.mellanox.com/public/repo/doca/latest/openarm64/
```

- Latest 2.9 LTS:

```
https://linux.mellanox.com/public/repo/doca/latest-2.9-LTS/openeuler22.03sp3/dpu-arm64/
```

- Latest 2.5 LTS:

```
https://linux.mellanox.com/public/repo/doca/latest-2.5-LTS/anolis8.6/dpu-arm64/
```

2. Create repo file `/etc/yum.repos.d/doca.repo`:

```
echo "[doCA]
name=DOCA Online Repo
baseurl=$DOCA_REPO
enabled=1
gpgcheck=0
priority=10
cost=10" > /etc/yum.repos.d/doCA.repo
```

3. Update package index:

```
yum makecache
```

4. Upgrade system:

1. Unlock kernel:

```
dnf install -y 'dnf-command(versionlock)'
dnf versionlock delete kernel*
```

2. Upgrade all packages:

```
dnf upgrade --nobest
```

3. Pin kernel:

```
dnf versionlock kernel*
```

4. Update GRUB:

```
sed -i 's/^GRUB_DEFAULT=.*GRUB_DEFAULT=0/' /etc/default/grub
grub2-mkconfig -o /boot/efi/EFI/<OS_NAME>/grub.cfg
```

Step 3: Apply New Version

The following are different options for applying the new version:

Reset type	Mode of Operation	Applying Reset Steps	Notes
Cold Boot (AC/DC Power Cycle)	<ul style="list-style-type: none"> DPU Mode NIC Mode 	<ol style="list-style-type: none"> (DPU Mode only) Gracefully shut down the BlueField Arm OS. Perform a full server power cycle. 	<ul style="list-style-type: none"> The firmware update is applied automatically during power-up. DPU Mode only: The BlueField Arm OS must be manually shut down before the reboot; otherwise, the update will not apply.
Standard Warm Reboot	<ul style="list-style-type: none"> DPU Mode NIC Mode 	<ol style="list-style-type: none"> (DPU Mode only) Gracefully shut down the BlueField Arm OS. Perform a server warm reboot. 	<ul style="list-style-type: none"> Updates firmware and software after reboot. DPU Mode only: The BlueField Arm OS must be manually shut down before the reboot;

Reset type	Mode of Operation	Applying Reset Steps	Notes
			otherwise, the update will not apply.
Coordinated Reset (Server + DPU)	DPU Mode	<p>1. When the administrator has completed all update flows, the DPU must be armed for the coordinated reset. Run the following command from the BlueField Arm OS. This sets a firmware trigger (<code>MFRL[reset_trigger]=0x48</code>) that instructs the DPU and its DPU-BMC to automatically reset in sync with the next host server reboot. This coordinated reset is required to apply the new firmware and software versions.</p> <pre>mlxreg -d /dev/mst/<device> -y --set "reset_trigger=0x48" -- reg_name="MFRL"</pre> <p>2. Perform a server warm reboot.</p>	<ul style="list-style-type: none"> • Relevant to Deferred Update Flow only. • The next warm reboot will: <ul style="list-style-type: none"> ◦ Gracefully shut down BlueField Arm cores ◦ Reset the NIC, Arm Complex, and BMC ◦ Boot from the new firmware image

Optional Steps Following the Installation of BF-Bundle (DPU Mode with BlueField Arm OS Running)

The following steps can be taken after a new version is applied.

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.
```

(i) Note

For now, please ignore tool's instruction to reboot

(i) 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 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>
```

(i) Note

If `tmfifo_net<n>` on the host does not have an IPv6 address, restart the RShim driver:

```
systemctl restart rshim
```

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
```

(i) 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.

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 Boot Time Optimizations

Several optimizations have been applied to the Ubuntu OS image to significantly reduce boot time.

This section details the configuration changes and their potential effects.

Network Service Timeout Reduction

Network services are often a major contributor to slow boot times.

To minimize delays, the timeout for network readiness checks was reduced to 5 seconds.

Configuration files:

```
# 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
```

Note

These reduced timeouts may affect DHCP-based configurations. If a network interface fails to obtain an IP address, increase the timeout values in the first two configuration files.

Grub Configuration

The GRUB bootloader timeout has been shortened to 2 seconds to accelerate boot.

Configuration in `/etc/default/grub`:

```
GRUB_TIMEOUT=2
GRUB_TIMEOUT_STYLE=countdown
```

This displays a 2-second countdown before booting Ubuntu.

Note

With such a short timeout, the standard keys (`Shift` or `Esc`) cannot be used to enter the GRUB menu. Use the F4 key instead to access the menu.

System Services

- Docker service – The `docker.service` is disabled by default in the Ubuntu OS image because it significantly increases boot time.
- Fast reboot with kexec – The `kexec` utility enables faster system reboots by bypassing the hardware initialization phase. The image includes the helper script `/usr/sbin/kexec_reboot` for convenient execution. Example:

```
# kexec_reboot
```

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 ≤ 16 GB, 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
```

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`.

- 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
```

Deploying BlueField Software from BlueField BMC

For information on deploying BlueField software using BFB from BlueField BMC, please refer to the [*NVIDIA BlueField BMC Software User Manual*](#).

Deploying BlueField Software Using PXE

The BlueField boot flow is similar to a standard server, allowing to boot via an external PXE/HTTP server.

Providing a BFB or ISO image via PXE server enables updating the BlueField software.

The update flow for the two software update image types, [BFB](#) and [ISO](#), is slightly different as elaborated on in the following subpages.

BFB Update vs. ISO Update

- PXE booting the BlueField device with a BFB file updates all components included in the BFB image at the end of the automated update process

Note

The BFB image is available in two formats:

- BF-Bundle – includes BlueField firmware, BlueField Arm OS, and DOCA
- BF-FW-Bundle – includes BlueField firmware only

- PXE booting the BlueField device with the ISO image provides the same result as BF-Bundle installation using standard installation method for Ubuntu OS

Note

Make sure that the BlueField's firmware version and the DOCA version running on BlueField Arm cores belong to the same release.

Setting BlueField to PXE Boot

Users may set the BlueField UEFI for PXE boot using any of the following methods:

- Direct access to BlueField UEFI menu (see section "[Setting Next-boot Device in BlueField UEFI Menu](#)")
- Out-of-band, using the Redfish interface over the BlueField BMC management LAN
- Using `efibootmgr` in the BlueField Linux OS

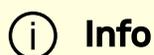
Regardless of the method, the following is a list of PXE devices in the BlueField UEFI which are compatible with the software upgrade procedure described in this chapter:

- `NET-NIC_P0-IPV4`
- `NET-NIC_P0-IPV6`
- `NET-NIC_P1-IPV4`
- `NET-NIC_P1-IPV6`
- `NET-OOB-IPV4`
- `NET-OOB-IPV6`

Setting Next-boot Device in BlueField UEFI Menu

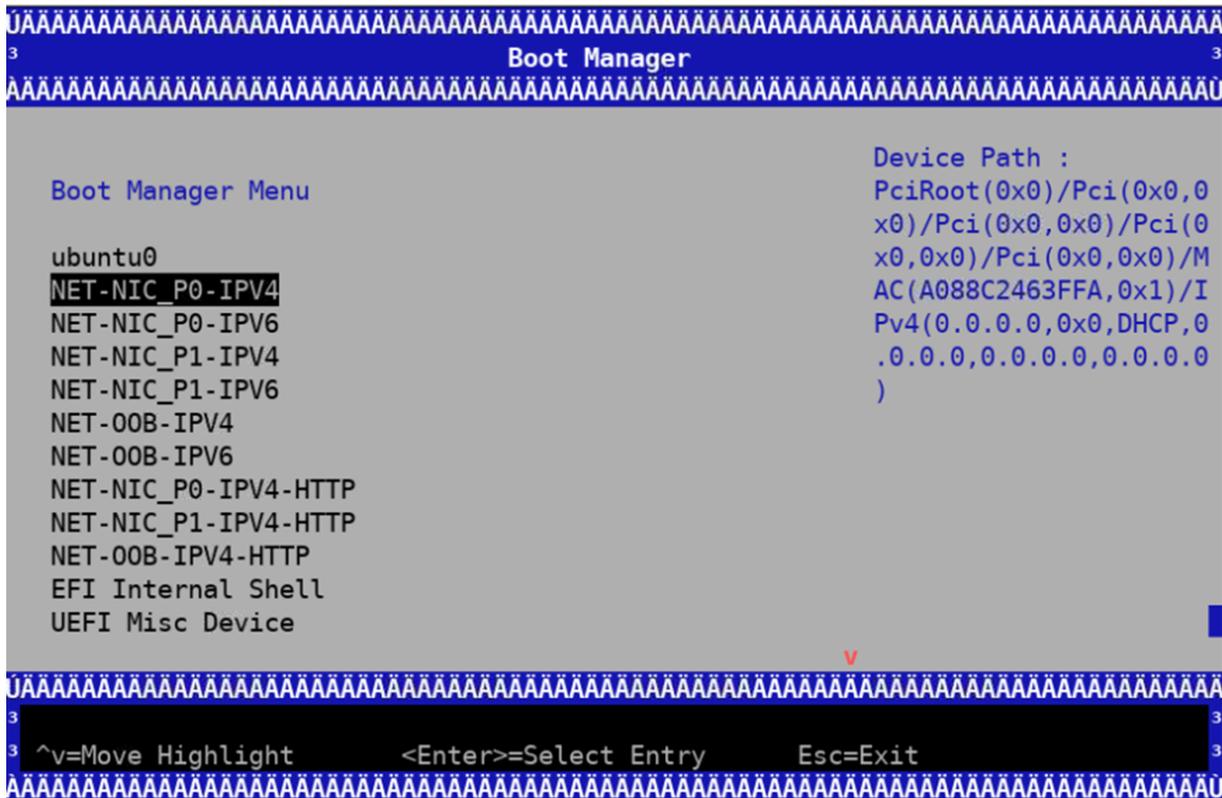
The following steps detail the PXE deployment sequence:

1. Enter into the BlueField UEFI Setup menu.



Refer to section "[Accessing the UEFI Menu](#)" for instructions.

2. Navigate to the Boot Manager menu.
3. Select the relevant PXE devices to boot with, depending on how BlueField is connected to the PXE network.



Setting BlueField to PXE Boot Using Redfish

To set boot option ID using Redfish:

```
curl -k -u '<username>:<password>' -X PATCH
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings
-d '{
  "Boot": {
    "BootSourceOverrideEnabled": "Once",
```

```
"BootSourceOverrideMode": "UEFI",  
"BootSourceOverrideTarget": "Pxe",  
"UefiTargetBootSourceOverride": "Boot000",  
"BootNext": "",  
"AutomaticRetryConfig": "Disabled"  
}  
' | jq
```

i Info

Notice the `UefiTargetBootSourceOverride` and `BootSourceOverrideTarget` fields.

i Info

Refer to the [BlueField BMC User Manual](#) for more information.

i 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](#).

Setting BlueField to PXE Boot Using efibootmgr

To set boot option ID using efibootmgr:


```

OPTION: 94 ( 3) Client NDI 010300 ...
OPTION: 93 ( 2) Client System 000b ..
OPTION: 60 ( 13) Vendor class identifier NVIDIA/BF/PXE
OPTION: 43 (131) Vendor specific info 8005424633000081 ..BF3...
30426c7565466965

0BlueFie
6c643a342e382e30

ld:4.8.0
2d322d6765373965 -2-

ge79e
3037662d64697274 07f-

dirt
7900000000000000

y.....
0000000000000000

.....
008248444f43415f

..HDOCA_
322e352e305f4253

2.5.0_BS
505f342e352e305f

P_4.5.0_
5562756e74755f32

Ubuntu_2
322e30342d312e32 2.04-

1.2
3032333131303800 0231108.
0000000000000000

.....
0000000000000000

.....
0000000000000000

.....
000000 ...

...

```

A DHCP server may include some of the following example information in its `dhcpd.conf` to match against the Bluefield above. Note that matching against the vendor specific information using a regex requires setting the substring start offset so that the search occurs after previous strings have ended:

```
...
# Match against vendor class ID
class "PXEClient" {
    match if substring (option vendor-class-identifier, 0, 13) =
"NVIDIA/BF/PXE";
    filename "/shimaa64.efi";
}

# Match against BF Type
class "BfType" {
    match if substring (option vendor-encapsulated-options, 0, 200)
~= "BF3";
    filename "/shimaa64.efi";
}

# Match against UEFI FW version
class "BfFwVer" {
    match if substring (option vendor-encapsulated-options, 6, 200)
~= "4.8.0-2-ge79e07f";
    filename "/shimaa64.efi";
}

# Match against BF Bundle version
class "BfBundleVer" {
    match if substring (option vendor-encapsulated-options, 55, 200)
~= "2.5.0-BSP_4.5.0_Ubuntu_22.04";
    filename "/shimaa64.efi";
}
...
```

Troubleshooting PXE Boot Issues

More information about how to troubleshoot PXE boot issues can be found in [NVIDIA BlueField Troubleshooting Guide](#).

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. Convert the BFB image file to PXE bootable image(s). Run:

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

For example:

```
# mlx-mkbfcb -x DOCA_2.7.0_BSP_4.7.0_Ubuntu_22.04-  
<version>.bfcb
```

(i) Note

`mlx-mkbfcb` 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-mkbfcb`.

2. Copy the output of the 2 files, `dump-image-v0` and `dump-initramfs-v0`, into the PXE server `tftp` path.

(i) Info

To use a kernel that is compressed with LZMA (as included in BFB as `dump-image-v0`) for PXE installation, it must be uncompressed before use:

```
$ file dump-image-v0
```

If the output says `LZMA compressed data`, proceed to uncompress.

To uncompress the kernel, run:

```
$ text xz -dc < dump-image-v0 > vmlinuz
```

This will uncompress the LZMA kernel image to a usable format called vmlinuz.

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 similar to the standard Ubuntu ISO deployment method. The BlueField ISO image is based on the standard Ubuntu ISO image for Arm64, with an updated kernel and added DOCA packages.

PXE booting the BlueField device with the ISO image results in Arm OS installation (including DOCA packages), NIC firmware and BMC firmware update (if `BMC_PASSWORD` is provided). This is equivalent to using the corresponding version of bf-bundle BFB.

An Ubuntu ISO image can be used if the RShim interface is not available or if there is an existing deployment system in place that can handle a standard Ubuntu ISO image.

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 --get-selections | 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"
    ],
    ....
}

```

i 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

`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$3B0RIrfX$T1Hry93NFUJzg3Nya00rE1
```

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

```
# vim bf.cfg
ubuntu_PASSWORD=' $1$3B0RIrfX$T1Hry93NFUJzg3Nya00rE1 '
```

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

Changing UEFI Password Using bf.cfg

To change the UEFI password, add the current UEFI password under parameter `UEFI_PASSWORD` and define the new UEFI password under `NEW_UEFI_PASSWORD` inside the `bf.cfg` configuration file.

Changing BMC Password Using bf.cfg

To change the BMC root password, add the current BMC root password under parameter `BMC_PASSWORD` and define the new BMC root password under `NEW_BMC_PASSWORD` inside the `bf.cfg` configuration file.

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:

```

#####
# This file contains configuration that can be pushed together to
# customize
# the BFB installation. The configuration is stored as
# /etc/bf.cfg which will
# be used by the 'bfcfg' or other tools as input.
#
# Uncomment the variables to customize the values as needed, Or
# else the
# values will be unchanged, or the default values are used when
# the variables
# are created.
# Note:
# - Try to keep at least one comment line from beginning, if
# possible, when
# pushed together with BFB;
# - Do not use spaces around the '=', it will not work for all
# BFBs.
#
#####

#####
# Manufacturing information (not applicable for BlueField-1).

```

```

# !!! Note: These variables are only allowed to be programmed
once, then the
#           values will be locked. Be sure to set to correct
values when
#           customizing them.
# !!! Note: This configuration may take effect after reset.
#####
# MAC address of the OOB network interface
#MFG_OOB_MAC=00:1a:ca:11:22:33
# OPN number of this board
#MFG_OPN=MBF1M332A
# SKU ID of this board
#MFG_SKU=MBF1M332A
# Model of this board
#MFG_MODL=X
# Serial Number of this board
#MFG_SN=X
# UUID of this board
#MFG_UUID=X
# Revision
#MFG_REV=X

#####
# Manufacturing information extended (applicable for BlueField-3
only).
# !!! Note: These variables are only allowed to be programmed
once, then the
#           values will be locked. Be sure to set to correct
values when
#           customizing them.
# !!! Note: This configuration takes effect after reset.
#####
# System Manufacturer
#MFG_SYS_MFR=X
# System Product Name
#MFG_SYS_PDN=X

```

```

# System Version
#MFG_SYS_VER=X
# Baseboard Manufacturer
#MFG_BSB_MFR=X
# Baseboard Product Name
#MFG_BSB_PDN=X
# Baseboard Version
#MFG_BSB_VER=X
# Baseboard Serial Number
#MFG_BSB_SN=X

#####
# Configuration to set the platform mode (applicable for
BlueField-3 only).
# Only 'DPU' or 'NIC' are permitted values. The configuration is
applied on
# next reboot.
# !!! Note: Once the configuration is applied, a power-cycle is
required
# for this configuration to take effect.
#####
#MFG_PLAT_MODE=DPU

#####
# Control flags for platform configuration
#####

# When set to 'TRUE', the file that encapsules the configuration
# file is saved whithin a persistent storage.
# Note the file will be saved, if and only if, the configuration
# is completed successfully.
#CTL_SAVE_CONFIG_FILE=FALSE

# Reset Lanugage.
# The EFI variables 'Lang*', 'PlatformLang*' are deleted.
#CTL_RESET_LANG=TRUE

```

```

# Reset all SYS_* attributes listed below.
#CTL_RESET_SYS=FALSE

# Delete all BOOT* configurations.
#CTL_DELETE_ALL_BOOT=FALSE

# Reset all misc configurations.
#CTL_RESET_MISC=FALSE

# Delete all UEFI secure boot keys
#CTL_DELETE_ALL_UEFI_SECURE_BOOT_KEYS=TRUE

# Update Secure boot state, either 'Enabled' or 'Disabled'.
# It is relevant when the platform is in user mode, i.e.,
# PK key installed.
#CTL_UEFI_SECURE_BOOT_STATE=DISABLED

# Delete UEFI password settings.
#CTL_DELETE_UEFI_PASSWORD=TRUE

#####
# 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 (only for BF1 BlueWhale board).

```

```
#SYS_ENABLE_2ND_EMMC=FALSE

# Enable eMMC boot partition protection.
#SYS_BOOT_PROTECT=FALSE

# Enable SPCR table in ACPI.
#SYS_ENABLE_SPCR=FALSE

# Select SPCR port when SYS_ENABLE_SPCR=TRUE.
# Either 0 (Port 0) or 1 (Port 1)
#SYS_SPCR_PORT=0

# Disable PCIe in ACPI.
#SYS_DISABLE_PCIE=FALSE

# Enable OP-TEE in ACPI (obsolete).
#SYS_ENABLE_OPTEE=FALSE

# Disable the TM Fifo ACPI
#SYS_DISABLE_TMFF=FALSE

# Disable the 'force_pxe' retry behavior.
# By default, PXE boot will keep retrying all the PXE devices.
# If disabled, it'll try PXE interfaces one round then continue
the normal
# booting sequence.
#SYS_DISABLE_FORCE_PXE_RETRY=FALSE

# Disable BMC Field Mode.
#SYS_ENABLE_BMC_FIELD_MODE=FALSE

# Threshold for the correctable errors to be observed
# before reporting it to the OS.
# Supported values are 0-4294967294
#SYS_CE_THRESHOLD=5000
```

```
# Disable OS error handling via HEST
#SYS_DISABLE_HEST=FALSE

# L3 Cache partition level.
# 0: 00.0%
# 1: 12.5%
# 2: 25.0%
# 3: 37.5%
# 4: 50.0%
# 5: 62.5%
# 6: 75.0%
# 7: 87.5%
#SYS_L3_CACHE_PARTITION_LEVEL=0

# Enable I2C3 in ACPI.
#SYS_ENABLE_I2C3=FALSE

# When set to 'TRUE' UEFI will keep retrying all the bootable
# devices as long as network boot devices are present.
#SYS_ENABLE_FORCE_BOOT_RETRY=FALSE

# Enable OEM MFG configuration.
#SYS_ENABLE_OEM_MFG_CONFIG=FALSE

# Disable I2C1 in ACPI.
#SYS_DISABLE_I2C1=FALSE

# Enable UEFI wait for BMC
#SYS_ENABLE_BMC_WAIT=FALSE

# Disable the auto-refresh of UEFI boot options.
#SYS_DISABLE_AUTO_BOOT_REFRESH=FALSE

# Enable BMC network configuration menu entry.
#SYS_DISPLAY_BMC_NET_CONFIG=FALSE
```

```

# Enable Redfish Feature.
#SYS_ENABLE_REDFISH=TRUE

# Enable RTCSync.
#SYS_RTCSYNC=FALSE

#####
# Configuration to enable UEFI Secure Boot with NVIDIA default
settings on
# The signed EFI Capsule
'/lib/firmware/mellanox/boot/capsule/EnrollKeysCap'
# is used to enroll NVIDIA default certificate files and reset
UEFI password
# to default.
# !!! Note: This configuration takes effect after reset.
#####
#UEFI_SECURE_BOOT=TRUE

#####
# Partition configuration
# Multiple devices can be added here as DISK<M>_NAME.
# Each device can have multiple partitions identified by
"DISK<M>_PART<N>_xxx"
# '<N>' is the partition ID, which represents partition
"DISK<M>_NAME"p<N>.
# For example, assuming DISK0_NAME is /dev/mmcblk0. N=1 means
partition
# /dev/mmcblk0p1, N=8 means partition /dev/mmcblk0p8, etc.
#
# Each device has the following definitions. Optional attributes
are marked
# as (optional) below.
#   DISK<M>_NAME: device path, such as /dev/mmcblk0
#   DISK<M>_PART<N>_SIZE: partition <N> size in MB
#
#                               Value 0 means 'max remaining space',
which is only

```

```

#                 allowed on one partition.
#   DISK<M>_PART<N>_TYPE: partition <N> type, which could be
#                 EFI, Linux or UUID defined in
#
https://en.wikipedia.org/wiki/GUID\_Partition\_Table
#   DISK<M>_PART<N>_MOUNT: mount path
#   DISK<M>_PART<N>_PERSIST: CREATE | KEEP (optional)
#                 CREATE: create or overwrite this
partition
#                 KEEP:   keep this partition, or
create if not exist
#                 Only one partition could be marked
as 'persist'.
#####
#DISK0_NAME=/dev/mmcblk0
#DISK0_PART1_SIZE=150
#DISK0_PART1_TYPE=EFI
#DISK0_PART1_MOUNT=/boot/efi
#DISK0_PART2_SIZE=0
#DISK0_PART2_TYPE=Linux
#DISK0_PART2_MOUNT=/
#DISK0_PART8_PERSIST=CREATE
#DISK0_PART8_SIZE=250
#DISK0_PART8_TYPE=Linux

#####
# Boot Order configuration
# Each entry BOOT<N> could have the following format:
# PXE or HTTP:
#   BOOT<N>=NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>-<IPV4 | IPV6>[-
HTTP]
# PXE over VLAN (vlan-id in decimal):
#   BOOT<N>=NET-<NIC_P0 | NIC_P1 | OOB | RSHIM>[.<vlan-id>]-<IPV4
| IPV6>[-HTTP]
# UEFI Shell:
#   BOOT<N>=UEFI_SHELL

```

```

# DISK: boot entries created during OS installation.
#   BOOT<N>=DISK
#
# Additional BOOT<N>_* parameters may be added to complement the
# Boot Order configuration. These optional field parameters are:
#   BOOT<N>_DESC      : boot description string (single/double
#                       quoted).
#       BOOT2_DESC='ubuntu'
#   BOOT<N>_DEVPATH  : boot device path string (single/double
#                       quoted).
#       Full device path, applicable to any boot option.
#
BOOT0_DEVPATH='PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)'
#   File path, applicable to DISK boot options.
#       BOOT2_DEVPATH='\\EFI\\ubuntu\\shimaa64.efi'
#   BOOT<N>_ARGS     : boot arguments string (single/double
#                       quoted).
#       Boot arguments are for future use.
#####
# In the example below, BOOT0 is use for PXE boot over the 2nd
# ConnectX port.
# BOOT1 is for HTTP boot over the 2nd ConnectX port. If both
# BOOT0 and BOOT1
# fail, it continues to boot from disk with boot entries created
# during OS
# installation.
#BOOT0=NET-NIC_P1-IPV4
#BOOT1=NET-NIC_P1-IPV4-HTTP
#BOOT2=DISK

#####
# Other misc configuration
#####
# MAC address of the rshim network interface.
#NET_RSHIM_MAC=00:1a:ca:ff:ff:01

```

```

# DHCP class identifier for PXE (arbitrary string up to 32
characters)
# If FACTORY_DEFAULT_DHCP_BEHAVIOR is TRUE (or not specified),
default value
# 'NVIDIA/BF/PXE' will be configured during BFB installation. Or
else it'll
# take the value configured in 'PXE_DHCP_CLASS_ID'.
#FACTORY_DEFAULT_DHCP_BEHAVIOR=FALSE
#PXE_DHCP_CLASS_ID=NVIDIA/BF/PXE

# DHCP IPv6 DUID configuration for network boot.
# Permitted values are 'UUID' or 'LLT'.
#NET_DHCP_IPV6_DUID=UUID

# Version of the BF Bundle. The version can be obtained through
# 'bfver' command line.
#BF_BUNDLE_VERSION=bf-bundle-2.7.0-27_24.04_ubuntu-22.04_prod

#####
# BlueField Arm platform firmware update
#####
# 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"

#####
# BlueField Arm OS Update
#####
# UPDATE_DPU_OS - Update/Install BlueField Operating System
(Default: yes)
#UPDATE_DPU_OS="yes"

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

```

```

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

# On some operating systems, the partition UUID may change after
the first
# boot following installation. To mitigate this, enable
FSTAB_USE_DEV_NAME=yes
# to use device names instead of UUIDs. (Default: no)
#FSTAB_USE_DEV_NAME=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.sha512.10000.5EB1FF92FDD89BDAF339
#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)
# Use openssl to set the password for the ubuntu user:
#
# $ openssl passwd -1
# Password:
# Verifying - Password:
# $1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1
#
#ubuntu_PASSWORD='$1$3B0RIrfX$TlHry93NFUJzg3Nya00rE1'

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

```

```

# WITH_NIC_FW_UPDATE=yes

# Force NIC firmware update even if the currently installed
version matches the provided
# version. (Default: no)
#FORCE_NIC_FW_UPDATE=no

# Reset NIC firmware after flashing to apply the new firmware
version. (Default: yes)
#NIC_FW_RESET="yes"

# Perform a level-3 NIC firmware reset if a level-0 reset was
attempted and failed. (Default: no)
#FORCE_NIC_FW_RESET="no"

#####
# 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"

# UPDATE_CERTIFICATES - Update Nvidia certificates (Default: yes)
# PEM certificates uploaded to the BMC only once.
#UPDATE_CERTIFICATES="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

# Clear the RSHIM log buffer on the DPU BMC before starting the
BMC component update
# process to ensure all messages are captured in the log.
(Default: yes)
#RESET_BMC_RSHIM_LOG="yes"

#####
# Hook function to customize the BFB installation
#####
# bfb_modify_os()
# - SHELL function called after file the 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 EMMC partitions format
and OS filesystem is extracted.

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

```

System Configuration Dump

The `bfcfg` script included with the BlueField Arm OS can be used to dump system configuration information modified with `bf.cfg` or through the UEFI menu. The `-d` parameter can be used to enable dump mode and the `-l` parameter used to set the dump level (how much configuration information is logged).

To perform a full system configuration dump, run:

```
# bfcfg -d -l 2
```

Deploying NVIDIA Converged Accelerator

Note

This page is relevant for A30 and A100 converged accelerator cards. For instructions on the NVIDIA® H20 NVL16 device (configured to run in NIC mode only), refer to [Deploying BlueField Software 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.

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 from Host](#)
- [Deploying BlueField Software from BlueField 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. Perform 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. Perform 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=Firmware 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](#).

To install the repository packages on the host side, follow the instructions detailed in the following sections of the [NVIDIA DOCA Installation Guide for Linux](#):

1. [Uninstalling Software from Host](#)
2. [Installing Prerequisites on Host for Target BlueField](#)
3. GPG and Kernel Module Signing (and all relevant subsections)
4. [Installing Software on Host](#)
 1. [DOCA Extra Package and doca-kernel-support](#)

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
<code>bluefield-edac</code>	BlueField-specific EDAC driver	☐	☐

Driver	Description	BlueField-2	BlueField-3
<code>dw_mmc_bluefield</code>	BlueField DW Multimedia Card driver	☐	☐
<code>sdhci-of-dwcmshc</code>	SDHCI platform driver for Synopsys DWC MSHC	☐	☐
<code>gpio-mlxbf2</code>	GPIO driver	☐	☐
<code>gpio-mlxbf3</code>	GPIO driver	☐	☐
<code>i2c-mlx</code>	I2C bus driver (<code>i2c-mlxbf.c</code> upstream)	☐	☐
<code>ipmb-dev-int</code>	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.	☐	☐
<code>ipmb-host</code>	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.	☐	☐
<code>mlxbf-gige</code>	Gigabit Ethernet driver	☐	☐
<code>mlxbf-livfish</code>	BlueField HCA firmware burning driver. This driver supports burning firmware for the embedded HCA in the BlueField SoC.	☐	☐
<code>mlxbf-pka</code>	BlueField PKA kernel module	☐	☐
<code>mlxbf-pmc</code>	Performance monitoring counters. The driver provides access to available performance modules through the <code>sysfs</code> interface. The performance modules in BlueField are present in several hardware blocks and each block has a certain set of supported events.	☐	☐
<code>mlxbf-ptm</code>	Kernel driver that provides a debugfs interface for the system software to monitor the BlueField device's power and thermal management parameters.	☐	☐

Driver	Description	BlueField-2	BlueField-3
<code>mlxbf-tmfifo</code>	TMFIFO driver for BlueField SoC	☐	☐
<code>mlx-bootctl</code>	Boot control driver. This driver provides a <code>sysfs</code> interface for systems management software to manage reset time actions.	☐	☐
<code>mlx-trio</code>	TRIO driver for BlueField SoC	☐	☐
<code>pwr-mlxbf</code>	Supports reset or low-power mode handling for BlueField.	☐	☐
<code>pinctrl-mlxbf</code>	Allows multiplexing individual GPIOs to switch from the default hardware mode to software-controlled mode.	☐	☐
<code>mlxbf-pmc</code>	Mellanox PMC driver	☐	☐

Updating BlueField Software Packages Using Standard Linux Tools

Please refer to section "Upgrading BlueField Using Standard Linux Tools" of the [NVIDIA DOCA Installation Guide for Linux](#) for information.

BFB FW-Bundle Extraction Process

The `bfb-tool` is a command-line utility used to manage BlueField BFB files (i.e., `bf-bundle` or `bf-fwbundle`). It supports two primary actions:

- Extracting payloads from a BFB
- Repacking BFBs for a specific PSID or OPN

This page provides comprehensive usage instructions, supported options, and examples.

Prerequisites

`qemu-aarch64-static` is required to extract the NIC firmware payload from the bundle.

This binary is included in the `qemu-user-static` package.

To install `qemu-user-static` via Docker:

```
$ docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
```

This command registers QEMU interpreters on the host, enabling execution of Arm binaries (such as `aarch64`) on `x86_64` systems using `binfmt_misc`.

See more information on `qemu-user-static` in [GitHub](#).

Command Syntax

```
bfb-tool <action> --bfb <BFB> --psid <PSID>|--opn <OPN>|--all [-p|--profile <Profile>] [-o|--output-dir <dir>] [-f|--output-format <format>] [-B|--output-bfb] [-r|--replace-fw <path>] [-v|--verbose] [-j|--json]
```

Actions

Action	Description
extract	Extracts the firmware payload for the specified PSID, OPN, or all configurations
repack	Extracts and rebuilds a new BFB file tailored to a specific PSID or OPN
info	Print version information for components contained in a BFB

Options

Option	Description
--bfb <BFB>	Path to the input BFB file (e.g., <code>bf-fwbundle-*.bfb</code>)
--psid <PSID>	Target PSID (e.g., <code>MT_0000001070</code>) for extraction or repacking
--opn <OPN>	Target OPN (e.g., <code>900-9D3B6-F2SV-PA0_Ax</code>) for extraction or repacking
--profile <Profile>	Configuration profile
-p, --profile <Profile>	Optional profile to apply during repacking
--output-dir <DIR>	Directory to store the extracted or repacked output files
-v, --verbose	Enable verbose output for detailed execution logs
-j, --json	Print version info in JSON format

Examples

- Extract payload for a specific PSID:

```
bfb-tool extract --bfb bf-fwbundle-2.9.2-40_25.02-prod.bfb --  
psid MT_0000001070
```

- Repack BFB for a specific OPN:

```
bfb-tool repack --bfb bf-fwbundle-2.9.2-40_25.02-prod.bfb --  
opn 900-9D3B6-F2SV-PA0_Ax
```

- Extract all configurations:

```
bfb-tool extract --bfb bf-fwbundle-2.9.2-40_25.02-prod.bfb --  
all
```

- Specify a custom output directory:

```
bfb-tool extract --bfb bf-fwbundle-2.9.2-40_25.02-prod.bfb --  
psid MT_0000001070 --output-dir /path/to/output
```

- Enable verbose output:

```
bfb-tool repack --bfb bf-fwbundle-2.9.2-40_25.02-prod.bfb --  
opn 900-9D3B6-F2SV-PA0_Ax -v
```

Usage Notes

- Ensure the specified BFB file exists and is accessible

- Only one of these options can be used per command: `--psid`, `--opn`, or `--all`
- Use `--profile` to apply a specific firmware configuration profile (if applicable)
- By default, extracted files are saved to `/tmp/<bf-name>/` unless overridden using `--output-dir`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

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

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF

ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

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

Copyright 2025. PDF Generated on 11/20/2025