



## **Update and Recovery**

# Table of contents

System Inventory	3
Deploying Software Using BFB	8
Boot Configuration	32

This section contains the following pages:

- [System Inventory](#)
- [Deploying Software Using BFB](#)
- [Boot Configuration](#)

---

# System Inventory

The Redfish `FirmwareInventory` schema is a component of the Redfish API standard used for providing detailed information about the firmware components, including their types and versions, within a computer system. It allows for easy management and monitoring of firmware-related aspects in a standardized manner.

- Get the firmware inventory collection

## Info

After the BMC boots, it may take a few seconds (6-8 in NVIDIA® BlueField®-2, and 2 in BlueField-3) until everything can be seen in the following list.

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

Example output:

```
{
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory",
  "@odata.type": "#SoftwareInventoryCollection.SoftwareInventoryCollection",
  "Members": [
    {
      "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/BMC_Firmware"
    },
    {
```

```

        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/Bluefield_FW_ERoT"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_ATF"
    },
    {
        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/DPU_ATF_pending"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_BOARD"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_BSP"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_NIC"
    },
    {
        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/DPU_NIC_pending"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_NODE"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_OFED"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_OS"
    },
    {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_SYS_IMAGE"
    },
    {

```

```

        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/DPU_UEFI"
      },
      {
        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/DPU_UEFI_pending"
      },
      {
        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/golden_image_arm"
      },
      {
        "@odata.id" :
"/redfish/v1/UpdateService/FirmwareInventory/golden_image_config"
      },
      {
        "@odata.id" : "/redfish/v1/UpdateService/FirmwareInventory/golden_image_nic"
      }
    ],
    "Members@odata.count" : 17,
    "Name" : "Software Inventory Collection"
  }

```

### Note

- Retrieving DPU object versions is supported when operating in DPU mode only.
- In NIC mode on BF3, the supported versions are:
  - DPU\_ATF
  - DPU\_NIC
  - DPU\_UEFI

- Pending versions are supported only on BF3

- Get a specific component information

### Info

In the following example, the `DPU_OS` inventory components are retrieved.

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

Example output:

```
{
  "@odata.id":
  "/redfish/v1/UpdateService/FirmwareInventory/DPU_OS",
  "@odata.type":
  "#SoftwareInventory.v1_4_0.SoftwareInventory",
  "Description": "Host image",
  "Id": "DPU_OS",
  "Members@odata.count": 1,
  "Name": "Software Inventory",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Systems/Bluefield/Bios"
    }
  ],
  "SoftwareId": "",

```

```
"Status": {
  "Conditions": [],
  "Health": "OK",
  "HealthRollup": "OK",
  "State": "Enabled"
},
"Updateable": true,
"Version": "DOCA_2.2.0_BSP_4.2.1_Ubuntu_22.04-8.23-07"
}
```

# Deploying Software Using BFB

NVIDIA® BlueField® devices support software deployment and upgrade through various BFB image types. For details on available image formats and their contents, refer to the "[Types and Methods of Updating BlueField Software Image](#)" page.

## BFB Installation

BlueField software and firmware can be deployed using one of two methods:

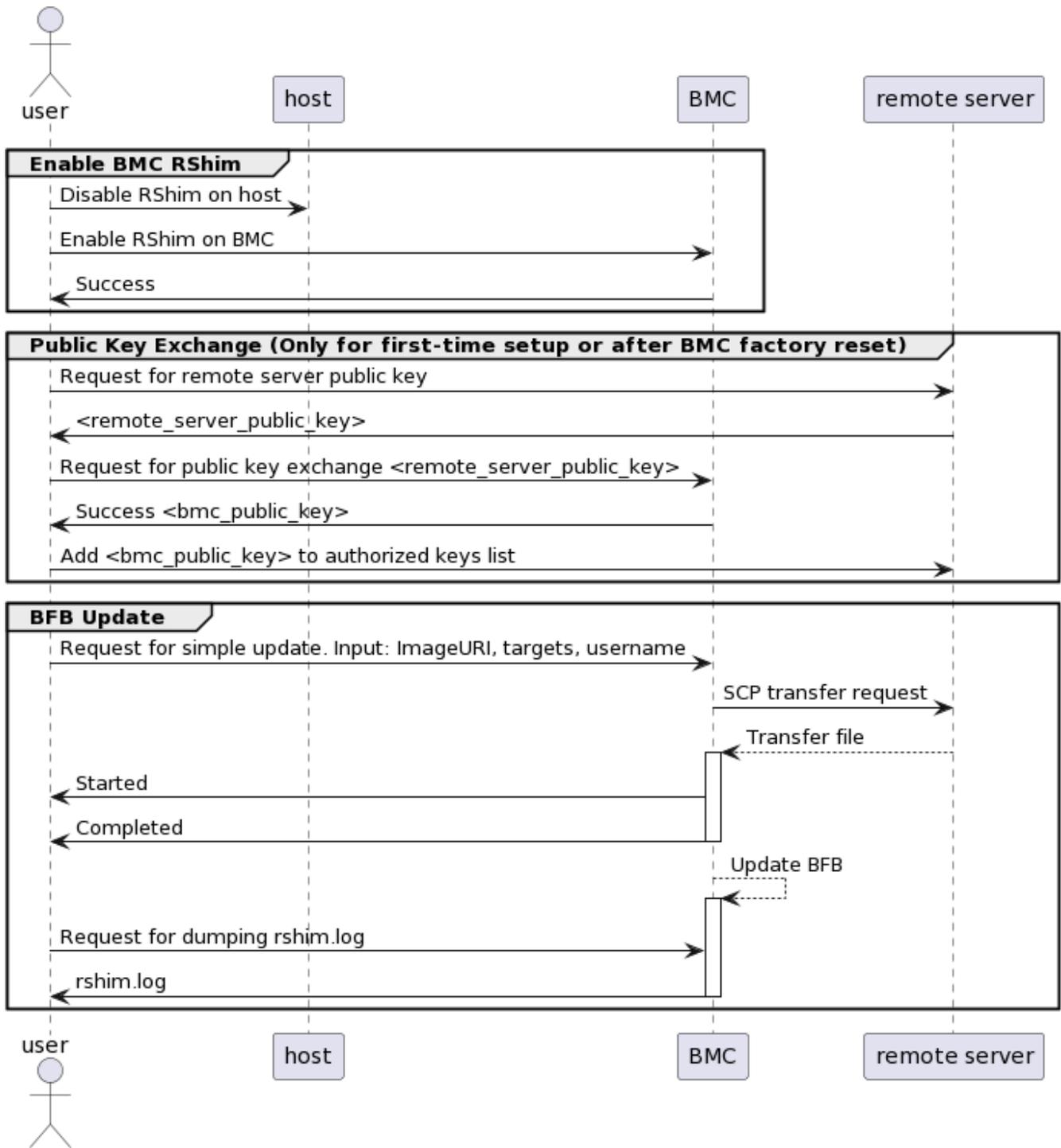
Update Flow	Description	Supported Image Types
Offline Update Flow	Traditional method where the DPU or SuperNIC is taken out of service immediately when the update begins. The device reboots into maintenance mode, applies firmware, system image, and DOCA component updates, and reboots again to activate the new versions. Ensures a clean, immediate transition but involves downtime. This flow supports recovery as well.	<ul style="list-style-type: none"><li>• BF-Bundle BFB (firmware + Arm OS + DOCA)</li><li>• BF-fwBundle BFB (firmware only)</li><li>• Per-SKU BF-fwBundle BFB</li></ul>
Deferred Update Flow	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 an <a href="#">activation command</a> and reset are applied, minimizing downtime in production environments.	<ul style="list-style-type: none"><li>• Per-SKU BF fwBundle BFB in flat format. Use <code>bf-tool</code> with the <code>--output-format flat</code> option to convert to flat format.</li></ul>

## BFB Installation Procedure

The BFB deployment process consists of these main stages:

	<b>Stage</b>	<b>Description</b>
1	Disable RShim (if applicable)	Ensure that the RShim interface is disabled on the host side where the given DPU resides to prevent interference with the BFB update process.
2	Transfer the BFB image	Initiate the image transfer using one of the supported methods: <ul style="list-style-type: none"><li>• <a href="#">Redfish interface</a> via <a href="#">SCP</a>, <a href="#">HTTP</a>, or <a href="#">HTTPS</a><ul style="list-style-type: none"><li>◦ When using SCP for the first time (or after a BMC factory reset), confirm the SSH identity of the BMC when connecting to the server hosting the BFB.</li><li>◦ Send the update request via the Redfish API.</li></ul></li><li>• <a href="#">Direct SCP</a> — Transfer the BFB directly to the BMC file system using secure copy.</li></ul>
3	<a href="#">Monitor installation progress</a>	Track the update process and verify installation status through Redfish logs, BMC console, or CLI output.
4	Apply the new version	Reboot the system to activate the new firmware and software. The specific reboot behavior depends on the selected update flow (offline or deferred).

## Update Flow



## Changing Default Credentials Using bf.cfg

If installing the BF-Bundle BFB with BlueField Arm OS, 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` ).

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$3B0RlrfX$TlHry93NFUJzg3Nya00rE1
```

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

```
# vim bf.cfg  
ubuntu_PASSWORD='$1$3B0RlrfX$TlHry93NFUJzg3Nya00rE1'
```

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

## Update Flow Image Transfer

### Offline Update Flow

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

### **(i) Note**

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

### **(i) Info**

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

## **Deferred Update Flow**

### **(i) Note**

Supported at beta level.

Deferred update flow enables upgrading DOCA components on NVIDIA® BlueField® platforms running in DPU mode while keeping services operational throughout the process. The update is applied only after a coordinated reset, minimizing downtime.

### **Deferred Update Flow Prerequisites**

1. Download the per-SKU `fw-bundle` BFB from [DOCA Downloads](#) page.

**Note**

The installed firmware must be BSP 4.13.0 (DOCA 3.2.0) or later.

2. Repackage the `bf-fwbundle` in flat format using the `bfb-tool`:

```
bfb-tool repack --bfb bf-fwbundle-<version>.bfb --psid <PSID>
--output-format flat
```

Expected output:

```
BFB: .../bf-fwbundle-*.flat.bfb
```

3. (Optional) If a `bf.cfg` file is required, bundle it into the flat BFB using `mlx-mkbf`.

```
mlx-mkbf --boot-args bf.cfg <input_flat.bfb>
<output_cfg_flat.bfb>
```

4. If operating in DPU mode, add the DPU-BMC credentials to `/etc/bf-upgrade.conf` on the Arm OS using the standard `bf.cfg` format. For more details, refer to "[Customizing BlueField Software Deployment](#)".
5. (Optional) To coordinate the BlueField reboot with the host reboot, run the following on 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.

## Initiate Firmware Deferred Update Flow Transfer

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

## Note

The parameter `Stage` is only supported when `Targets` is set to `redfish/v1/UpdateService/ FirmwareInventory/DPU_OS`. Another deferred update will fail if the staging has not completed.

Example success message if the request is valid and a task is created:

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

## Transfer Command Parameters

- `image_uri` – specifies the complete location of the BFB file on the remote server. It is constructed by joining the Remote Server IP and the Absolute File Path with a single forward slash (i.e., `<Remote_IP>/<Absolute_Path>`).

### **Note**

Because absolute paths start with a slash (e.g., `/tmp/...`), the final string will typically contain a double slash (`//`) between the IP and the directory. For example, if the IP is `10.10.10.10` and the file path is `/tmp/image.bfb`, the resulting URI is:  
`"ImageURI" : "10.10.10.10//tmp/image.bfb"`.

- `TransferProtocol` – set to either `SCP`, `HTTP`, `HTTPS`

### **Note**

If using HTTPS, make sure the BMC has a certificate to authenticate the HTTPS server. Or install a valid certificate to authenticate:

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

- `username` – username on the remote server (only required for SCP)
- `bmc_ip` – BMC IP address

- `Stage` – a value of `True` indicates a deferred flow, a value of `False` or omitting this parameter indicates an offline update flow

## Setting Up Secure Connection

### **Note**

Relevant only for SCP users with Redfish.

### **Note**

The following is an example for how to generate the server public key on Ubuntu 22.04 and it may be different on other OS distributions/versions.

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

### **Info**

OpenSSH is required for this step.

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

Where:

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

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

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

Where:

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

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

```
{
  "@Message.ExtendedInfo": [
    {
      "@odata.type": "#Message.v1_1_1.Message",
```

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

## Tracking Image Transfer Status and Progress

After receiving a success message of a valid `SimpleUpdate` request and a `running` task state. Run the following Redfish command to track image transfer status and progress:

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

### Note

During the transfer, the `PercentComplete` value remains at 0 for offline update flow. If no errors occur, the `TaskState` is set to `Running`, and a keep-alive message is generated every 5 minutes. Once the transfer is completed, the `PercentComplete` is set to `100`, and the `TaskState` is updated to `Completed`. Upon failure, a message is generated with the relevant resolution.

Example:

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

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

## Installation Status and Activation

### Tracking Offline Update Flow Installation Status

In the Offline Update Flow, once the image transfer finishes, users can use the RShim miscellaneous messages log dump to track the installation's progress and status.

1. Initiate request for dump download:

```
sudo curl -k -u root:'<password>' -d '{"DiagnosticDataType":  
"Manager"}' -X POST  
https://<ip_address>/redfish/v1/Managers/Bluefield_BMC/LogServ
```

Where:

- `<ip-address>` – BMC IP address
- `<password>` – BMC password

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

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

Where:

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

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

```
sudo curl -k -u root:'<password>' -H 'Content-Type:  
application/json' -X GET
```

```
https://<ip_address>/redfish/v1/Managers/Bluefield_BMC/LogServ
--output </path/to/tar/log_dump.tar.xz>
```

Where:

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

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

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

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

### **Note**

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

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

```
curl -k -u root:"<PASSWORD>" -H "Content-Type:
application/json" -X GET
```

```
https://<bmc_ip>/redfish/v1/UpdateService/FirmwareInventory/DPU_0S"
```

Example response:

```
"@odata.id":  
"/redfish/v1/UpdateService/FirmwareInventory/DPU_0S",  
"@odata.type":  
"#SoftwareInventory.v1_4_0.SoftwareInventory",  
"Description": "Host image",  
"Id": "DPU_0S",  
"Members@odata.count": 1,  
"Name": "Software Inventory",  
"RelatedItem": [  
  {  
    "@odata.id": "/redfish/v1/Systems/Bluefield/Bios"  
  }  
],  
"SoftwareId": "",  
"Status": {  
  "Conditions": [],  
  "Health": "OK",  
  "HealthRollup": "OK",  
  "State": "Enabled"  
},  
"Updateable": true,  
"Version": "DOCA_2.2.0_BSP_4.2.1_Ubuntu_22.04-8.23-07"
```

## Deferred Update Flow

### Checking Staging Status

Check the staging status after the transfer (i.e., the `SimpleUpdate` task) is completed successfully. A successful result of the staging procedure will be

`com.nvidia.BF.Rshim.Status.Completed` after staging completes.

```
curl -k -u root:<password> -H "Content-Type: application/json" -X GET
https://<bmc_ip>/redfish/v1/Managers/Bluefield_BMC/Actions/Oem/NvidiaManager.GetUpdateStatus
{
  "UpdateStatus": "com.nvidia.BF.Rshim.Status.Completed"
}
```

### **(i) Note**

The `UpdateStatus` can be:

- `'com.nvidia.BF.Rshim.Status.Invalid'`  
0000019b-22f7-dcdf-a9fb-b6f7c5430003
- `'com.nvidia.BF.Rshim.Status.Idle'`
- `'com.nvidia.BF.Rshim.Status.InProgress'`
- `'com.nvidia.BF.Rshim.Status.Completed'`
- `'com.nvidia.BF.Rshim.Status.Failed'`

The default status is `com.nvidia.BF.Rshim.Status.Idle` and it take a while to update the status from `com.nvidia.BF.Rshim.Status.Idle` to `com.nvidia.BF.Rshim.Status.InProgress` after the `SimpleUpdate` command is sent. The final status should be `com.nvidia.BF.Rshim.Status.Completed` or `com.nvidia.BF.Rshim.Status.Failed`.

## Activate the Firmware Components

Once staging is completed successfully, issue the `Activate` command. Activation is required to apply the new staged components:

```
curl -k -u root:'<password>' \
  -H "Content-Type: application/json" \
  -X POST
https://<bmc_ip>/redfish/v1/Managers/Bluefield_BMC/Actions/Oem/NvidiaManager.Activate
```

Notes on BMC firmware activation:

- Regular BFB bundle – BMC firmware is updated without needing this manual activation command.
- PLDM BFB bundle – This activation command is required to apply the new BMC firmware.

## DOCA Components Update

To complete an update to a new GA release, the DOCA Components on the Arm OS are to be updated as well. User may SSH into the DPU Arm OS and use standard Linux tools to update the DOCA components. See section "Upgrading BlueField Using Standard Linux Tools" in [DOCA documentation](#) for more details.

## Applying New BFB Image

The following are different options for applying the new version:

Reset Type	Mode of Operation	Applying Reset Steps	Notes
Cold Boot (AC/DC	<ul style="list-style-type: none"><li>• DPU Mode</li></ul>	1. (DPU Mode only) Gracefully shut down the BlueField Arm OS.	<ul style="list-style-type: none"><li>• The firmware update is</li></ul>

Reset Type	Mode of Operation	Applying Reset Steps	Notes
Power Cycle)	<ul style="list-style-type: none"> <li>NIC Mode</li> </ul>	2. Perform a full server power cycle.	<p>applied automatically during power-up.</p> <ul style="list-style-type: none"> <li>DPU Mode only: The BlueField Arm OS must be manually shut down before the reboot; otherwise, the update will not apply.</li> </ul>
Standard Warm Reboot	<ul style="list-style-type: none"> <li>DPU Mode</li> <li>NIC Mode</li> </ul>	<ol style="list-style-type: none"> <li>(DPU Mode only) Gracefully shut down the BlueField Arm OS.</li> <li>Perform a server warm reboot.</li> </ol>	<ul style="list-style-type: none"> <li>Updates firmware and software after reboot.</li> <li>DPU Mode only: The BlueField Arm OS must be manually shut down before the reboot; otherwise, the update will not apply.</li> </ul>
Coordinated Reset (Server + DPU)	DPU Mode	<ol style="list-style-type: none"> <li>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</li> </ol>	<ul style="list-style-type: none"> <li>Relevant to Deferred Update Flow only.</li> <li>The next warm reboot will: <ul style="list-style-type: none"> <li>Gracefully shut down BlueField Arm cores</li> </ul> </li> </ul>

Reset Type	Mode of Operation	Applying Reset Steps	Notes
		<p>reboot. This coordinated reset is required to apply the new firmware and software versions.</p> <pre> mlxreg -d /dev/mst/&lt;device&gt; -y --set "reset_trigger=c" -- reg_name="MFRL" </pre> <p>2. Perform a server warm reboot.</p>	<ul style="list-style-type: none"> <li>Reset the NIC, Arm Complex, and BMC</li> <li>Boot from the new firmware image</li> </ul>

## Verify New Components are Running

After DPU reboots, check that the components have been updated:

```

curl -k -u root:'<Password>' -X GET https://<bmc ip>/redfish/v1/UpdateService/FirmwareInventory/DPU_NIC
curl -k -u root:'<Password>' -X GET https://<bmc ip>/redfish/v1/UpdateService/FirmwareInventory/DPU_ATF
curl -k -u root:'<Password>' -X GET https://<bmc ip>/redfish/v1/UpdateService/FirmwareInventory/DPU_UEFI

```

## Troubleshooting Scenarios

- If RShim is disabled:

```

{
  "error": {
    "@Message.ExtendedInfo": [
      {

```

```

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

```

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

```

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

```

```
operation failed."
    }
  ]
}
```

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

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

### Info

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

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

Where:

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

Then repeat the following steps:

1. [Preparing Secure Access for Image Transfer.](#)
2. [Offline Update Flow.](#)

- If the BMC identity is not confirmed:

```
{
  "@odata.type":
"#MessageRegistry.v1_4_1.MessageRegistry",
  "Message": "Transfer of image '<file_name>' to
'/dev/rshim0/boot' failed.",
  "MessageArgs": [
    "<file_name>",
    "/dev/rshim0/boot"
  ],
  "MessageId": "Update.1.0.TransferFailed",
  "Resolution": "Unauthorized Client: Please use the
PublicKeyExchange action to receive the system's public key
and add it as an authorized key on the remote server",
  "Severity": "Critical"
}
```

```
...
"PercentComplete": 0,
  "StartTime": "<start_time>",
  "TaskMonitor":
"/redfish/v1/TaskService/Tasks/<task_id>/Monitor",
  "TaskState": "Exception",
  "TaskStatus": "Critical"
```

### Info

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

- If SCP fails:

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

```
"TaskMonitor":  
"/redfish/v1/TaskService/Tasks/<task_id>/Monitor",  
"TaskState": "Exception",  
"TaskStatus": "Critical"
```

---

# Boot Configuration

BMC supports boot option selection commands using the Redfish or IPMI interfaces. UEFI on NVIDIA® BlueField® can query for the boot options through an IPMI/Redfish command. The BMC IPMI command supports changing the boot device selector flag only through the following options: PXE boot, or the default boot device as selected in the boot menu on BlueField. In contrast, the Redfish interface supports all available boot options.

## Boot Config Using Redfish

### Retrieving Active Boot Configuration Values

- To retrieve the active boot configuration, run:

```
curl -k -u root:'<password>' -X GET  
https://<bmc_ip>/redfish/v1/Systems/Bluefield
```

#### Info

The relevant configurations would be under `Boot`.

- To retrieve all boot options (active and pending):

```
curl -k -u root:'<password>' -X GET  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/BootOptions/
```

- To retrieve detailed information on a specific boot option:

```
curl -k -u root:'<password>' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/BootOptions/<boo
option>
```

## Retrieving Information on Pending Boot Configurations

- To retrieve the pending boot settings:

```
curl -k -u root:'<password>' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings
```

- The following command retrieves only `BootOptions` configurations with a pending value different than the active one.

```
curl -k -u root:'<password>' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings/BootOpt
```

- To retrieve the details of a specific pending boot option:

```
curl -k -u root:'<password>' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings/BootOpt
id>
```

## Applying Pending Boot Configurations

## **Note**

Power reset of the BlueField is necessary for these changes to take effect.

- To alter the boot configuration, applying patches to the setting attribute is required :

```
curl -k -u root:'<password>' -X PATCH
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings -d
'{"Boot":{ ... }{'
```

- To set the pending boot order. The list must contain all the Boot option, even if the boot option is disabled.

```
curl -k -u root:'<password>' -X PATCH
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings/
-d '{"Boot":{ "BootOrder": ["Boot0002", ..., "BootXXX"]
}}{'
```

- To alter the bootOption value, currently supporting only BootOptionEnable

```
curl -k -u root:'<password>' -X PATCH
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings/BootOpt
id> -d '{"BootOptionEnabled": false}'
```

## **Changing BootOrder Configuration**

To set boot order using boot order schema, follow this procedure:

1. Check the current boot order by doing GET on the `ComputerSystem` schema over 1GbE OOB to the BlueField BMC. Look for the `BootOrder` attribute under the `Boot`.

```
curl -k -X GET -u root:<password> https://<BF-BMC-IP>/redfish/v1/Systems/<SystemID>/ | python3 -m json.tool
{
  ....
  "Boot": {
    ....
  "BootOrder": [
    "Boot0017",
    "Boot0001",
    "Boot0002",
    "Boot0003",
    "Boot0004",
    "Boot0005",
    "Boot0006",
    "Boot0007",
  ],
  ....
}
....
}
```

2. To get the details of a particular entity in the `BootOrder` array, perform a GET to the respective `BootOption` URL over 1GbE OOB to the BlueField BMC. For example, to get details of `Boot0006`, run:

```
curl -k -X GET -u root:<password> https://<BF-BMC-IP>/redfish/v1/Systems/<SystemID>/BootOptions/Boot0006 |
python3 -m json.tool
```

```

{
  "@odata.type": "#BootOption.v1_0_3.BootOption",
  "@odata.id":
"/redfish/v1/Systems/SystemId/BootOptions/Boot0006",
  "Id": "Boot0006",
  "BootOptionEnabled": true,
  "BootOptionReference": "Boot0006",
  "DisplayName": "UEFI HTTPv6 (MAC:B8CEF6B8A006)",
  "UefiDevicePath":
"PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/Pci(0x0,0
}

```

- To change the boot order, the entire `BootOrder` array must be PATCHed to the pending settings URI. For this example of the `BootOrder` array, if you intend to have `Boot0006` at the beginning of the array, then the PATCH operation is as follows:

### **(i) Note**

Updating the `BootOrder` array results in a permanent boot order change (persistent across reboots).

```

curl -k -u root:<password> -X PATCH -d '{ "Boot": {
"BootOrder": [ "Boot0006", "Boot0017", "Boot0001",
"Boot0002", "Boot0003", "Boot0004", "Boot0005", "Boot0007", ]
}}' https://<BF-BMC-
IP>/redfish/v1/Systems/<SystemID>/Settings | python3 -m
json.tool

```

- After a successful PATCH, reboot the BlueField and check if the settings have been applied by doing a GET on the `ComputerSystem` schema.

5. If the `BootOrder` array is updated as intended then the settings have been applied and the BlueField should boot as per the order in preceding cycles.
6. If `BootSourceOverrideEnabled` is set to `Once`, boot override is disabled and any related properties are reset to their former values to avoid repetition. If it is set to `Continuous`, then on every reboot, BlueField would keep performing boot override (`HTTPBoot`).

## Example of Changing BootOrder Configuration

To get the supported boot options:

```
curl -k -u root:<password>' -X GET
https://<bmc_ip>/redfish/v1/Systems/Bluefield/BootOptions
{
  "@odata.id": "/redfish/v1/Systems/Bluefield/BootOptions",
  "@odata.type": "#BootOptionCollection.BootOptionCollection",
  "Members": [
    {
      "@odata.id":
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000"
    },
    {
      "@odata.id":
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000A"
    },
    {
      "@odata.id":
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000B"
    },
    {
      "@odata.id":
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000C"
    },
    {
```

```
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000D"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000E"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot000F"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0001"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0002"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0003"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0004"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0005"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0006"  
  },  
  {
```

```
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0007"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0008"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0009"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0010"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0011"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0012"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0013"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0014"  
  },  
  {  
    "@odata.id" :  
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0015"  
  },  
  {
```

```

      "@odata.id" :
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0016"
    },
    {
      "@odata.id" :
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0017"
    },
    {
      "@odata.id" :
"/redfish/v1/Systems/Bluefield/BootOptions/Boot0040"
    }
  ],
  "Members@odata.count" : 25,
  "Name" : "Boot Option Collection"
}

```

To set the pending boot order settings:

### Info

In this example, 25 boot options are present. Therefore, the command to establish the boot option order must encompass all 25 options in the active `BootOrder` list according to the desired sequence.

```

curl -k -u root:'<password>' -X PATCH
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings -d
'{"Boot":{ "BootOrder": ["Boot0040", "Boot0017", "Boot0000",
"Boot0001", "Boot0002", "Boot0003", "Boot0004", "Boot0005",
"Boot0006", "Boot0007", "Boot0008", "Boot0009", "Boot000A",
"Boot000B", "Boot000C", "Boot000D", "Boot000E", "Boot000F",

```

```
"Boot0010", "Boot0011", "Boot0012", "Boot0013", "Boot0014",  
"Boot0015", "Boot0016"] }}}
```

## Boot Source Override

Boot Source Override allows administrators to remotely control the system's boot sequence without requiring physical access to configure boot order settings. This capability supports one-time or persistent boot source overrides, making it useful for automated OS deployment, system recovery, remote diagnostics, and enforcing specific security boot policies.

By dynamically setting the boot target (e.g., PXE, UEFI HTTP), administrators gain flexibility for provisioning, firmware updates, and disaster recovery workflows.

## Boot Source Override Config Using Redfish

### Get Boot Source Override Configuration

To retrieve the current boot source override configuration:

```
curl -k -u root:'<password>' -X GET  
https://<bmc_ip>/redfish/v1/Systems/Bluefield
```

Example output:

```
"Boot": {  
  ...  
  "BootSourceOverrideEnabled": "Disabled",  
  "BootSourceOverrideMode": "UEFI",  
  "BootSourceOverrideTarget": "None",  
  "HttpBootUri": "path",  
  "StopBootOnFault": "Never",  
  "UefiTargetBootSourceOverride": "None"
```

```
...  
}
```

## Set Boot Source Override Configuration

To set the boot source override, use:

```
curl -k -u root:'<password>' -X PATCH \  
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings \  
-d '{  
  "Boot":{  
    "BootSourceOverrideEnabled": "Once",  
    "BootSourceOverrideMode": "UEFI",  
    "BootSourceOverrideTarget": "UefiHttp",  
    "UefiTargetBootSourceOverride": "None",  
    "BootNext": "",  
    "AutomaticRetryConfig": "Disabled"  
  }  
'
```

### Note

The boot override setting take effect on the next boot and are reflected in the `redfish/v1/Systems/Bluefield` boot schema.

The following parameters can be set when configuring the boot source via the Redfish command:

- `BootSourceOverrideEnabled` –
  - `Disabled` – Disable override

- `Once` – Apply override for next boot only
- `Continuous` – Always use the boot override setting
- `BootSourceOverrideMode` – Must be set to `UEFI`
- `BootSourceOverrideTarget` – Must be one of the values allowed under `Boot/BootSourceOverrideTarget@Redfish.AllowableValues`
- `UefiTargetBootSourceOverride` – required if `BootSourceOverrideTarget` is set to `UefiTarget`. Set to the `UefiDevicePath` attribute of the desired boot option.
- `BootNext` – Used if `BootSourceOverrideTarget` is set to `UefiBootnext`
- `AutomaticRetryConfig` – Only `Disabled` is supported

## Examples

### Set Next Boot to a Specific UEFI HTTP Target

1. Query the `BootOptions` attributes:

```
curl -k -u root:'<password>' -X GET \
https://<bmc_ip>/redfish/v1/Systems/Bluefield/BootOptions/Boot
```

Example output:

```
{
  "BootOptionReference" : "Boot0002",
  "DisplayName" : "NET-OOB-IPV4-HTTP",
  "UefiDevicePath" : "MAC(B83FD2CA4B27,0x1)/IPv4(0.0.0.0,0x0,DHCP,...)/Uri()"
}
```

2. Use the `UefiDevicePath` value as the `UefiTargetBootSourceOverride`:

```
curl -k -u root:'<password>' -X PATCH \
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings \
-d '{
  "Boot":{
    "BootSourceOverrideEnabled": "Once",
    "BootSourceOverrideMode": "UEFI",
    "BootSourceOverrideTarget": "UefiTarget",
    "UefiTargetBootSourceOverride":
"MAC(B83FD2CA4B27,0x1)/IPv4(0.0.0.0,...)/Uri()",
    "AutomaticRetryConfig": "Disabled"
  }
}'
```

### Set Next Boot to PXE (Non-persistent)

```
curl -k -u root:'<password>' -X PATCH \
https://<bmc_ip>/redfish/v1/Systems/Bluefield/Settings \
-d '{
  "Boot":{
    "BootSourceOverrideEnabled": "Once",
    "BootSourceOverrideMode": "UEFI",
    "BootSourceOverrideTarget": "Pxe"
  }
}'
```

## Boot Source Override Config Using IPMI

The `ipmitool` utility allows configuring the Boot Source Override option, enabling the system to boot from a PXE server or another specified device.

## Retrieving the Current Boot Override Settings

To view the current boot override configuration, run:

```
ipmitool chassis bootparam get 5
```

This command returns information about:

- Whether the boot override option is valid.
- Whether it is persistent or applies to the next boot only.
- The configured boot device type.

## Configuring One-Time PXE Boot with Timeout

To configure a one-time PXE boot with a 60-second timeout, use the following command:

```
ipmitool chassis bootparam set bootflag force_pxe  
options=timeout
```

### **Note**

If the DPU is not reset within 60 seconds, the boot parameters will be invalidated.

## Configuring One-Time PXE Boot without Timeout

To configure a one-time PXE boot without the 60-second timeout, run:

```
ipmitool chassis bootparam set bootflag force_pxe options=no-  
timeout
```

### **(i) Note**

The boot override timer is only applicable for BlueField-3.

### **(i) Note**

It is not recommended to use `ipmitool chassis bootparam` without explicitly specifying the `options` parameter, as this may result in unpredictable timer behavior.

## **Resetting Boot Override to Default**

To clear the boot override and return to the default boot device, use:

```
ipmitool chassis bootparam set bootflag none
```

## **Setting Persistent PXE Boot**

To configure the system to always boot from PXE (persistent override), execute:

```
ipmitool chassis bootdev pxe options=persistent
```

### **i Info**

The persistent option prevents the 60-second timeout from being triggered.

### **i Info**

If you modify bootdev or bootparam settings without explicitly specifying `options=persistent`, the persistent configuration will be disabled.

## **Behavior of Boot Source Override on BlueField**

The Boot Source Override configuration set through the BMC remains persistent until one of the following occurs:

- It is explicitly reset to `none`.
- The BFB image is updated, which will clear the override settings.

## 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 12/15/2025