

## Data Center / Cloud

[Forums](#)[Sign In](#)[Search](#)

### Generative AI / LLMs

### Robotics



### Content Creation / Rendering

### Data Science

NVIDIA Doc  
Zero Trust (

### Networking

king Solutions > RDG for DPF  
reman services

### Simulation / Modeling / Design

On Thi

### Conversational AI

[Scope](#)[Abbreviations and Acronyms](#)[Introduction](#)[References](#)[Solution Architecture](#)[Key Components and Technologies](#)[Solution Design](#)[Solution Logical Design](#)[Firewall Design](#)[Software Stack Components](#)[Bill of Materials](#)[Deployment and Configuration](#)[Node and Switch Definitions](#)[Wiring](#)[Hypervisor Node](#)[Bare Metal Worker Node](#)[Fabric Configuration](#)[Updating Cumulus Linux](#)[Configuring the Cumulus Linux Switch](#)[Host Configuration](#)[Hypervisor Installation and Configuration](#)[Prepare Infrastructure Servers](#)[Provision Master VMs Using MaaS](#)[K8s Cluster Deployment and Configuration](#)[DPF Installation](#)[Software Prerequisites and Required Variables](#)[DPF Operator Installation](#)[DPF System Installation](#)[DPU Service Installation](#)[Verification](#)[Authors](#)[Is this page helpful?](#)

# RDG for DPF Zero Trust (DPF-ZT) with DOCA Telemetry Service(DTS) and Blueman services

Created on Nov 13, 2025

*Updated on Jan 18, 2026 (DPF 25.10.0 GA)*

## Scope

This Reference Deployment Guide (RDG) provides comprehensive instructions for deploying the NVIDIA DOCA Platform Framework (DPF) on high-performance, bare-metal infrastructure in **Zero-Trust mode**. The guide focuses on setting up the **DOCA Telemetry Service (DTS)** and the **BlueMan** Service on **NVIDIA® BlueField®-3 DPUs**, enabling a unified interface for accessing essential DPU information, health status, and telemetry metrics.

The guide is intended for experienced system administrators, systems engineers, and solution architects who build highly secure bare-metal environments with Host-Based Networking enabled using NVIDIA BlueField DPUs for acceleration, isolation, and infrastructure offload.

This document is an extension of the [\*\*RDG for DPF Zero Trust \(DPF-ZT\) - NVIDIA Docs\*\*](#) (referred to as the **Baseline RDG**). It details the additional steps and modifications required to deploy the DTS and BlueMan Services into the Baseline RDG environment.

### ✓ Note

- This reference implementation, as the name implies, is a specific, opinionated deployment example designed to address the use case described above.
- Although other approaches may exist for implementing similar solutions, this document provides a detailed guide for this specific method.

## Abbreviations and Acronyms

Term	Definition	Term	Definition
BFB	BlueField Bootstream	NGC	NVIDIA GPU Cloud
DOCA	Data Center Infrastructure-on-a-Chip Architecture	NFS	Network File System

Term	Definition	Term	Definition
DPF	DOCA Platform Framework	OOB	Out-of-Band
DPU	Data Processing Unit	RDG	Reference Deployment Guide
DTS	DOCA Telemetry Service	RDMA	Remote Direct Memory Access
K8S	Kubernetes	RoCE	RDMA over Converged Ethernet
KVM	Kernel-based Virtual Machine	VPC	Virtual Private Cloud
MAAS	Metal as a Service	ZT	Zero Trust
MTU	Maximum Transmission Unit		

## Introduction

The **NVIDIA BlueField-3 Data Processing Unit (DPU)** is a 400 Gb/s infrastructure compute platform designed for line-rate processing of software-defined networking, storage, and cybersecurity workloads. It combines powerful compute resources, high-speed networking, and advanced programmability to deliver **hardware-accelerated, software-defined solutions** for modern data centers.

**NVIDIA DOCA** unleashes the full potential of the BlueField platform by enabling rapid development of applications and services that **offload, accelerate, and isolate** data center workloads.

**DOCA Telemetry Service (DTS)** collects data from built-in providers (data providers such as `sysfs`, `ethtool` and `tc`, and aggregation providers such as `fluent_aggr` and `prometheus_aggr`), and from external telemetry applications.

**DOCA BlueMan** runs in the DPU as a standalone web dashboard and consolidates all the basic information, health, and telemetry counters into a single interface.

All the information that BlueMan provides is gathered from the DOCA Telemetry Service (DTS).

However, deploying and managing DPUs, especially at scale, presents operational challenges. Without a robust provisioning and orchestration system, tasks such as lifecycle management, service deployment, and network configuration for service function chaining (SFC) can quickly become **complex and error prone**. This is where the **DOCA Platform Framework (DPF)** comes into play.

**DPF** automates the full DPU lifecycle, and simplifies advanced network configurations. With DPF, services can be deployed seamlessly, allowing for **efficient offloading and intelligent routing** of traffic through the DPU data plane.

By leveraging DPF, users can **scale and automate DPU management** across Bare Metal, Virtual, and Kubernetes customer environments - **optimizing performance while simplifying operations**.

DPF supports multiple deployment models. This guide focuses on the **Zero Trust bare-metal deployment model**. In this scenario:

- The DPU is **managed** through its **Baseboard Management Controller (BMC)**
- All management traffic occurs over the **DPU's out-of-band (OOB)** network
- The host is considered as an untrusted entity towards the data center network. The DPU acts as a barrier between the host and the network.
- The host sees the **DPU** as a standard NIC, with no access to the internal DPU management plane (**Zero Trust Mode**)

This **Reference Deployment Guide (RDG)** provides a **step-by-step example** for installing DPF in Zero-Trust mode. It also includes **practical demonstrations of performance optimization**, validated using standard **RDMA and TCP workloads**.

As part of the reference implementation, **open-source components outside the scope of DPF** (e.g., MAAS, pfSense, Kubespray) are used to simulate a realistic customer deployment environment. The guide includes the full end-to-end deployment process, including:

- Infrastructure provisioning
- DPF deployment
- DPU provisioning (redfish)
- Service configuration and deployment
- Service chaining.

This document is an extension of the **RDG for DPF Zero Trust (DPF-ZT) - NVIDIA Docs** (referred to as the **Baseline RDG**). It details the additional steps and modifications required to deploy the DTS and BlueMan Services into the Baseline RDG environment.

## References

- [\*\*NVIDIA BlueField DPU\*\*](#)
- [\*\*NVIDIA DOCA\*\*](#)
- [\*\*NVIDIA DPF Release Notes\*\*](#)
- [\*\*NVIDIA DPF GitHub Repository\*\*](#)
- [\*\*NVIDIA DPF System Overview\*\*](#)
- [\*\*NVIDIA Ethernet Switching\*\*](#)
- [\*\*NVIDIA Cumulus Linux\*\*](#)
- [\*\*What is K8s?\*\*](#)
- [\*\*Kubespray\*\*](#)

## Solution Architecture

### Key Components and Technologies

- [\*\*NVIDIA BlueField® Data Processing Unit \(DPU\)\*\*](#)

The NVIDIA® BlueField® data processing unit (DPU) ignites unprecedented innovation for modern data centers and supercomputing clusters. With its robust compute power and integrated software-defined hardware accelerators for networking, storage, and security, BlueField creates a secure and accelerated infrastructure for any workload in any environment, ushering in a new era of accelerated computing and AI.

- [\*\*NVIDIA DOCA Software Framework\*\*](#)

NVIDIA DOCA™ unlocks the potential of the NVIDIA® BlueField® networking platform. By harnessing the power of BlueField DPUs and SuperNICs, DOCA enables the rapid creation of applications and services that offload, accelerate, and isolate data center workloads. It lets developers create software-defined, cloud-native, DPU- and SuperNIC-accelerated services with zero-trust protection, addressing the performance and security demands of modern data centers.

- [\*\*NVIDIA ConnectX SmartNICs\*\*](#)

10/25/40/50/100/200 and 400G Ethernet Network Adapters

The industry-leading NVIDIA® ConnectX® family of smart network interface cards (SmartNICs) offer advanced hardware offloads and accelerations.

NVIDIA Ethernet adapters enable the highest ROI and lowest Total Cost of Ownership for hyperscale, public and private clouds, storage, machine learning, AI, big data, and telco platforms.

- [\*\*NVIDIA LinkX Cables\*\*](#)

The NVIDIA® LinkX® product family of cables and transceivers provides the industry's most complete line of 10, 25, 40, 50, 100, 200, and 400GbE in Ethernet and 100, 200 and 400Gb/s InfiniBand products for Cloud, HPC,

hyperscale, Enterprise, telco, storage and artificial intelligence, data center applications.

- **NVIDIA Spectrum Ethernet Switches**

Flexible form-factors with 16 to 128 physical ports, supporting 1GbE through 400GbE speeds.

Based on a ground-breaking silicon technology optimized for performance and scalability, NVIDIA Spectrum switches are ideal for building high-performance, cost-effective, and efficient Cloud Data Center Networks, Ethernet Storage Fabric, and Deep Learning Interconnects.

NVIDIA combines the benefits of **NVIDIA Spectrum™** switches, based on an industry-leading application-specific integrated circuit (ASIC) technology, with a wide variety of modern network operating system choices, including

**NVIDIA Cumulus® Linux**, **SONiC** and **NVIDIA Onyx®**.

- **NVIDIA Cumulus Linux**

NVIDIA® Cumulus® Linux is the industry's most innovative open network operating system that allows you to automate, customize, and scale your data center network like no other.

- **Kubernetes**

Kubernetes is an open-source container orchestration platform for deployment automation, scaling, and management of containerized applications.

- **Kubespray**

Kubespray is a composition of **Ansible** playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks and provides:

- A highly available cluster
- Composable attributes
- Support for most popular Linux distributions

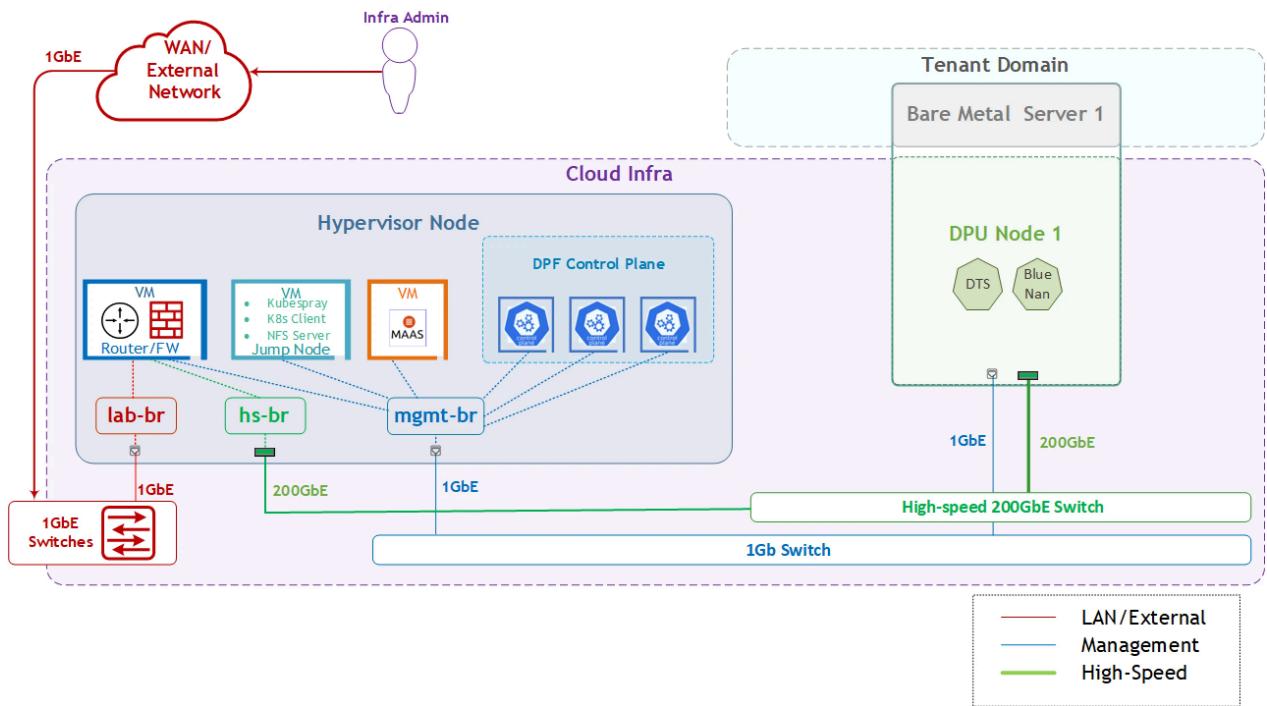
## **Solution Design**

### **Solution Logical Design**

The logical design includes the following components:

- 1 x Hypervisor node (KVM-based) with ConnectX-7:
  - 1 x Firewall VM
  - 1 x Jump Node VM
  - 1 x MaaS VM
  - 3 x K8s Master VMs running all K8s management components
- 1 x Worker nodes (PCI Gen5), each with a 1 x BlueField-3 NIC
- Single High-Speed (HS) switch

- 1 Gb Host Management network



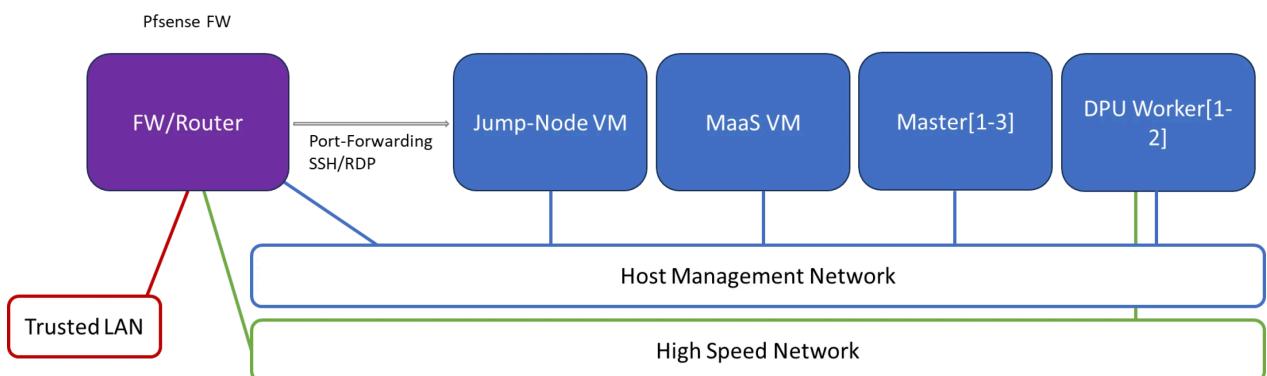
## Firewall Design

The pfSense firewall in this solution serves a dual purpose:

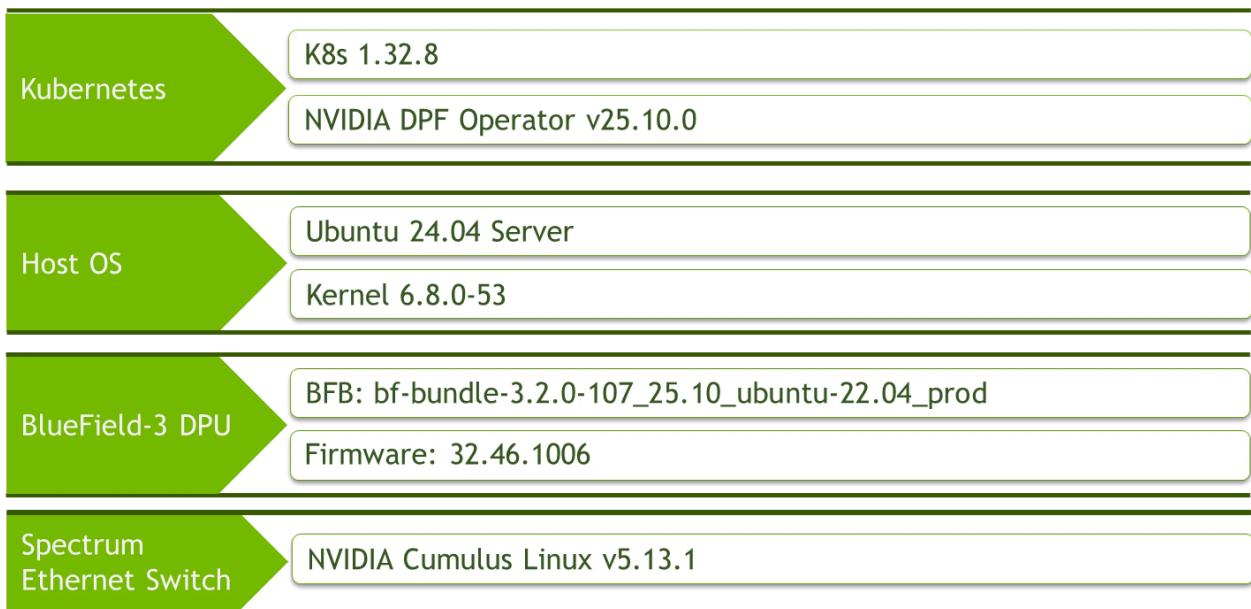
- Firewall—provides an isolated environment for the DPF system, ensuring secure operations
- Router—enables Internet access for the management network

Port-forwarding rules for SSH and RDP are configured on the firewall to route traffic to the jump node's IP address in the host management network. From the jump node, administrators can manage and access various devices in the setup, as well as handle the deployment of the Kubernetes (K8s) cluster and DPF components.

The following diagram illustrates the firewall design used in this solution:



## Software Stack Components



## ⓘ Warning

Make sure to use the **exact same versions** for the software stack as described above.

## Bill of Materials

#	Part	OPN	Qnty	Description
1	Hypervisor Server	x86 based server	1	x86 server system to run all virtualized instances CPU: 2 x Intel Xeon Platinum 8168 (24 cores @ 2.7GHz) with PCI Gen4x32 RAM: 384GB Storage: 1.6TB (NVMe) Network: ConnectX-7 dual-port adapter
2	Bare-Metal Server	x86 based server	1	x86 server system CPU: 2 x Intel(R) Xeon(R) Platinum 8380 (40 cores @ 2.3GHz) with PCI Gen4x32 RAM: 256GB Storage: 3.8TB (NVMe) Network: BlueField-3 P-Series
3	SN2201 Ethernet Switch	MSN2201-CB2FC	1	Spectrum OOB Management Switch with 48 RJ45 1GbT + 4 QSFP28 100GbE Ports 
4	MSN3700 Ethernet Switch	MSN3700-VS2FC	1	Spectrum-2 Based 200GbE 1U Open Ethernet Switch 32 QSFP56 Ports (Used for High-speed Network) 
5	Ethernet Direct Attach Copper Cables	MCP1650-H	2	QSFP56 passive copper cable (200Gbps) 
6	BlueField-3 DPU	900-9D3B6-00CV-AA0	1	BlueField-3 B3220 P-Series DPU, dual-port 200GbE QSFP112, PCIe Gen 5.0 x16 16 Arm cores, 32GB memory

*The network adapter cards are used in item #2.*

## Deployment and Configuration

## Node and Switch Definitions

These are the definitions and parameters used for deploying the demonstrated fabric:

Switches Ports Usage		
Hostname	Rack ID	Ports
mgmt-switch	1	swp1-2
hs-switch	1	swp1-2

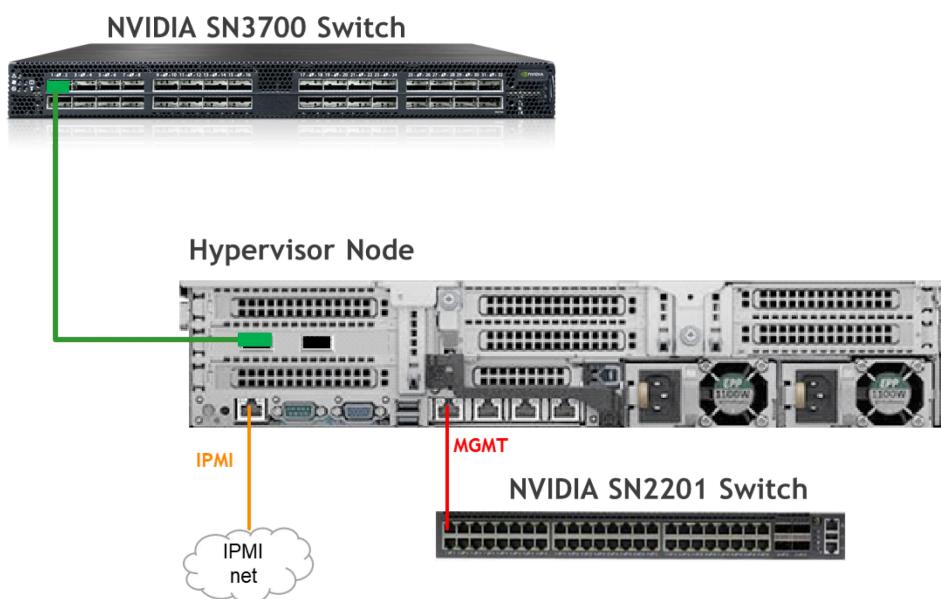
**Expand**

Hosts				
Rack	Server Type	Server Name	Switch Port	IP and NICs
Rack1	Hypervisor Node	hypervisor	mgmt-switch: swp1  hs-switch: swp1	lab-br (interface eno1): Trusted LAN IP mgmt-br (interface eno2): - hs-br (interface enp1s0): -
Rack1	Firewall (Virtual)	fw	-	WAN (lab-br): Trusted LAN IP LAN (mgmt-br): 10.0.110.254/24 OPT1(hs-br): 10.0.123.254/22

<b>Hosts</b>				
Rack1	Jump Node (Virtual)	jump	-	<b>enp1s0:</b> 10.0.110.253/24
Rack1	MaaS (Virtual)	maas	-	<b>enp1s0:</b> 10.0.110.252/24
Rack1	Master Node (Virtual)	master1	-	<b>enp1s0:</b> 10.0.110.1/24
Rack1	Master Node	master2	-	<b>enp1s0:</b> 10.0.110.2/24

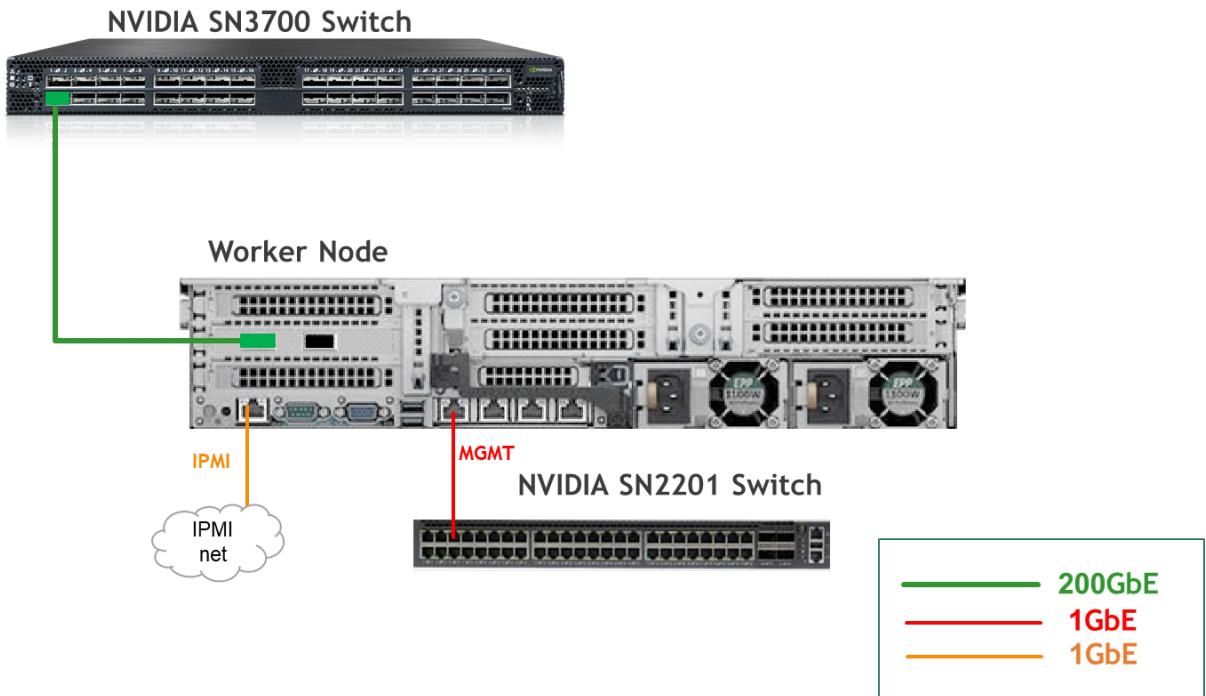
## Wiring

### Hypervisor Node



—	200GbE
— —	1GbE

### Bare Metal Worker Node



## Fabric Configuration

### Updating Cumulus Linux

As a best practice, make sure to use the latest released Cumulus Linux NOS version.

For information on how to upgrade Cumulus Linux, refer to the [Cumulus Linux User Guide](#).

### Configuring the Cumulus Linux Switch

The SN3700 switch (`hs-switch`), is configured as follows:

▼ Collapse Source

**SN3700 Switch Console**

```
nv set bridge domain br_hs untagged 1
nv set interface swp1-2 bridge domain br_hs
nv set interface swp1-2 link state up
nv set interface swp1-2 type swp
nv config apply -y
nv config save -y
```

The SN2201 switch (`mgmt-switch`) is configured as follows:

▼ **Collapse Source**

## SN2201 Switch Console

```
nv set interface swp1-2 link state up
nv set interface swp1-2 type swp
nv set interface swp1-2 bridge domain br_default
nv set bridge domain br_default untagged 1
nv config apply
nv config save -y
```

## Host Configuration

### **Warning**

Make sure that the BIOS settings on the worker node servers have SR-IOV enabled and that the servers are tuned for maximum performance.

All worker nodes must have the same PCIe placement for the BlueField-3 NIC and must display the same interface name.

Make sure that you have DPU BMC and OOB MAC addresses.

No change from the [\*\*Reference Deployment Guide \(Baseline RDG\)\*\*](#) (Section "Deployment and Configuration", Subsection " Host Configuration ").

## Hypervisor Installation and Configuration

No change from the Baseline RDG (Section "Deployment and Configuration", Subsection "Hypervisor Installation and Configuration").

## Prepare Infrastructure Servers

No change from the Baseline RDG (Section "Deployment and Configuration", Subsection "Prepare Infrastructure Servers") regarding Firewall VM, Jump VM, MaaS VM.

## Provision Master VMs Using MaaS

No change from the Baseline RDG (Section "Deployment and Configuration", Subsection "Provision Master VMs Using MaaS").

## K8s Cluster Deployment and Configuration

The procedures for initial Kubernetes cluster deployment using Kubespray for the master nodes, and subsequent verification, remain unchanged from the Baseline RDG (Section "K8s Cluster Deployment and Configuration", Subsections: "Kubespray Deployment and Configuration", "Deploying Cluster Using Kubespray Ansible Playbook", "K8s Deployment Verification").

## DPF Installation

The DPF installation process (Operator, System components) largely follows the Baseline RDG.

## Software Prerequisites and Required Variables

1. Start by installing the remaining **software perquisites**.

 **Collapse Source**

### Jump Node Console

```
## Connect to master1 to copy helm client utility that was installed
$ depuser@jump:~$ ssh master1
depuser@master1:~$ cp /usr/local/bin/helm /tmp/

## In another tab
depuser@jump:~$ scp master1:/tmp/helm /tmp/
depuser@jump:~$ sudo chown root:root /tmp/helm
depuser@jump:~$ sudo mv /tmp/helm /usr/local/bin/

## Verify that envsubst utility is installed
depuser@jump:~$ which envsubst
/usr/bin/envsubst
```

2. Proceed to clone the **doca-platform Git repository**:

 **Collapse Source**

### Jump Node Console

```
$ git clone https://github.com/NVIDIA/doca-platform.git
```

3. Change directory to **doca-platform** and checkout to **tag v25.10.0**:

▼ **Collapse Source**

### Jump Node Console

```
$ cd doca-platform/
$ git checkout v25.10.0
```

4. Before deploying the objects under

`doca-platform/dpuservices/dts-blueman/` directory, a few adjustments are required. Create a directory from where all the commands will be run:

▼ **Collapse Source**

### Jump Node Console

```
$ mkdir doca-platform/dpuservices/dts-blueman/
$ cd doca-platform/dpuservices/dts-blueman/
```

5. Change the BMC root's password.

In Zero Trust mode, provisioning DPUs requires authentication with Redfish. In order to do that, you must set the same root password to access the BMC for all DPUs DPF is going to manage. For more information on how to set the BMC root password refer to [BlueField DPU Administrator Quick Start Guide](#). Connect to the DPU BMC over SSH to change the BMC root's password on all DPUs.

▼ **Collapse Source**

### Jump Node Console

```
$ ssh root@10.0.110.201
root@10.0.110.201's password: <BMC Root Password. Default root/ope
```

6. Modify the variables in `manifests/00-env-vars/envvars.env` to fit your environment, then source the file:

### ⓘ Warning

Replace the values for the variables in the following file with the values that fit your setup. Specifically, pay attention to `DPUCLUSTER_INTERFACE`,

[BMC\_ROOT\_PASSWORD].

**Expand**

▼ **Collapse Source**

## envvars.env

```
## IP Address for the Kubernetes API server of the target cluster
## This should never include a scheme or a port.
## e.g. 10.10.10.10
export TARGETCLUSTER_API_SERVER_HOST=10.0.110.10

## Virtual IP used by the load balancer for the DPU Cluster. Must
## be allocated by DHCP.
export DPUCLUSTER_VIP=10.0.110.200

## Interface on which the DPUCluster load balancer will listen. Should
## be ens160
export DPUCLUSTER_INTERFACE=ens160

## IP address to the NFS server used as storage for the BFB.
export NFS_SERVER_IP=10.0.110.253

## The repository URL for the NVIDIA Helm chart registry.
## Usually this is the NVIDIA Helm NGC registry. For development purposes
## export HELM_REGISTRY_REPO_URL=https://helm.ngc.nvidia.com/nvidia/doca

## The DPF REGISTRY is the Helm repository URL where the DPF Operator
## is deployed from. Usually this is the NVIDIA Helm NGC registry. For development
## purposes
## export REGISTRY=https://helm.ngc.nvidia.com/nvidia/doca

## The DPF TAG is the version of the DPF components which will be
## deployed
## export TAG=v25.10.0

## URL to the BFB used in the `bfb.yaml` and linked by the DPUSet.
## export BFB_URL="http://nbu-nfs.mellanox.com/auto/sw_mc_soc_release"

## IP_RANGE_START and IP_RANGE_END
## These define the IP range for DPU discovery via Redfish/BMC interface
## Example: If your DPUs have BMC IPs in range 10.0.110.201-224

## Start of DPUDiscovery IpRange
## export IP_RANGE_START=10.0.110.205
```

```
## End of DPUDiscovery IpRange
export IP_RANGE_END=10.0.110.205

# The password used for DPU BMC root login, must be the same for all DPU nodes
```

7. Export environment variables for the installation:

### Jump Node Console

 **Collapse Source**

```
$ source envvars.env
```

## **DPF Operator Installation**

No change from the Baseline RDG (Section "DPF Installation", Subsection "DPF Operator Installation").

## **DPF System Installation**

No change from the Baseline RDG (Section "DPF Installation", Subsection "DPF System Installation").

## **DPU Service Installation**

### **Change the DPUDeployment, DPUServiceConfiguration, DPUServiceTemplate yaml files.**

Before deploying the objects under `docta-platform/dpuservices/dts-bluman/` directory, a few adjustments are required.

1. Create the following YAML to define a `BFB` resource that downloads the Bluefield Bitstream to a shared volume :

### bf.yaml

 **Collapse Source**

```
---
apiVersion: provisioning.dpu.nvidia.com/v1alpha1
kind: BFB
```

```

metadata:
  name: bf-bundle
  namespace: dpf-operator-system
spec:
  url: $BFB_URL

```

2. Create the `DPUFlavor` using the following YAML:

**Expand**

▼ **Collapse Source**

### DPUFlavor.yaml

```

---
apiVersion: provisioning.dpu.nvidia.com/v1alpha1
kind: DPUFlavor
metadata:
  name: dpf-provisioning-dts-blueman
  namespace: dpf-operator-system
spec:
  dpuMode: zero-trust
  bfcfgParameters:
    - UPDATE_ATF_UEFI=yes
    - UPDATE_DPU_OS=yes
    - WITH_NIC_FW_UPDATE=yes
  configFiles:
    - operation: override
      path: /etc/mellanox/mlnx-bf.conf
      permissions: "0644"
      raw: |
        ALLOW_SHARED_RQ="no"
        IPSEC_FULL_OFFLOAD="no"
        ENABLE_ESWITCH_MULTIPORT="yes"
    - operation: override
      path: /etc/mellanox/mlnx-ovs.conf
      permissions: "0644"
      raw: |
        CREATE_OVS_BRIDGES="no"
        OVS_DOCA="yes"
    - operation: override
      path: /etc/mellanox/mlnx-sf.conf
      permissions: "0644"
      raw: ""

```

```
grub:
```

```
    kernelParameters:
        - console=hvc0
        - console=ttyAMA0
        - earlycon=p1011,0x13010000
        - fixrttc
        - net.ifnames=0
        - biosdevname=0
        - iommu.passthrough=1
        - cgroup_no_v1=net_prio,net_cls
```

3. Create the `[DPUDeployment.yaml]` file:

**▼ Collapse Source**

### DPUDeployment.yaml

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUDeployment
metadata:
  name: dts-blueman
  namespace: dpf-operator-system
spec:
  dpus:
    bfb: bf-bundle
    dpuSets:
      - nameSuffix: dpuset-dts-blueman
        nodeSelector:
          matchLabels:
            feature.node.kubernetes.io/dpu-enabled: "true"
        dpuSelector:
          provisioning.dpu.nvidia.com/dpudevice-service-name: dts-b]
    flavor: dpf-provisioning-dts-blueman
    nodeEffect:
      hold: true
  services:
    dts:
      serviceTemplate: dts
      serviceConfiguration: dts
    blueman:
      serviceTemplate: blueman
      serviceConfiguration: blueman
```

### ✓ Note

Please notice that with default nodeEffect above, DPU provisioning workflow will be paused and wait for an external signal (annotation) in order to proceed, as demonstrated in upcoming steps.

To implement a fully automated process that won't require user intervention, see [\*\*customAction option\*\*](#).

4. Create the `DPUServiceconfig_dts.yaml` file:

### DPUServiceconfig\_dts.yaml

✓ [Collapse Source](#)

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceConfiguration
metadata:
  name: dts
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "dts"
```

5. Create the `DPUServicetemplate_dts.yaml` file:

### DPUServicetemplate\_dts.yaml

✓ [Collapse Source](#)

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceTemplate
metadata:
  name: dts
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "dts"
  helmChart:
    source:
      repoURL: $HELM_REGISTRY_REPO_URL
      version: 1.0.8
      chart: doca-telemetry
```

6. Create the `DPUServiceconfig_blueman.yaml` file:

**▼ Collapse Source**

## DPUServiceconfig\_blueman.yaml

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceConfiguration
metadata:
  name: blueman
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "blueman"
```

7. Create the `DPUServicetemplate_blueman.yaml` file:

**▼ Collapse Source**

## DPUServiceTemplate.yaml

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceTemplate
metadata:
  name: blueman
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "blueman"
  helmChart:
    source:
      repoURL: $HELM_REGISTRY_REPO_URL
      version: 1.0.8
      chart: doca-blueman
```

8. Apply all of the YAML files mentioned above using the following command:

**▼ Collapse Source**

## Jump Node Console

```
$ cat *.yaml | envsubst | kubectl apply -f -
```

## 9. To follow the progress of DPU provisioning, run the following

↙ **Collapse Source**

### Jump Node Console

```
$ watch -n10 "kubectl describe dpu -n dpf-operator-system | grep '
```

10. Wait for the **NodeEffect stage** (at this point the provisioning is paused, waiting for external signal).

Run following command on all/specific DPU nodemaintanace object/s to proceed with provisioning:

↙ **Collapse Source**

### Jump Node Console

```
$ kubectl annotate dpunodemaintenances -n dpf-operator-system --a]
```

11. To follow the progress of DPU provisioning, run the following

↙ **Collapse Source**

### Jump Node Console

```
$ watch -n10 "kubectl describe dpu -n dpf-operator-system | grep '
Every 10.0s: kubectl describe dpu -n dpf-operator-system | grep 'N'
```

Dpu Node Name:	dpu-node-mt243
----------------	----------------

Last Transition Time:	2026-01-18T13:35:01Z
-----------------------	----------------------

Type:	Initialized
-------	-------------

Last Transition Time:	2026-01-18T13:35:02Z
-----------------------	----------------------

Type:	BFBReady
-------	----------

Last Transition Time:	2026-01-18T13:35:02Z
-----------------------	----------------------

Type:	NodeEffectReady
-------	-----------------

Last Transition Time:	2026-01-18T13:35:03Z
-----------------------	----------------------

Type:	InterfaceInitialized
-------	----------------------

Last Transition Time:	2026-01-18T13:35:04Z
-----------------------	----------------------

Type:	FWConfigured
-------	--------------

Last Transition Time:	2026-01-18T13:35:04Z
-----------------------	----------------------

Type:	BFBPrepared
-------	-------------

Last Transition Time:	2026-01-18T13:44:56Z
-----------------------	----------------------

Type:	OSInstalled
Last Transition Time:	2026-01-18T13:47:59Z
Type:	Rebooted
Phase:	Rebooting

12. Wait for the **Rebooted stage** and then **Power Cycle** the bare-metal host manual.

After the DPU is up, run following command for each DPU worker:

### Jump Node Console

**▼ Collapse Source**

```
$ kubectl -n dpf-operator-system annotate dpu dpu node mt24386019aq prc
```

13. At this point, the DPU workers should be added to the cluster. As they being added to the cluster, the DPUs are provisioned.

### Jump Node Console

**▼ Collapse Source**

```
$ watch -n10 "kubectl describe dpu -n dpf-operator-system | grep 'Name: dpu-node-mt24386019aq' | tail -1"
```

Dpu Node Name: dpu-node-mt24386019aq

Type:	InternalIP
Type:	Hostname
Last Transition Time:	2026-01-18T13:35:01Z
Type:	Initialized
Last Transition Time:	2026-01-18T13:35:02Z
Type:	BFBReady
Last Transition Time:	2026-01-18T13:35:02Z
Type:	NodeEffectReady
Last Transition Time:	2026-01-18T13:35:03Z
Type:	InterfaceInitialized
Last Transition Time:	2026-01-18T13:35:04Z
Type:	FWConfigured
Last Transition Time:	2026-01-18T13:35:04Z
Type:	BFBPrepared
Last Transition Time:	2026-01-18T13:44:56Z
Type:	OSInstalled
Last Transition Time:	2026-01-18T13:55:40Z
Type:	Rebooted

```
Last Transition Time: 2026-01-18T13:55:40Z
Type: DPUClusterReady
Last Transition Time: 2026-01-18T13:55:41Z
Type: Ready
Phase: Ready
```

14. Finally, validate that all the different DPU-related objects are now in the Ready state:

### Jump Node Console

**▼ Collapse Source**

```
$ echo 'alias dpfctl="kubectl -n dpf-operator-system exec deploy/dpf-operator-system"' > /etc/bashrc
$ source /etc/bashrc
$ dpfctl describe dpudeployments
NAME                                     NAMESPACE
DPF0operatorConfig/dpfoperatorconfig      dpf-operator-system
└──DPUDeployments
    └──DPUDeployment/dts-blueman          dpf-operator-system
        ├──DPUSets
        |   └──DPUSet/dts-blueman-dpuset-dts-blueman  dpf-operator-system
        |       ├──BFB/bf-bundle                  dpf-operator-system
        |       ├──DPUNodes
        |       |   └──1 DPUNodes...            dpf-operator-system
        |       └──DPUs
        |           └──1 DPUs...              dpf-operator-system
        └──Services
            ├──DPUServiceTemplates
            |   ├──DPUServiceTemplate/blueman  dpf-operator-system
            |   └──DPUServiceTemplate/dts      dpf-operator-system
            └──DPUServices
                └──2 DPUServices...        dpf-operator-system

$ echo "alias ki='KUBECONFIG=/home/depuser/dpu-cluster.config kubectl'" > /etc/bashrc
$ source /etc/bashrc
$ kubectl get secrets -n dpu-cplane-tenant1 dpu-cplane-tenant1-admin
$ ki get node -A
NAME                           STATUS  ROLES   AGE     VERSION
dpu-node-mt24386019aq-mt24386019aq  Ready  <none>  21m    v1.34

$ kubectl get dpu -A
NAME                         STATUS  READY
dpf-operator-system  dpu-node-mt24386019aq-mt24386019aq  True
```

```
$ kubectl wait --for=condition=ready --namespace dpf-operator-system
dpu.provisioning.dpu.nvidia.com/dpu-node-mt24386019aq-mt24386019ac
```

## **Verification**

Here's a step-by-step procedure to check the **DTS** and **Blueman DPUServices** were deployed on your NVIDIA BlueField DPU.

To be able to log into BlueMan and view the **local DTS instance** data in a convenient way, the management IP address of the DPU should be entered to a web browser located in the same network as the DPU. In this RDG, it will be demonstrated by using **RDP** to connect to the **Jump** node and opening a web browser in it (same as with MaaS, Firewall).

1. To find out the DPU management IP address in the `10.0.110.0/24` subnet, obtain the DPU names .

### **Jump Node Console**

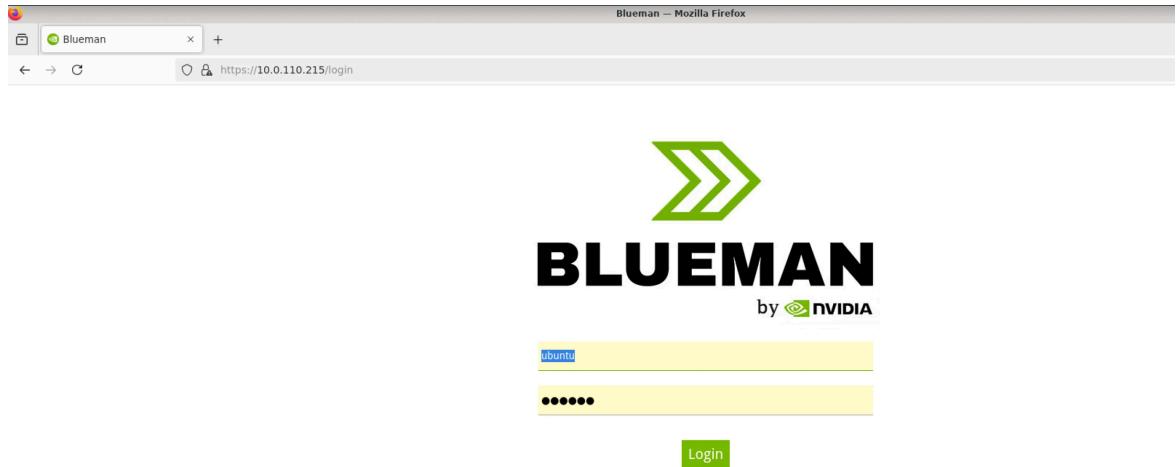
```
$ kubectl get dpus -n dpf-operator-system
NAME                           READY   PHASE   AGE
dpu-node-mt24386019aq-mt24386019ac   True    Ready   24m
```

2. Obtain the DPU management IP :

### **Jump Node Console**

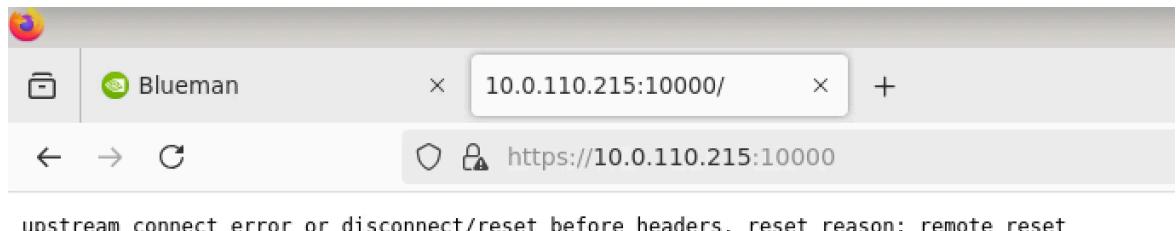
```
$ kubectl get dpus -n dpf-operator-system -o json | jq '.items[0].status.addresses[0].ip'
```

3. In the RDP session, open a web browser and enter `https://<DPU_INTERNAL_IP>`. A warning of self-signed certificate should appear; click accept the risk and proceed.  
Afterwards it will open the login page:



The login credentials to use are the same pair used for the SSH connection to the DPU ( `ubuntu/ubuntu` ). However, login straight away won't work and an additional certificate exception in the browser has to be made.

4. Open another tab in the browser and enter **[https://<DPU\\_INTERNAL\\_IP>:10000](https://<DPU_INTERNAL_IP>:10000)**. It will again prompt a warning of self-signed certificate; click accept the risk to add it to your browser exception list. An error message similar to the following will be displayed, but it doesn't matter since it's an internal address to fetch resources from—in other words, the error message can be ignored.



5. Return to the Blueman login page, enter the credentials, and you should be able to login.

The screenshot shows the Blueman management interface in Mozilla Firefox. The left sidebar has tabs for Info, Health, and Telemetry. The main area is divided into several sections:

- General Info**: Shows system parameters like Lifecycle state (GA Secured), Serial Number (MT2438019AQ), Part Number (900-90384-00CV-EAO), Bluefield OS Version (bf-bundle-3.2.0-107\_25.10\_uubuntu-24.04\_prod), DOCA Version (24.04), Driver (MLNX\_OFED\_LINUX-25.10-1.2.0), Kernel Release (6.8.0-1012-bluefield), Kernel Version (#16-Ubuntu SMP PREEMPT\_DYNAMIC Mon Oct 13 15:36:22 UTC 2025), and OS Version (24.04).
- Installed Packages**: Lists packages with their versions, such as acpid (1.2.0-34~ubuntu2), adduser (3.137~ubuntu1), apt (2.0.1~ubuntu4.0.1~ubuntu0.24.04.4), and apt-base (2.8.3).
- CPU Info**: Details architecture (aarch64), byte order (LittleEndian), number of CPUs (8), and vendor information.
- FW Info**: Shows configuration settings for Port 1, including ECP(ESWITCH\_MANAGER, PAGE\_SUPPLIER), NVM\_EMU\_MNG\_NUMS\_VENDOR\_ID, NVM\_EMU\_MNG\_NUM\_MSIX, NVM\_EMU\_MNG\_NUM\_VF\_MSIX, VIRTIO\_BLK\_EMULATION\_NUM\_MSIX, VIRTIO\_P5\_EMULATION\_NUM\_MSIX, VIRTIO\_NET\_EMULATION\_NUM\_MSIX, PF\_NUM\_PF\_MSIX\_VALID, and PER\_PF\_NUM\_SF.
- DPU Operation Modes**: Lists host configurations.

At the bottom left, there's a collapse button and the text "Version: 2.1.2".

**Done.**

## Authors



### Boris Kovalev

Boris Kovalev has worked for the past several years as a Solutions Architect, focusing on NVIDIA Networking/Mellanox technology, and is responsible for complex machine learning, Big Data and advanced VMware-based cloud research and design. Boris previously spent more than 20 years as a senior consultant and solutions architect at multiple companies, most recently at VMware. He has written multiple reference designs covering VMware, machine learning, Kubernetes, and container solutions which are available at the NVIDIA Documents website.

**NVIDIA, the NVIDIA logo, and BlueField 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. <sup>TM</sup>**

**© 2025 NVIDIA Corporation. All rights reserved.**

## 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.

Last updated on Jan 19, 2026

---

**Corporate Info**

[NVIDIA.com Home](#)

[About NVIDIA](#)

---

**NVIDIA Developer**

[Developer Home](#)

[Blog](#)

---

**Resources**

[Contact Us](#)

[Developer Program](#)

[Privacy Policy](#) | [Your Privacy Choices](#) | [Terms of Service](#) | [Accessibility](#) | [Corporate Policies](#) | [Product Security](#) | [Contact](#)

Copyright © 2026 NVIDIA Corporation