Data Center / Cloud

Generative AI / LLMs

Robotics

Content Creation / Rendering

Data Science

Networking

Simulation / Modeling / Design

Conversational AI

Forums

Sign In

Searc

NVIDIA Doc king Solutions › RDG for DPF Zero Trust (

On Thi

RDG for DPF Zero Trust (DPF-ZT) with Argus DPU service

Created on Sep 15, 2025

Updated on Jan 18 2026 (v25.10 GA)

Scope

This Reference Deployment Guide (RDG) provides comprehensive instructions for deploying the NVIDIA DOCA Platform Framework (DPF) with the DOCA Argus service on high-performance, bare-metal infrastructure in **Zero-Trust mode**. It focuses on the setup and use of DPU-based services on **NVIDIA® BlueField®-3 DPUs** to deliver secure, isolated, and hardware-accelerated environments.

The guide is intended for experienced system administrators, systems engineers, and solution architects who build highly secure bare-metal environments 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 Argus Service 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	NFS	Network File System
DOCA	Data Center Infrastructure-on-a-Chip Architecture	OOB	Out-of-Band
DPF	DOCA Platform Framework	OVN	Open Virtual Network
DPU	Data Processing Unit	PF	Physical Function

Term	Definition	Term	Definition
K8S	Kubernetes	RDG	Reference Deployment Guide
KVM	Kernel-based Virtual Machine	RDMA	Remote Direct Memory Access
MAAS	Metal as a Service	RoCE	RDMA over Converged Ethernet
MTU	Maximum Transmission Unit	VPC	Virtual Private Cloud
NGC	NVIDIA GPU Cloud	ZT	Zero Trust

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.

One such service is the **DOCA Argus** Service provides Workload Threat Detection is a novel approach for container threat detection in AI workloads and microservices, utilizing a Bluefield DPU to perform live machine introspection at the hardware level. This approach analyzes specific snippets of volatile memory to provide real-time visibility into container activity and behavior at the network, host, and application levels.

The state of container node images is continuously monitored in real-time, checking for deviations from their secure, compliant versions and configurations to detect and stop runtime attacks. These insights also include the ability to identify attacks targeting network facing applications/services.

The Argus service provides events and data on any object on the OS (host/VM) without any configuration needed and without any active part from the user or the

host.

Examples what Argus service provides:

- Any new processes with its PID, name, attributes, and status.
- Reverse shells with process and network connection details such as source & destination IP and number of transferred bytes.
- SHA256 hash of running executable and loaded libraries

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 extends the capabilities of the DPF-managed Kubernetes cluster described in the **RDG for DPF Zero Trust (DPF-ZT) - NVIDIA Docs** (referred to as the **Baseline RDG**) by deploying the **NVIDIA DOCA Argus Service** within the existing DPF deployment to achieve a comprehensive, accelerated infrastructure.

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

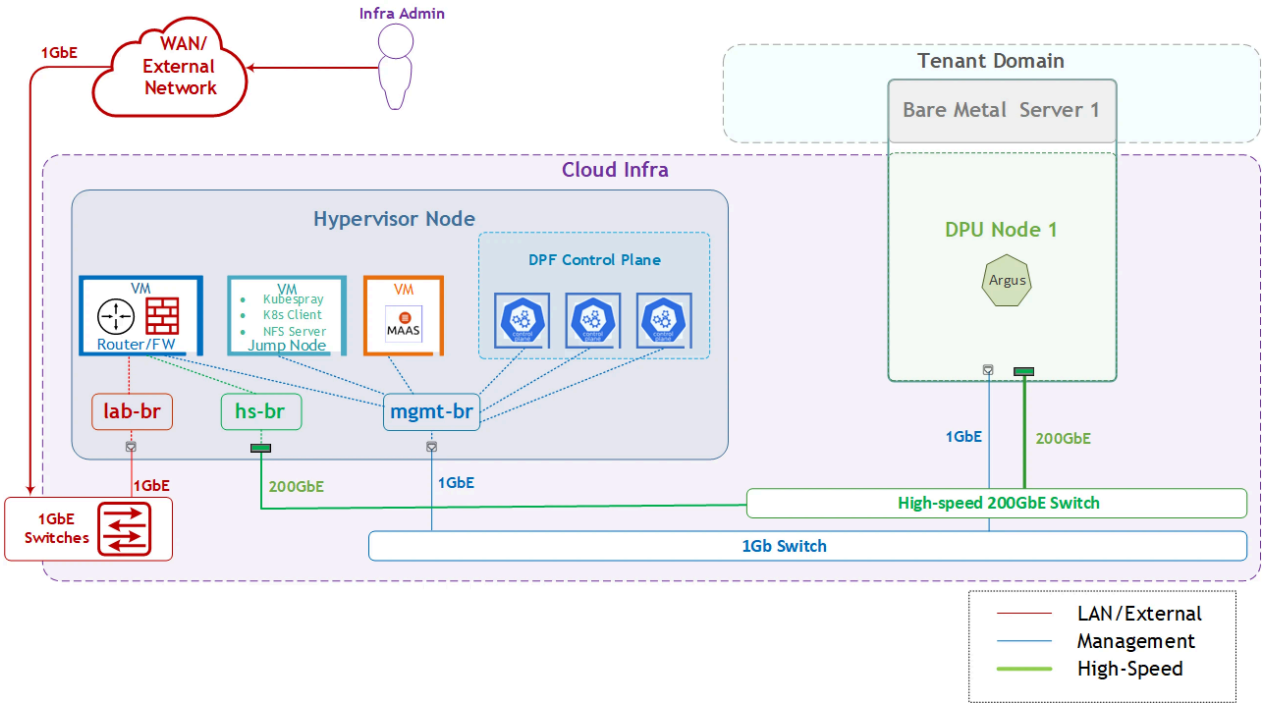
Solution Design

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



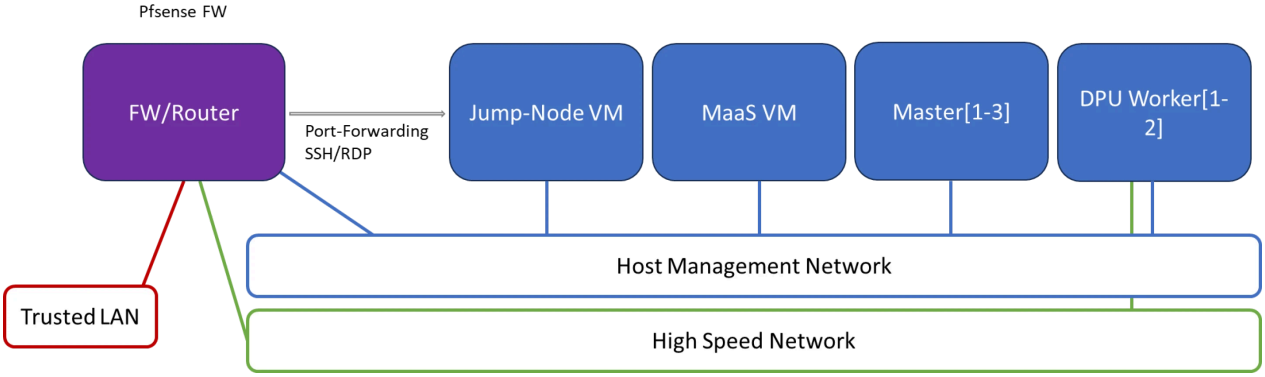
Firewall Design

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

Kubernetes	K8s 1.32.8
	NVIDIA DPF Operator v25.10.0
Host OS	Ubuntu 24.04 Server
	Kernel 6.8.0-53
BlueField-3 DPU	BFB: bf-bundle-3.2.0-107_25.10_ubuntu-22.04_prod
	Firmware: 32.46.1006
Spectrum Ethernet Switch	NVIDIA Cumulus Linux v5.13.1

ⓘ 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

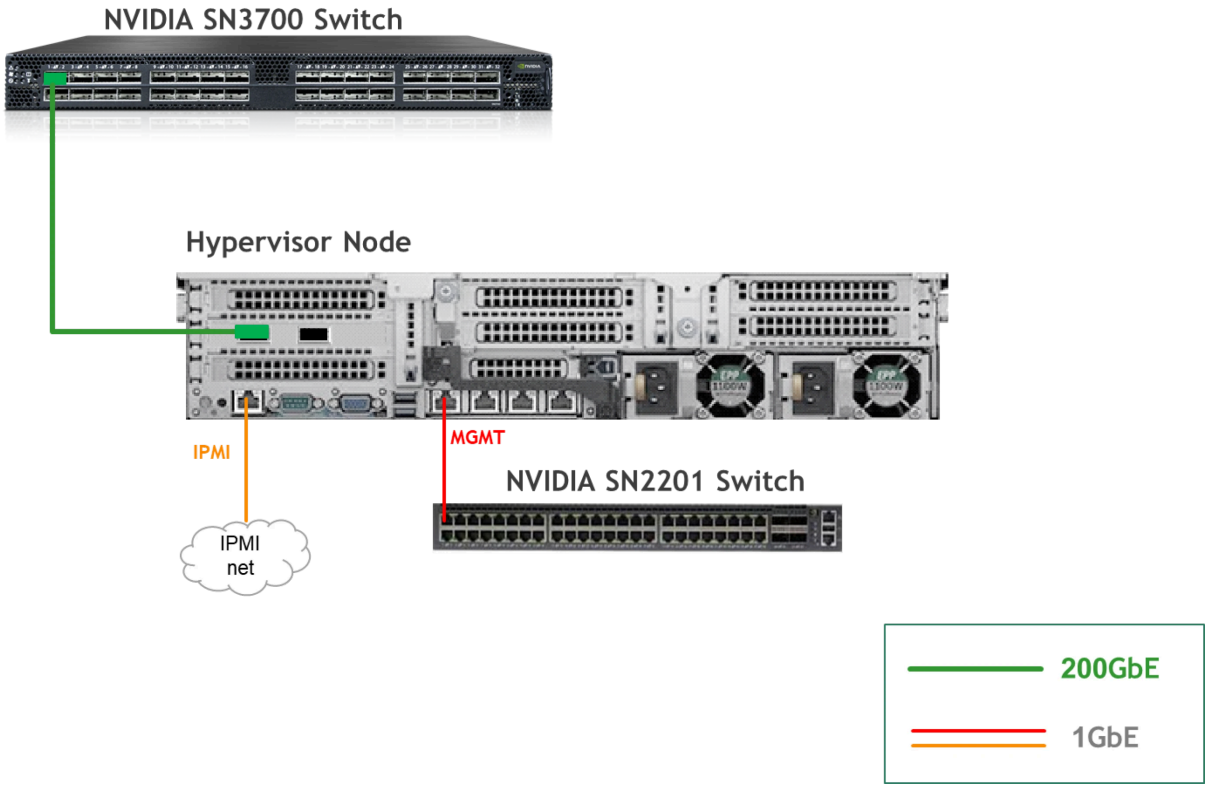
Expand

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	

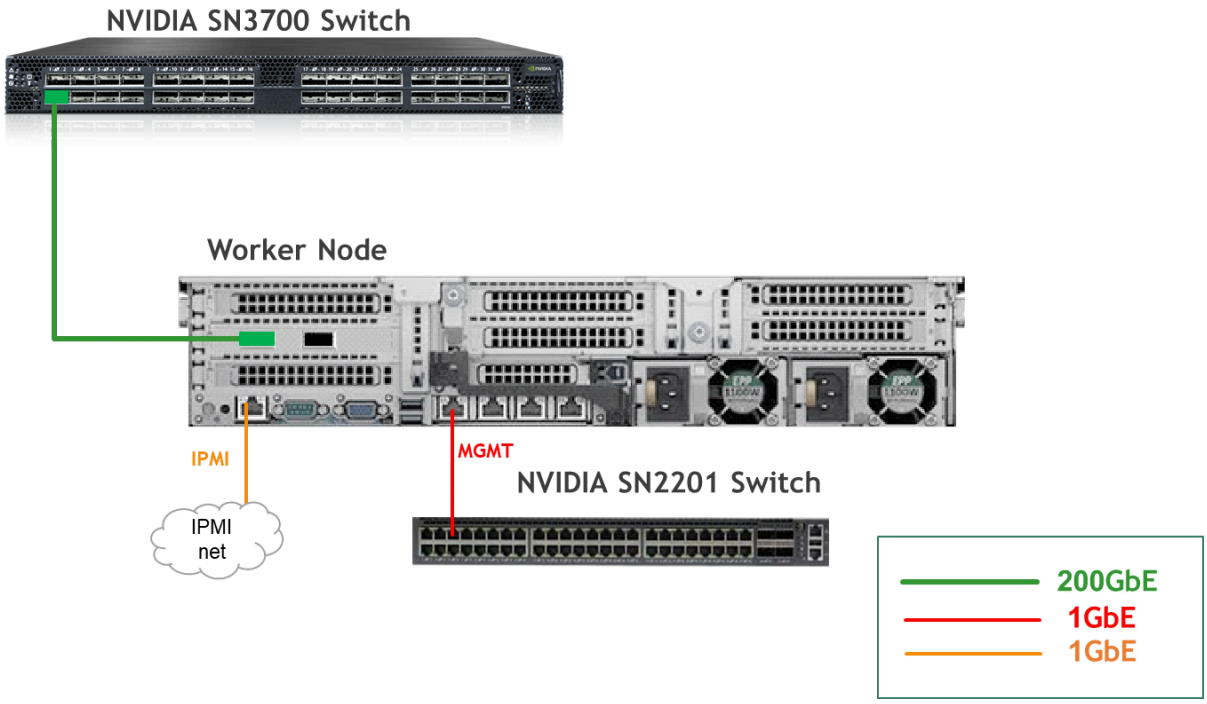
Hosts				
				OPT1(hs-br): 10.0.123.254/22
Rack1	Jump Node (Virtual)	`jump`	-	**enp1s0:** 10.0.110.253/24
Rack1	MaaS (Virtual)	`maas`	-	**enp1s0:** 10.0.110.252/24
Rack1	Master Node (Virtual)	`master1`	-	**enp1s0:** 10.0.110.1/24
Rack1	Master Node	`master2`	-	**enp1s0:** 10.0.110.2/24

Wiring

Hypervisor Node



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:

SN3700 Switch Console — Collapse Source

```
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:

SN2201 Switch Console — Collapse Source

```
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**.

Jump Node Console — Collapse Source

```
## Connect to master1 to copy helm client utility that was install
$ 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**:

Jump Node Console — Collapse Source

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

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

Jump Node Console — Collapse Source

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

4. Change directory to **readme.md** from where all the commands will be run:

Jump Node Console — Collapse Source

```
$ cd doca-platform/dpuservices/argus/
```

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.

Jump Node Console — Collapse Source

```
$ ssh root@10.0.110.201
root@10.0.110.201's password: <BMC Root Password. Default root/0pe
```

6. Modify the variables in `manifests/00-env-vars/argus_vars.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

manifests/00-env-vars/argus_vars.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
## allocated by DHCP.
export DPUCLUSTER_VIP=10.0.110.200

## Interface on which the DPUCluster load balancer will listen. SH
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 p
export HELM_REGISTRY_REPO_URL=https://helm.ngc.nvidia.com/nvidia/d

## The DPF REGISTRY is the Helm repository URL where the DPF Opera
## Usually this is the NVIDIA Helm NGC registry. For development p
export REGISTRY=https://helm.ngc.nvidia.com/nvidia/doca
## The DPF TAG is the version of the DPF components which will be
export TAG=v25.10.0
## URL to the BFB used in the `bfb.yaml` and linked by the DPUSet.
export BFB_URL="https://content.mellanox.com/BlueField/BFBs/Ubuntu

## IP_RANGE_START and IP_RANGE_END
## These define the IP range for DPU discovery via Redfish/BMC int
## Example: If your DPUs have BMC IPs in range 10.0.110.201-224
## export IP_RANGE_START=10.0.110.201
## export IP_RANGE_END=10.0.110.224

## Start of DPUDiscovery IpRange
```

```
export IP_RANGE_START=10.0.110.201

## End of DPUDiscovery IpRange
export IP_RANGE_END=10.0.110.204

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

7. Export environment variables for the installation:

Jump Node Console — Collapse Source

```
$ source argus_vars.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 `doca-platform/dpuservices/argus/` directory, a few adjustments are required.

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

bfb.yaml — Collapse Source

```
---
apiVersion: provisioning.dpu.nvidia.com/v1alpha1
kind: BFB
metadata:
  name: bf-bundle
  namespace: dpf-operator-system
spec:
  url: $BFB_URL
```

2. Run the command to create the `BFB`:

Jump Node Console — Collapse Source

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

3. Change the **DPUFlavor** using the following YAML:

Expand

DPUFlavor.yaml — Collapse Source

```
---
apiVersion: provisioning.dpu.nvidia.com/v1alpha1
kind: DPUFlavor
metadata:
  name: dpf-provisioning-argus
  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=pl011,0x13010000
      - fixrttc
      - net.ifnames=0
      - biosdevname=0
      - iommu.passthrough=1
      - cgroup_no_v1=net_prio,net_cls
```

4. Change the `DPUDeployment.yaml` file:

DPUDeployment.yaml — Collapse Source

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUDeployment
metadata:
  name: argus
  namespace: dpf-operator-system
spec:
  dpus:
    bfb: bf-bundle
    dpuSets:
    - nameSuffix: dpuset-argus
      nodeSelector:
        matchLabels:
          feature.node.kubernetes.io/dpu-enabled: "true"
    flavor: dpf-provisioning-argus
    nodeEffect:
      hold: true
```

```
  serviceChains:
    switches:
    - ports:
      - serviceInterface:
          matchLabels:
            uplink: p0
    upgradePolicy:
      applyNodeEffect: true
  services:
    argus:
      serviceConfiguration: argus
      serviceTemplate: argus
```

✓ 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**.

5. Change the `DPUServiceConfiguration.yaml` file:

DPUServiceConfiguration.yaml — Collapse Source

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceConfiguration
metadata:
  name: argus
  namespace: dpf-operator-system
spec:
  deploymentServiceName: argus
  serviceConfiguration:
    helmChart:
      values:
        config:
          isLocalPath: false
        containerImage: $ARGUS_NGC_IMAGE_URL
```

6. Change the `DPUServiceTemplate.yaml` file:

DPUServiceTemplate.yaml Collapse Source

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceTemplate
metadata:
  name: argus
  namespace: dpf-operator-system
spec:
  deploymentServiceName: argus
  helmChart:
    source:
      chart: doca-argus
      repoURL: $HELM_REGISTRY_REPO_URL
      version: 1.0.0
```

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

Jump Node Console Collapse Source

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

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

Jump Node Console Collapse Source

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

9. Wait for the **NodeEffect stage** (at this point the provisioning is paused, waintig for external signal).
   Run following command on all/specific DPU nodemaintanace object/s to proceed with provisioning:

Jump Node Console Collapse Source

```
$ kubectl annotate dpunodemaintenances -n dpf-operator-system --al
```

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

Jump Node Console Collapse Source

```
$ 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-mt233
    Last Transition Time:  2026-01-12T13:57:52Z
    Type:                  BFBPrepared
    Last Transition Time:  2026-01-12T14:02:27Z
    Type:                  BFBTransferred
    Last Transition Time:  2026-01-12T13:57:52Z
    Type:                  FWConfigured
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  InterfaceInitialized
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  NodeEffectReady
    Last Transition Time:  2026-01-12T14:36:31Z
    Reason:                OemLastState
    Type:                  OSInstalled
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  BFBReady
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  Initialized
    Last Transition Time:  2026-01-12T14:39:31Z
    Type:                  Rebooted
  Phase:               Rebooting
```

11. 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 dpunode --all provisioni
```

12. 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 '
Every 10.0s: kubectl describe dpu -n dpf-operator-system | grep 'N

  Dpu Node Name:                                  dpu-node-mt233
    Type:       InternalIP
    Type:       Hostname
    Last Transition Time:  2026-01-12T15:29:52Z
    Type:                  Ready
    Last Transition Time:  2026-01-12T13:57:52Z
    Type:                  BFBPrepared
    Last Transition Time:  2026-01-12T14:02:27Z
    Type:                  BFBTransferred
    Last Transition Time:  2026-01-12T15:29:52Z
    Type:                  DPUClusterReady
    Last Transition Time:  2026-01-12T13:57:52Z
    Type:                  FWConfigured
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  InterfaceInitialized
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  NodeEffectReady
    Last Transition Time:  2026-01-12T15:29:52Z
    Type:                  NodeEffectRemoved
    Last Transition Time:  2026-01-12T14:36:31Z
    Reason:                OemLastState
    Type:                  OSInstalled
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  BFBReady
    Last Transition Time:  2026-01-12T13:57:51Z
    Type:                  Initialized
    Last Transition Time:  2026-01-12T15:29:52Z
    Type:                  Rebooted
  Phase:               Ready
```

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

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/c

$ dpfctl describe dpudeployments
NAME                                  NAMESPACE            STATUS
....
└─DPUDeployments
  └─DPUDeployment/argus                                dpf-operator-s
    ├─DPUServiceChains
    │ └─DPUServiceChain/argus-kjbb2                    dpf-operator-s
    ├─DPUSets
    │ └─DPUSet/argus-dpuset-argus                      dpf-operator-s
    │   ├─BFB/bf-bundle                                dpf-operator-s
    │   └─DPUs
    │     └─DPU/dpu-node-mt2402xz0f7x-mt2402xz0f7x     dpf-operator-s
    └─Services
      ├─DPUServiceTemplates
      │ └─DPUServiceTemplate/argus                     dpf-operator-s
      └─DPUServices
        └─DPUService/argus-76pxl                       dpf-operator-s


$ echo "alias ki='KUBECONFIG=/home/depuser/dpu-cluster.config kube
$ kubectl get secrets -n dpu-cplane-tenant1 dpu-cplane-tenant1-adm
$ ki get node -A
NAME                                  STATUS   ROLES    AGE   VERSI
dpu-node-mt2337xz04qz-mt2337xz04qz    Ready    <none>   46m   v1.34

$ kubectl get dpu -A
NAMESPACE             NAME                                  READY
dpf-operator-system   dpu-node-mt2337xz04qz-mt2337xz04qz    True

$ kubectl wait --for=condition=ready --namespace dpf-operator-syst
dpu.provisioning.dpu.nvidia.com/dpu-node-mt2402xz0f7x-mt2402xz0f7x
```

Verification

Here's a step-by-step procedure to check the DOCA Argus service on your NVIDIA BlueField DPU.

✓ Note

Ubuntu 24.04 was installed on the servers.

1. Open the first worker server console.

Jump Node Console — Collapse Source

```
$ ssh worker1
```

2. Add iommu configuration in the `/etc/default/grub` file:

First BM Server Console — Collapse Source

```
root@worker1:~# vim /etc/default/grub

## Add iommu=pt intel_iommu=on in GRUB_CMDLINE_LINUX_DEFAULT param

GRUB_CMDLINE_LINUX_DEFAULT="iommu.passthrough=1 intel_iommu=on"
```

3. Reboot the server.

Second BM Server Console — Collapse Source

```
root@worker1:~# reboot
```

4. For test we will run the ***sleep 100*** command.

Collapse Source

Second BM Server Console

```
root@worker1:~# sleep 100&
```

5. C onnect to the first DPU `OOB` over SSH and change the `OOB` ubuntu's user password(d efault password is **ubuntu**).

DPU BM Server Console — Collapse Source

```
root@worker1:~# ssh ubuntu@10.0.110.211
```

6. Run following command to see Argus log events about the `sleep 100` process on the worker host.

Expand

DPU BM Server Console — Collapse Source

```
ubuntu@dpu-node-mt2402xz0f7x-mt2402xz0f7x:~$ jq 'select(.activity_

{
  "name": "process_created",
  "process_details": {
    "process_id": "2089",
    "process_name": "sleep",
    "process_file_name": "sleep",
    "process_self_exec_id": "8",
    "process_parent_process_id": "2047",
    "process_cpu_clock_cycles": "2082047",
    "process_real_group_id": "1000",
    "process_real_user_id": "1000",
    "process_command_line_arguments": "sleep 100",
    "process_state": "RUNNING",
    "process_pid_namespace": "4026531836",
    "process_mount_points_namespace": "4026531841",
    "process_network_namespace": "4026531840",
    "process_hash_sha256": "4a193eb6f25eecf27bad523cb8a53ec4d4077
    "process_hash_sha1": "bab62b22ddb568b245ebc0132200a5e2ddd8577
```

```
      "process_hash_md5": "ecdb9cd1468ff7151564b334b73161f5",
      "process_file_size_bytes": "35336",
      "process_folder_path": "/usr/bin/",
      "process_creation_time_iso_8601_ns": "2025-09-15T13:58:35.624
      "process_container_id": ""
    }
  }
  {
    "name": "thread_created",
    "process_details": {
      "process_id": "2089",
      "process_name": "sleep",
      "process_file_name": "sleep",
      "process_self_exec_id": "8",
      "process_parent_process_id": "2047",
      "process_cpu_clock_cycles": "2082047",
      "process_real_group_id": "1000",
      "process_real_user_id": "1000",
      "process_command_line_arguments": "sleep 100",
      "process_state": "RUNNING"
```


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.

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 Sep 16, 2025