DGX B200 · GPU FABRIC OPERATIONS

Making DGX B200 RDMA-ready.

InfiniBand, SR-IOV, RDMA, and Kubernetes GPU networking lessons from an 8-node DGX B200 cluster.

8 DGX B200
cluster scale

400G IB
scale-out fabric

K8s + RDMA
workload plane

github.com/ziwon
ai platform architect

STORYLINE

Operational flow for GPU fabric

01	**Why different**	Collective traffic exposes every weak link.	02	**DGX topology**	Scale-up NVLink, scale-out IB rails.
03	**K8s + SR-IOV**	NAD, Device Plugin, kubelet, CNI.	04	**Failure modes**	Visible VF, missing RDMA state.
05	**Ownership**	Host manager vs Kubernetes CNI.	06	**Takeaways**	RDMA-ready criteria for DGX B200.

OPERATIONAL THESIS

AI networking is the job path

The network path is part of the GPU platform.

Ping and Pod IP are not enough. RDMA-ready means GPU locality, PF/VF mapping, GUID/LID, Subnet Manager state, and Kubernetes allocation all line up.

8 nodes	**8** rails	**400** Gbps	**5** checks
DGX B200 cluster	per DGX network shape	scale-out IB target	RDMA-ready VF state
64 B200 GPUs	GPU-local paths	NDR-class fabric	GUID · LID · SM

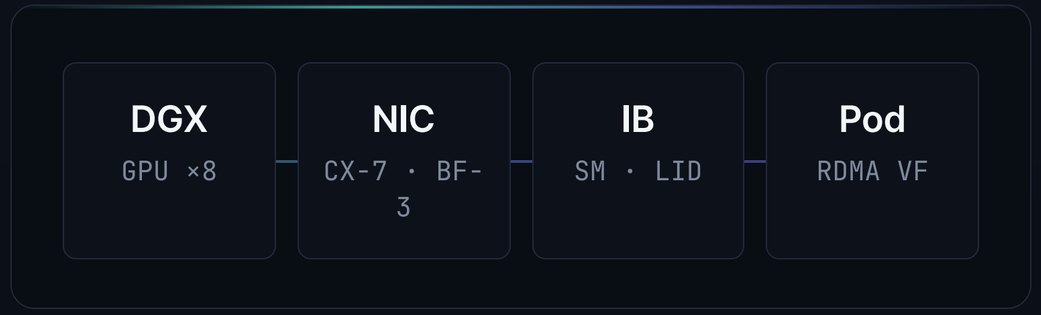
The work is not buying GPUs. It is opening a reliable path between them.

CLUSTER SHAPE

DGX B200 is topology, not port count

LAYER	ROLE IN THE FABRIC
Compute	8 DGX B200 nodes, 8 B200 GPUs per node
Scale-up	NVLink / NVSwitch inside each node
Scale-out	ConnectX-7 InfiniBand / Ethernet paths
Switching	QM9700 or QM8700-class IB fabric
Kubernetes	Cilium + Multus + SR-IOV CNI + Device Plugin

SYSTEM VIEW



— **GPU locality** and RDMA VF locality

— **OSFP, PCI bus, PF/VF, mlx5_x** mapping

— **Topology-aware scheduling** as a performance guardrail

PATH MODEL

One fabric, end to end

01	**GPU**	B200 GPU, NCCL collective traffic, local NVLink domain	`compute`
02	**PCIe / NVSwitch**	GPU affinity and host topology decide the nearest network path	`locality`
03	**ConnectX-7 / BF-3**	PF, VF, RDMA device, firmware, driver binding	`mlx5`
04	**InfiniBand fabric**	Subnet Manager, GUID, LID, link width/speed, switch counters	`fabric`
05	**Kubernetes**	Device Plugin, Multus, SR-IOV CNI, NAD, kubelet allocation	`scheduler`
06	**Pod / framework**	ibv_devinfo, NCCL logs, PyTorch distributed job behavior	`workload`

RAIL-OPTIMIZED DESIGN

Rail locality matters

1:1
GPU-to-rail thinking

PF/VF
preserve locality

NCCL
exposes weak links

KUBERNETES + RDMA

Kubernetes does not infer fabric

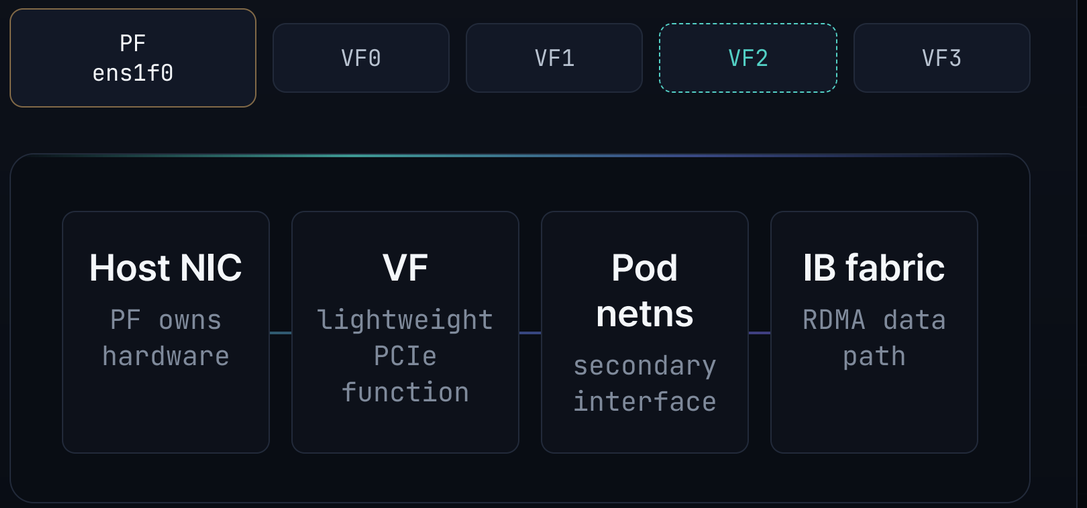
FAILURE PATTERN

`nvidia.com/gpu` can be correct while the RDMA VF lands on the wrong NIC. The job may run, but performance will drift.

OPERATOR VIEW

- NAD, resourceName, kubelet allocation
- Pod visibility vs fabric visibility
- Topology Manager and scheduler policy

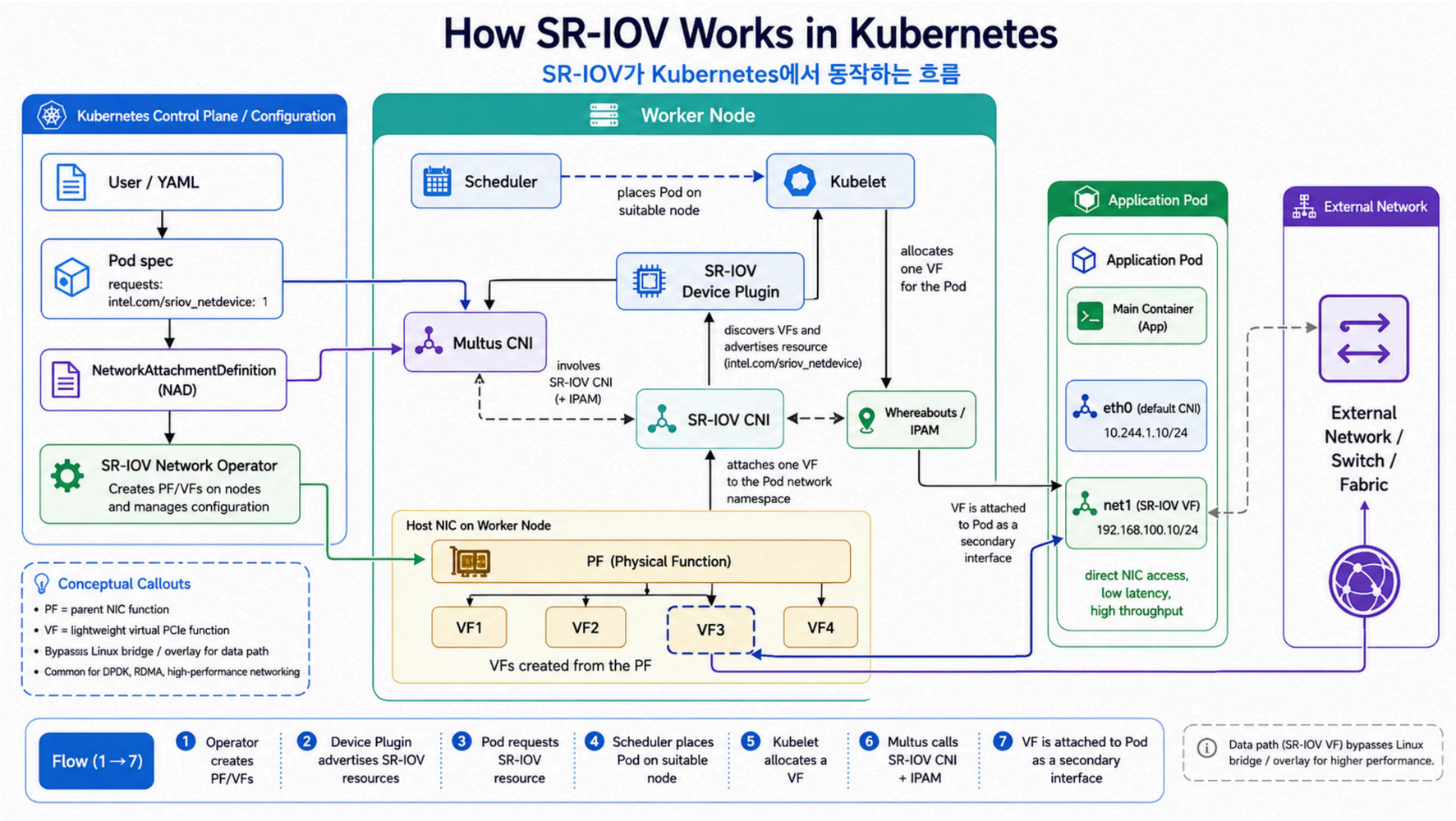
SR-IOV assigns a PCI function



CORE TERMS

- **PF**: physical parent function and VF policy
- **VF**: lightweight PCIe function passed to the workload
- **SR-IOV CNI**: moves the selected VF into the Pod namespace
- **Device Plugin**: advertises VF resources to kubelet
- **Multus**: attaches the secondary network

VF allocation flow



WHAT TO VERIFY

Attachment vs allocation

NETWORK ATTACHMENT PATH

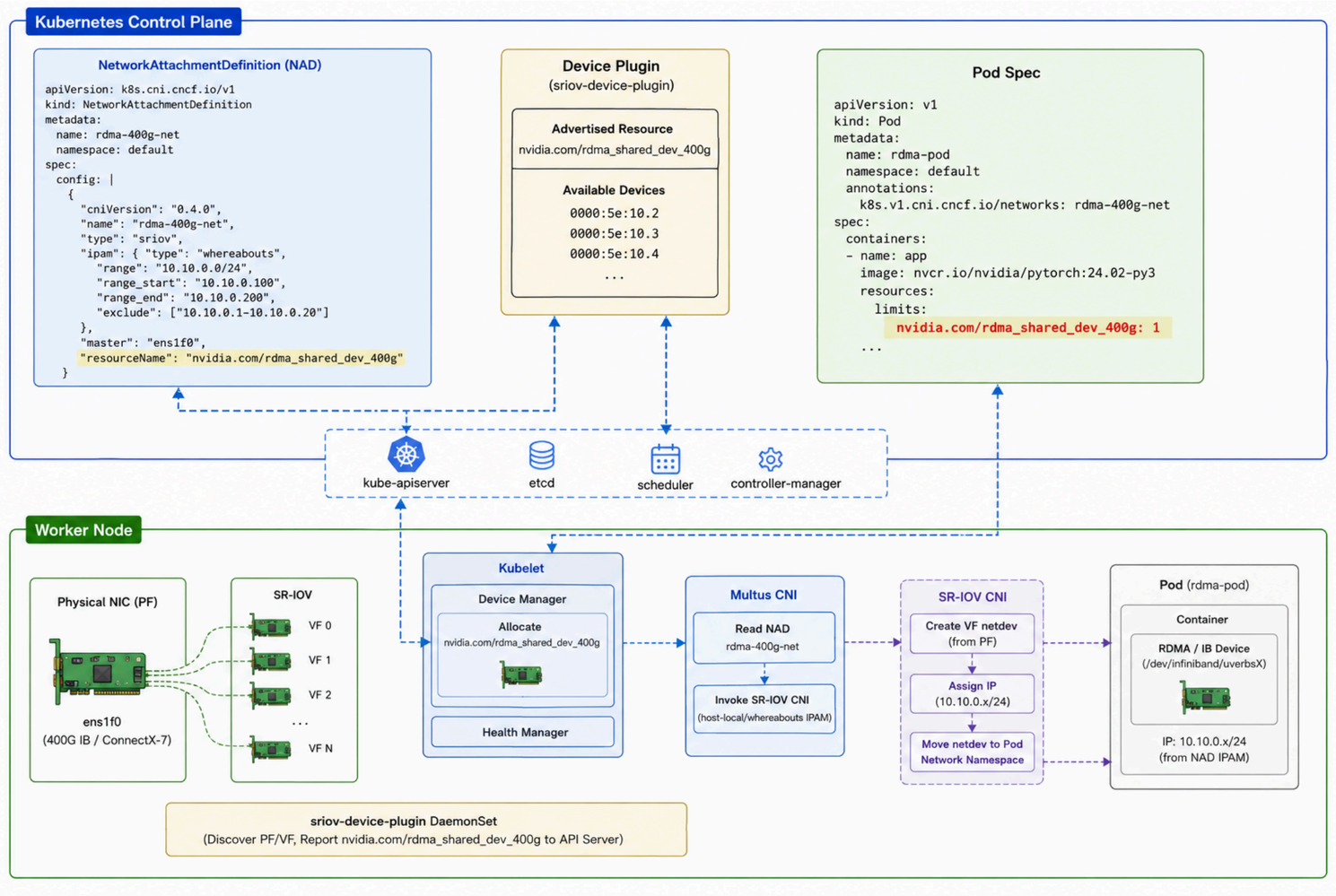
- **NAD**: secondary network definition
- **Multus**: annotation parsing and CNI call
- **SR-IOV CNI**: VF move and IPAM handoff

DEVICE ALLOCATION PATH

- **Device Plugin**: extended resource advertisement
- **Scheduler**: node placement by resource
- **Kubelet Device Manager**: concrete PCI assignment

DGX B200 / IB view: Pod VF visibility still needs RDMA device, GUID, LID, SM, and `mlx5_ib` alignment.

Resource name alignment



OPERATIONAL GUARDRAILS

Four SR-IOV guardrails

- **Network purpose split.** `eth0`: service/control, `net1`: RDMA path
- **Multus is an attachment caller.** It does not complete the fabric.
- **Whereabouts is IPAM.** It does not fix LID/GUID/SM state.
- **Device Plugin creates scheduling resources.** CNI follows kubelet allocation.
- **Visible VF is not RDMA-ready.** Check `ibv_devinfo`, LID, GUID, driver binding.
- **Locality still matters.** Match GPU, rail, and local NIC.

Summary: NAD chooses the network. Device Plugin reserves the VF. Kubelet and CNI move it into the Pod.

TROUBLESHOOTING CASE

Visible VF ≠ usable RDMA

pod@training-job:~

```
$ lspci | grep Mellanox
VF present
$ kubectl describe node | grep rdma
resource exposed
$ ibv_devinfo
no active LID / unusable RDMA path
# device exists, fabric membership is incomplete
```

CHECK	WHY IT MATTERS
GUID	Device identity on the InfiniBand fabric.
LID	Fabric address assigned by the Subnet Manager.
SM state	Subnet Manager: the IB control plane that admits endpoints.
mlx5_ib	Linux RDMA driver for NVIDIA/Mellanox NICs.
PF/VF mapping	Wrong port mapping creates false positives.

KEY LESSON

VF Visible ≠ RDMA Ready

Required path before distributed training

GUID → LID → mlx5_ib → Fabric Membership → Topology Validation

OBSERVABILITY CHECKLIST

Observe beyond GPU

NODE / GPU

- nvidia-smi topo -m
- mst status
- devlink dev show

INFINIBAND / RDMA

- ibstat
- ibv_devinfo
- iblinkinfo
- perfquery

KUBERNETES

- kubectl describe node
- kubectl get network-attachment-definitions
- kubectl logs ds/sriov-device-plugin

GPU

utilization and NCCL behavior

NIC

RDMA throughput and counters

IB

LID/GUID and switch health

Pod

actual device visibility

CASE STUDY

VF ownership conflict

FAILURE SETUP

- New SR-IOV VF appears on the host as `ibp*`
- Multus + SR-IOV CNI tries to move it into the Pod namespace
- `systemd-networkd` claims the VF lifecycle first

CORE MESSAGE

Host visibility is required. Host management is not.

Discovery belongs to the host. Ownership belongs to Multus, SR-IOV CNI, and IPAM.

FAILURE FLOW

Host owns VF, Pod loses `net1`

vf-lifecycle.trace

```
SR-IOV VF created
↓
udev event emitted
↓
systemd-networkd detects VF
↓
per-VF network unit activates
↓
Host OS claims VF
↓
Multus / SR-IOV CNI tries Pod move
↓
IPAM / netns handoff fails
↓
Pod net1 gets no VF IP
```

OBSERVED SYMPTOMS

- VF visible on the host
- Pod secondary interface not stable
- `Whereabouts` / IPAM does not assign the expected IP
- Pod VF IP succeeds after disabling the host unit

Root cause: VF lifecycle ownership conflict, not missing hardware.

RESOLUTION

Keep VFs unmanaged on host

OPERATOR FIX

- Exclude new SR-IOV VFs from host-side network managers
- Block automatic management for `ibp*` and VF interfaces
- Keep VFs visible but unmanaged on the host
- Let Multus + SR-IOV CNI + IPAM own the final move

STATE	RESULT
Host owns VF	Pod IP assignment fails
CNI owns VF	Pod secondary interface works

Lifecycle rule: High-performance networking includes device lifecycle ownership, not just bandwidth.

CLOSING MESSAGES

RDMA-ready is an end-to-end state

FABRIC STATE

GUID · LID · SM · `mlx5_ib`

DEVICE OWNERSHIP

VF visible, host unmanaged, CNI-owned move

TOPOLOGY FIT

GPU path · rail · local NIC

Rule: if any layer disagrees, the VF is not ready.

FURTHER READING

Implementation sources and vendor docs

NVIDIA K8s on Bare Metal - Ethernet

Bare-metal Kubernetes, SR-IOV, RDMA/RoCE, and NVIDIA NICs.

NVIDIA DGX B200 User Guide

DGX B200 system overview, GPU, NVLink/NVSwitch, and networking hardware.

Introduction to NVIDIA DGX B200 Systems

Component descriptions for DGX B200 compute, storage, and network cards.

NVIDIA Network Operator

Kubernetes RDMA, GPUDirect, SR-IOV, and NVIDIA networking components.

NVIDIA DOCA: Kubernetes Using SR-IOV

SR-IOV CNI and RDMA device plugin concepts for Kubernetes Pods.

Multus CNI

Secondary network attachment and multi-homed Pods in Kubernetes.

SR-IOV Network Device Plugin

Advertising and allocating SR-IOV VF resources through Kubernetes.

NVIDIA GPUDirect RDMA User Manual

Direct RDMA data paths between NVIDIA GPUs and network adapters.

github.com/ziwon

AI Platform Architect

Building reliable infrastructure for AI systems, from GPU fabrics to agentic workflows.

AI Infrastructure

Data Center Networking

LLMOps & Model Serving

Agentic AI

The horizon keeps moving.
That's why I'm still learning, still building.