Container Networking Architecture for AI Workloads

Kubernetes + GPU integration

Akash Patel

TABLE OF CONTENTS

OBJECTIVES & NON-GOALS

- **Objectives:** Low-jitter, high-throughput east-west for distributed training; simple app ingress/egress; clean multi-tenant boundaries; debuggable; upgradeable without mass downtime.
- **Non-Goals:** For training traffic, do **not** rely on service mesh or L7 proxies; avoid unnecessary overlays in the hot path.

KUBERNETES SOLUTION OVERVIEW

LOGICAL SEPARATION

- Frontend (FE) fabric: Primary CNI on eth0. Hosts Ingress/Gateway, UIs/APIs, dashboards. Internet/DMZ edge sits here.
- Backend fabric: Dual-rail RoCE/IB for NCCL (inter-GPU). Multus SR-IOV VFs (net1/net2) to pods.
- Storage fabric: RoCE/IB for NVMe-oF / parallel FS. Optional VFs (net3/net4) to pods.

NODE TYPES

CONTROL PLANE PODS

System components running on the controller nodes, scheduled as **static pods** (not normal deployments); it runs :

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- etcd
- Sometimes extra add-ons like the CNI operator, admission webhooks, metrics-server.

Placement

- Not on GPU compute nodes.
- Typically:
  - A small set of dedicated x86 servers (3, 5, or 7 nodes for HA).
  - Dual-NIC, usually on a management / frontend network only — they don't need backend RDMA or storage rails.
  - Often VMs on an infra cluster (e.g., OpenShift management cluster, bare-metal K8s management nodes, or even a cloud-managed
  - /etc/kubernetes/manifests on control plane node contains the pod manifests.
  - These are **not** affected by GPU job scheduling.

WORKER PODS

These are training pods, operators, and monitoring agents — scheduled on GPU nodes

Run kubelet, CNI agents, SR-IOV device plugin, NVIDIA GPU Operator pods, logging/metrics agents, and training pods.

Have multiple physical NICs:

- Frontend (100G) — primary CNI, connects to control plane.
- Backend rails (2×400G) — RDMA/NCCL only, no control plane traffic.
- Storage rails (2×400G) — NVMe-oF or FS client, no control plane traffic.

The kubelet only registers the frontend IP with the API server; backend/storage networks are invisible to Kubernetes unless you explicitly add them via Multus.

CONTAINER TYPES

MAIN CONTAINER

What it is:

- The primary workload inside a pod — the reason the pod exists.
- Runs for the entire lifetime of the pod (from start until termination).

For training Pod

- This is the training process container (PyTorch, DeepSpeed, Horovod, TensorFlow, etc.).
- Owns the GPUs (e.g., nvidia.com/gpu: 8) and the high-speed Multus SR-IOV interfaces (net1, net2, …).
- Handles NCCL/UCX communication, data loading, checkpoint writing.

Key points:

- There can be more than one main container in a pod, but in HPC/AI training we almost always use exactly one for performance and NUMA affinity reasons.

INIT CONTAINER

What it is:

- A special container that runs before the main container starts.
- Runs sequentially (one after the other, not in parallel).
- Always completes and exits before the main container is launched.

For Training Pod:

- Prepares the environment for training:
  - Generates rankfiles/hostfiles with the backend interface IPs from Multus (net1, net2).
  - Pre-downloads or stages datasets to local NVMe to avoid slow startup.
  - Sets sysctl parameters for RDMA or disables rp_filter.
- Does not need GPUs or high bandwidth for long; runs briefly.

Key points:

- They have their own filesystem layer but can share volumes with the main container to pass data.
- Great for pre-flight checks and **one-time setup** without baking that logic into your training image.

SIDECAR CONTAINER

What it is:

- A container that runs **in parallel** with the main container for the lifetime of the pod.
- Shares the same network namespace (and optionally volumes) as the main container.

For Training Pod

- Usually **avoided** in GPU training pods because sidecars:
  - Consume CPU/memory.
  - Can interfere with RDMA performance if misconfigured.
  - Make shutdown/startup sequencing harder at 10k+ pod scale.
- If used at all, it's for:

- Lightweight **metrics exporters** (Prometheus exporters, DCGM in-container).
- Log shippers (FluentBit).
- Debugging agents (temporary).

Key points:

- Unlike init containers, they don't block the main container from starting
- Unlike main containers, they're not the primary workload

SR-IOV

NIC HARDWARE

- What **it is:** A PCIe feature in the NIC that allows a single physical function (PF) to expose multiple lightweight "virtual functions" (VFs).
- Why **it matters for RDMA/RoCE:**
  - Each VF can present itself as a separate RDMA-capable network interface.
  - You can pass a VF directly into a container/pod so the RDMA traffic bypasses the host kernel datapath (zero-copy, GPU Direct RDMA works).
  - Without SR-IOV, the pod's traffic would have to go through the host networking stack (extra latency, no direct NIC queue mapping).

SR-IOV *CNI* (THE KUBERNETES INTEGRATION)

- **What it is:** A CNI plugin that tells Kubernetes *how* to attach a specific VF from the node into a pod's network namespace.
- **Why it matters:**
  - K8s by itself doesn't know anything about your NIC's SR-IOV capabilities.
  - SR-IOV CNI takes a VF that's already created/configured on the node and wires it into the pod as net1, net2, etc., using the Multus framework.
  - Works with **SR-IOV Device Plugin** to advertise RDMA-capable VFs as allocatable resources in K8s (rdma/hca_shared_a, etc.).

NETWORKING BETWEEN CONTROL PLANE & WORKERS

- Uses **frontend CNI network**:
  - kubelet ↔ kube-apiserver
  - CNI DaemonSets (e.g., Multus, SR-IOV device plugin) reporting status
- Control plane **never talks** over backend or storage rails.
- This separation ensures that **training congestion** doesn't break cluster management.

PRIMARY CNI (CONTROL/DATA PLANE FOR "NORMAL" PODS)

- Cilium (eBPF, kube-proxy replacement) or Calico (eBPF mode).
- K8s primary CNI = service discovery, control traffic, lightweight I/O

- Creates the default eth0 in every pod.
- Provide pod-to-pod, NetworkPolicies, services, and Frontend traffic.
  - Handles pod IPs, Services, DNS, Ingress/LoadBalancer, kube-proxy/eBPF, etc.
- Traffic goes through the Linux networking stack (conntrack/iptables or eBPF).
- Ideal for frontend/control/ordinary app traffic—not for RDMA hot paths.
- Optional: Cilium ClusterMesh if you run multiple clusters.

SECONDARY CNI VIA MULTUS (TRAINING DATA PATH & HOT STORAGE)

Attach a dedicated high-performance interface into selected pods:
Multus (meta-CNI) + SR-IOV CNI

- RoCEv2 path: sriov-cni to pass a VF from the ConnectX NIC; enable RDMA.
- Multus lets a pod have extra VF/interfaces (e.g., net1, net2 …) in addition to eth0.
  - VFs from the storage NICs (net3/net4) to mount high-throughput storage directly on to pod;
- SR-IOV CNI attaches virtual functions (VFs) of physical NICs directly into the pod for near bare-metal I/O (RDMA/DPDK possible).
- These extra interfaces typically bypass kube-proxy/conntrack and are used for Backend (NCCL/UCX) & Storage paths (NVMe-oF ROCEv2)

  Think: Primary CNI = Kubernetes features.
  Multus+SR-IOV = raw performance pipes you opt-in per pod

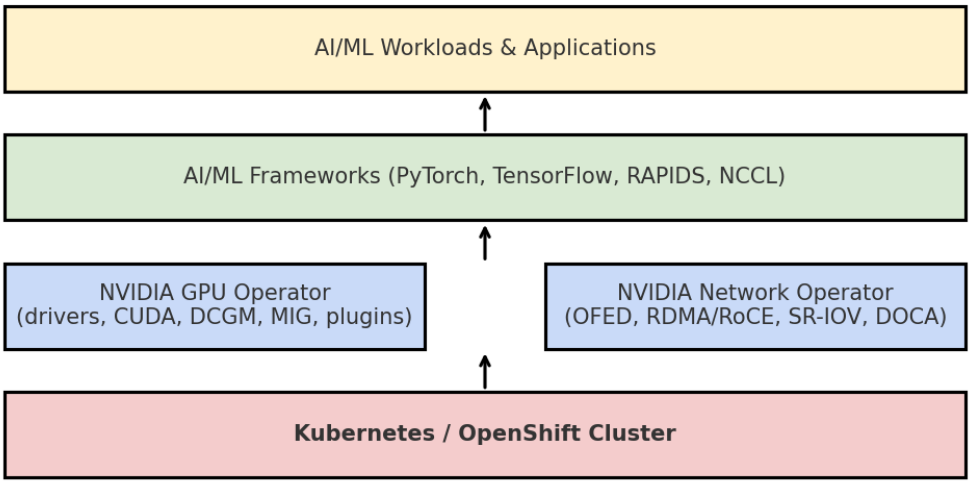
KUBERNETES NETWORKING

PRACTICAL WIRING

- **Pod interfaces:**
  - eth0 = primary CNI (Frontend network, 100G)
  - net1, net2 = Backend rails (SR-IOV VFs, 400G+400G)
  - net3, net4 = Storage rails (SR-IOV VFs, 400G+400G)
- **Env for NCCL/UCX:** limit training to backend rails
  - NCCL_SOCKET_IFNAME=net1,net2
  - NCCL_IB_HCA=mlx5_0,mlx5_1 (or UCX UCX_NET_DEVICES= …)
- **Routing hygiene:**
  - **Default route only on eth0 (Frontend).**
  - No default routes on net1-4; only specific /16s.
  - Disable strict rp_filter (reverse path)if you see asymmetric paths: net.ipv4.conf.all.rp_filter=0.
- **MTU:** 9000 on backend/storage rails end-to-end; 1500–9000 on Frontend per policy.
- **QoS:** backend and storage in separate lossless classes (RoCEv2); PFC+ECN with watchdog.

CORE COMPONENTS

- NVIDIA GPU Operator (installs driver, container toolkit, DCGM, MIG management).
- NVIDIA Network Operator *(for RoCE/IB)* to configure SR-IOV VFs, RDMA, net-qos, and K8s RDMA device resources.
- SR-IOV Device Plugin (advertises rdma/hca* resources).
- MOFED / RDMA-core aligned with NIC firmware. ** app (NCCL, MPI, NVMe-Of) use to drive RDMA verbs and talk to NIC using these libraries



NODE KERNEL & HOST SETTINGS (ROCEV2)

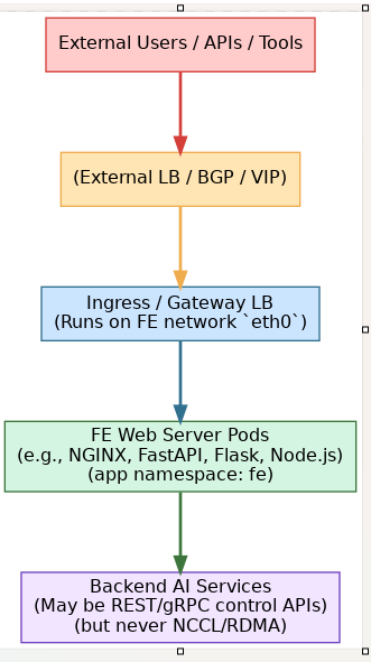
- MTU 9000 on VF/PF and fabric.
- ECN: net.ipv4.tcp_ecn=1 for control-plane TCP; UDP ECN handled by fabric.
- Flow control: PFC only on training class; disable global pause.

SERVICES & TRAFFIC PATTERNS

FRONTEND (FE) SERVICES:

OVERVIEW

This the flow for Frontend Fabric



Why is it recommended to use both: External LB and Ingress/Gateway LB?

1. Separation of concerns:
   - L4 gets traffic to the cluster reliably;
   - L7 applies application logic.
   - Each scales independently.

2. Resilience & scale:
   - L4 (VIP + BGP/ECMP) spreads connections across nodes; Lose a node? ECMP rebalances.
   - L7 scales by adding more ingress pods. Lose an ingress pod? Service routing sends to another.

3. Security & policy:
   - Put mTLS/JWT/WAF-like checks at L7 without exposing backend services directly to the outside.

4. Multi-tenancy/governance:
   - Platform team owns the L4/VIP surface; app teams manage L7 routes via Ingress/Gateway API CRDs.

EXTERNAL LB

These are various options for External LB to choose from

- BGP to ToR (MetalLB—BGP mode or Cilium BGP)
    - Pros: simple, horizontal scale via ECMP, no appliance, native to K8s.
    - Cons: requires BGP-capable ToRs and a clean FE subnet/VRF.

- Cloud/Managed LB (AWS NLB/ALB, Azure Standard LB, GCP GLB;)
    - Pros: turnkey, WAF/DDoS add-ons.
    - Cons: cloud-only; opaque datapath; $$ at scale.

- Hardware ADC (F5/A10/NSX-ALB, etc.)
    - Pros: rich L7 features, enterprise edge.
    - Cons: extra boxes, cost, another hop; overkill for internal FE.

- L2 VIP (keepalived/VRRP)
    - Pros: dead simple.
    - Cons: single-active node, limited throughput, weak HA at scale.

- DNS round-robin / Anycast (edge)
    - Pros: great Internet-edge distribution.
    - Cons: this can help get traffic to the right DC/Region, but doesn't meet the external LB requirement within DC, and hence you would still need one from the above options

BGP to ToR with Cilium BGP is the recommended option for design for these reasons:

- No extra hardware: VIPs are announced by Cilium agents on FE nodes; ToRs ECMP to them.
- Integrated stack: Cilium as primary CNI (recommended), no extra LB pods (vs. MetalLB speakers/FRR).
- Performance: Cilium's eBPF datapath + ToR ECMP = low latency, line-rate L4, and clean scale-out by simply adding FE ingress replicas.
- Resilience: Add BFD on BGP peering for sub-second failover; node/pod loss just rebalances via ECMP.
- Isolation fit: Announce /32 VIPs only on the FE VRF/subnet; backend/storage RDMA rails stay untouched.

INGRESS/GATEWAY LB:

Web server containers reside in the FE namespace and are the first Kubernetes workloads to handle external user/API traffic before it reaches backend training/storage rails. This works as L7 reverse proxy running as pods in the Frontend (FE) namespace

Function:

- Serve Web UIs / APIs for:
- Training job submission (Kubeflow UI, Ray Dashboard, etc.)
- Model registry, metadata queries
- Monitoring dashboards (Grafana, custom AI metrics UI)
- Terminate TLS / Handle Auth (if no service mesh on FE)
- Reverse proxy to backend REST/gRPC services

One of these can be chosen based on the use case

- NGINX Ingress – simple, huge community, great for HTTP(S) APIs & UIs.
- HAProxy Ingress – very fast L7, great TCP/HTTP mix, low latency.

- Gateway API (Envoy/Contour/Cilium Gateway/HAProxy) – the **NEW** k8s-native way; cleaner CRDs, multi-team friendly. Prefer this for greenfield.
- Expose with: cloud LB or bare-metal LB (BGP/MetalLB/Cilium LB).

2) Optional service mesh — FE only

- If you need mTLS, authN/Z, rate-limit, JWT/OIDC, put a mesh (e.g., Istio/Linkerd) only on FE namespaces.
- Never sidecar the training pods; the hot path stays on RDMA rails with no L7.

ADDRESSING & ISOLATION

- FE-only VRF/subnets (e.g., `vrf-fe`, `10.20.50.0/24`).
- No L3 routing to Backend (NCCL) or Storage (NVMe-oF) fabrics.
- VIPs are /32 in FE (announced via BGP/ECMP from FE nodes).
- Nodes/pods use primary CNI only (e.g., Cilium eBPF) on `eth0`.

INTERFACES INSIDE THE POD

- Single interface: `eth0` (primary CNI).
- No Multus/SR-IOV on FE pods.
- FE pods expose HTTP(S)/gRPC via ClusterIP/Headless/NodePort; Ingress/Gateway fronts them.

DUAL-RAIL USAGE (A/B)

- Not used. FE uses a single fabric; scale/HA via replicas + ECMP, not dual rails.
- Spread ingress/gateway replicas across FE nodes/racks.

MTU & BUFFERS (CRITICAL)

- MTU 1500 (default). Jumbo optional but not required.
- No PFC/ECN classes; keep FE queues lossy with sane AQM on ToRs.
- Tune ingress workers, keepalive, connection limits (L7 concern).

QOS & CONGESTION CONTROL (ROCEV2)

- N/A (FE does not carry RoCE).
- If needed, use TCP BBR/CUBIC and L7 rate-limits at Gateway.

G) SERVICE DISCOVERY FOR RANKS

- FE service discovery via Kubernetes DNS (`svc.ns.svc.cluster.local`) and Gateway/Ingress hostnames (e.g., `api.example.com`).
- external-dns can publish VIPs to corporate DNS.

SECURITY

- North/South NGFW/WAF in front of FE VIPs.
- Default-deny NetworkPolicies; allow from `fe-gateway` → `fe-apps` only.
- mTLS/JWT/OIDC at Gateway; signed images (Cosign); least-privilege RBAC.

FAILURE HANDLING

- BGP/ECMP to FE nodes (optionally BFD for sub-second failover).
- Multiple ingress/gateway replicas; PDBs + HPA.
- Node loss → ECMP rebalances; pod loss → Service routing selects healthy pod.

EXAMPLE

BGP ToR Peering policy (example):

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeeringPolicy
metadata:
  name: fe-bgp
spec:
  nodeSelector:
    matchLabels: { fabric.role: fe }   # label FE nodes
  virtualRouters:
  - localASN: 65010
    serviceSelector: {}              # advertise all LB Services in this cluster/VRF
    neighbors:
    - peerASN: 64512
      peerAddress: 10.20.50.1        # ToR-A
      connectRetryTimeSeconds: 5
      holdTimeSeconds: 9
      keepAliveTimeSeconds: 3
      ebgpMultihop: false
      authSecretRef: { name: bgp-md5 }
    - peerASN: 64512
      peerAddress: 10.20.50.2        # ToR-B
```

Ingress/Gateway Service (VIP from FE pool):

```
apiVersion: v1
kind: Service
metadata:
  name: gw-lb
  namespace: fe-gateway
spec:
  type: LoadBalancer
  loadBalancerIP: 10.20.50.110       # VIP announced via BGP
  selector: { app: gateway }
```

```
  ports:
  - { name: https, port: 443, targetPort: 8443 }
```
.

L7 routing — Gateway API (preferred)

```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata: { name: haproxy-gw }
spec: { controllerName: haproxy.org/gateway-controller }

---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata: { name: fe-gw, namespace: fe-gateway, labels: { app: gateway } }
spec:
  gatewayClassName: haproxy-gw
  addresses: [{ type: IPAddress, value: 10.20.50.110 }]
  listeners:
  - name: https
    protocol: HTTPS
    port: 443
    hostname: api.example.com
    tls: { mode: Terminate, certificateRefs: [{ name: fe-tls }] }

---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata: { name: fe-route, namespace: fe-apps }
spec:
  parentRefs: [{ name: fe-gw, namespace: fe-gateway }]
  hostnames: ["api.example.com"]
  rules:
  - matches: [{ path: { type: PathPrefix, value: "/ui" } }]
    backendRefs: [{ name: fe-ui-svc, port: 80 }]
  - matches: [{ path: { type: PathPrefix, value: "/api" } }]
    backendRefs: [{ name: fe-api-svc, port: 8080 }]
```

If you prefer Ingress: swap Gateway/HTTPRoute for NGINX/HAProxy Ingress resources; the VIP Service stays the same.

Security & segmentation (FE only)

Default-deny, allow from Gateway to apps, block FE→RDMA subnets.

```
# Namespace default deny
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: default-deny, namespace: fe-apps }
spec:
```

```
  podSelector: {}
  policyTypes: ["Ingress", "Egress"]

---
# Allow only Gateway to reach FE app Services
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: allow-from-gateway, namespace: fe-apps }
spec:
  podSelector: { matchLabels: { app.kubernetes.io/part-of: fe-app } }
  ingress:
  - from:
    - namespaceSelector: { matchLabels: { ns: fe-gateway } }
    ports: [{ protocol: TCP, port: 80 }, { protocol: TCP, port: 8080 }]

---
# Block egress to backend/storage subnets
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: deny-rdma-nets, namespace: fe-apps }
spec:
  podSelector: {}
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        - 10.60.0.0/16    # Backend Rail-A
        - 10.61.0.0/16    # Backend Rail-B
        - 10.70.0.0/16    # Storage rail(s)
```

Cilium Network Policy

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: fe-allow-gw-to-api
  namespace: fe-apps
spec:
  endpointSelector:
    matchLabels:
      app.kubernetes.io/name: fe-api          # pods being protected
  ingress:
  - fromEndpoints:
    - matchLabels:
        k8s:io.kubernetes.pod.namespace: fe-gateway   # only traffic from gateway pods
```

```
  toPorts:
  - ports:
    - port: "8080"        # containerPort of fe-api (name it "http" in your Deployment)
      protocol: TCP
    rules:
      http:
      - method: "GET"
        path:  "^/api/v1/health$"
      - method: "POST"
        path:  "^/api/v1/jobs$"
        headers:
        - name: "content-type"
          value: "application/json"
      - method: "GET"
        path:  "^/api/v1/models/[^/]+$"
        host:  "api.example.com"   # optional host match
```

Kyverno policies to block unsigned images

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-signed-images
spec:
  validationFailureAction: enforce
  rules:
  - name: check-image-signature
    match:
      resources:
        kinds:
        - Pod
    verifyImages:
    - image: "registry.example.com/*"
      keyless:
        issuer: "https://token.actions.githubusercontent.com"
        subject: "repo:myorg/*"
```

BACKEND (BE) TRAINING:

- NCCL uses the Multus VF (RoCE/IB) interface; bypass kube-proxy.

ADDRESSING & ISOLATION

- **Per-rail subnets:** e.g., 10.60.0.0/16 (rail-A), 10.61.0.0/16 (rail-B).
- **No NAT**, jumbo MTU end-to-end (see MTU below).

INTERFACES INSIDE THE POD

- **RoCEv2:** attach two Multus networks: roce-a → net1, roce-b → net2.
- **IB:** either hostNetwork with the ib* device, or Multus with IB SR-IOV.
- **Do not** run kube-proxy, iptables, or a service mesh on these interfaces.

DUAL-RAIL USAGE (A/B)

- **Do not bond** RDMA links for LACP—use **multi-NIC** at the framework level.
- Enable **NCCL/UCX multi-rail**, e.g.:
    - NCCL_SOCKET_IFNAME=net1,net2
    - NCCL_IB_HCA=mlx5_0,mlx5_1 (map net1, net2 to these two NIC- mlx5_0 & mlx5_1)
    - NCCL_CROSS_NIC=1 (allow GPU traffic to spread across multiple NIC, for higher utilization)
    - UCX_NET_DEVICES=mlx5_0:1,mlx5_1:1
- Fabric side: ECMP on leaf/spine for per-flow hashing; consistent hashing to reduce reordering.

MTU & BUFFERS (CRITICAL)

- Set 9000 (or 8900+) on VF, PF, switch ports, and SVIs—everywhere on Backend.
- RDMA does not “discover” MTU mismatches; one wrong hop = bad performance/timeouts.
- Ensure per-queue headroom includes PFC pause frames at 9k.

QOS & CONGESTION CONTROL (ROCEV2)

- Map backend DSCP/802.1p to a **dedicated lossless class** only.
- **PFC** enabled on that class with **PFC watchdog**.
- **ECN/WRED** on leaf/spine at ~70–80% queue occupancy.
- **ETS** to reserve bandwidth for the backend class and cap best-effort.

SECURITY

- **No NetworkPolicies** on the backend path (they’re L3/L4 constructs that don’t apply to RDMA in practice).

- Keep registry access, CI/CD, and auth on the **Frontend** network.

FAILURE HANDLING

- Single rail down: multi-rail NCCL/UCX continues on the surviving NIC; you'll see reduced throughput but job should live.
- Node drain/ToR reload: design jobs with elastic launch where possible; otherwise make the failure domain a **Scheduling Unit (SU)** and keep ranks SU-local.

EXAMPLE

** Net3 and Net4 can be created similar way for Backend-Storage Fabric

NetworkAttachmentDefinitions (NAD-one per rail):

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: roce-a
  namespace: ai-train
  annotations:
    k8s.v1.cni.cncf.io/resourceName: rdma/hca_shared_a

spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "type": "sriov",
      "name": "roce-a",
      "ipam": { "type": "host-local",
        "ranges": [[{ "subnet": "10.60.0.0/16" }]],
        "routes": [{ "dst": "0.0.0.0/0" }]
      },
      "mtu": 9000
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: roce-b
  namespace: ai-train
  annotations:
    k8s.v1.cni.cncf.io/resourceName: rdma/hca_shared_b
spec:
  config: |
    {
```

```
    "cniVersion": "0.3.1",
    "type": "sriov",
    "name": "roce-b",
    "ipam": { "type": "host-local",
      "ranges": [[{ "subnet": "10.61.0.0/16" }]],
      "routes": [{ "dst": "0.0.0.0/0" }]
    },
    "mtu": 9000
  }
```

Training pod (GPU + dual-rail RDMA):

```
apiVersion: v1
kind: Pod
metadata:
  name: train-w0
  namespace: ai-train
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [{ "name": "roce-a", "interface": "net1" },
       { "name": "roce-b", "interface": "net2" }]
spec:
  restartPolicy: Never
  nodeSelector:
    feature.node.kubernetes.io/pci-15b3.present: "true"
  containers:
  - name: worker
    image: nvcr.io/nvidia/pytorch:24.06-py3
    resources:
      limits:
        nvidia.com/gpu: 8
        rdma/hca_shared_a: 1
        rdma/hca_shared_b: 1
    env:
    - { name: NCCL_DEBUG, value: INFO }
    - { name: NCCL_SOCKET_IFNAME, value: "net1,net2" }
    - { name: NCCL_IB_HCA, value: "mlx5_0,mlx5_1" }
    - { name: NCCL_CROSS_NIC, value: "1" }
    - { name: NCCL_NET_GDR_LEVEL, value: "PHB" }
    securityContext: { allowPrivilegeEscalation: false, capabilities: { drop: ["ALL"] } }
```

STORAGE (STO):

OVERVIEW

Choose one of the storage system

NVMe-oF RoCE:

- Keep storage on its own VRF/fabric.
- Use NVMe multipathing (host nvme multipath or vendor client) across Rail-A/Rail-B.
- Do not LACP the storage RDMA NICs; let the protocol/client multipath.

Parallel FS (BeeGFS/Lustre/VAST RDMA/NFS):

- For RDMA clients, use client multi-rail; for NFS/TCP you may use LACP on storage NICs only if the

ADDRESSING & ISOLATION

- Storage-only VRF(s) (e.g., `vrf-sto`) with dedicated subnets:
  - Single rail: `10.70.0.0/16`
  - Dual rail: `10.70.0.0/16` (Rail-A), `10.71.0.0/16` (Rail-B).
- No routing to FE or Backend RDMA subnets.
- Targets expose stable portals (IPs/DNS) per rail.

INTERFACES INSIDE THE POD

- Attach only to pods that need high-throughput storage:
- net3/net4 via Multus + SR-IOV CNI (VF from storage NIC rails).
- Alternative: host-mounted storage (preferred for many apps) to keep pods simple.

DUAL-RAIL USAGE (A/B)

- Default: Active/Standby via ANA/multipath (simplest, predictable latency).
- Active/Active only if array supports symmetric access; enable round-robin/queue-depth balancing.
- Each pod (or host) should connect to both portals (A/B) and rely on `nvme-multipath`.

MTU & BUFFERS (CRITICAL)

- MTU 9000 end-to-end (NIC ↔ ToR ↔ spine ↔ target). Verify with `ping -M do -s 8972`.
- Size NIC rings appropriately; avoid interrupt storms.
- Ensure storage target NICs and switches match MTU and PFC profiles exactly.

QOS & CONGESTION CONTROL (ROCEV2)

- Separate lossless class from Backend NCCL.
- PFC only on storage priority; enable PFC watchdog.
- ECN/DCQCN tuned to ~70–80% queue occupancy; WRED on congested queues.
- Headroom per port sized for worst-case incast during checkpoints.

SECURITY

- VRF/ACL isolation; no FE route.

- NVMe-TLS if required; secrets for host NQN + credentials in Kubernetes Secret.
- Limit which namespaces can attach storage VFs (RBAC + Admission/Policy).

FAILURE HANDLING

- Link/ToR/NIC failure → multipath flips to the other rail.
- Target failover handled by array/namespace ANA states.
- Monitor I/O error counters, latency spikes, path state; alert on degraded to optimized transitions.

EXAMPLES

Adding Net3/Net4 for Storage Fabric

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: sto-a
  namespace: ai-train
  annotations:
    k8s.v1.cni.cncf.io/resourceName: rdma/hca_sto_a
spec:
  config: |
    {"cniVersion":"0.3.1","type":"sriov","name":"sto-a",
     "ipam":{"type":"host-local","ranges":[[{"subnet":"10.70.0.0/16"}]],
           "routes":[{"dst":"10.70.0.0/16"}]},
     "mtu":9000}
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: sto-b
  namespace: ai-train
  annotations:
    k8s.v1.cni.cncf.io/resourceName: rdma/hca_sto_b
spec:
  config: |
    {"cniVersion":"0.3.1","type":"sriov","name":"sto-b",
     "ipam":{"type":"host-local","ranges":[[{"subnet":"10.71.0.0/16"}]],
           "routes":[{"dst":"10.71.0.0/16"}]},
     "mtu":9000}
```

APPENDIX A- KUBERNETES SIZING

It's a workload choice, not a fixed rule:

- Most large training jobs: 1 training pod per node (requesting all 8 GPUs on an XE9680).
  - Then 256 nodes ⇒ ~256 training pods for a 2000-GPU job.
- Alternative mappings (less common for big training):
  - 2 pods × 4 GPUs each, or 8 pods × 1 GPU each (or MIG for inference).
- Plus daemonset pods (GPU Operator, SR-IOV DP, monitoring, logging, CNI agents, etc.)—typically a handful more per node.

Operationally, 1 pod/node keeps networking, NCCL rank mapping, and VF allocation simplest at scale.

VF ALLOCATION

With SR-IOV CNI, VFs are allocated to pods (not shared by default):

- If a training pod needs two backend rails, it gets two VFs (e.g., net1 = rail-A, net2 = rail-B).
- If you also give the pod storage rails, add two more VFs (net3, net4).
- Each VF inside a pod gets its IP (from the NetworkAttachmentDefinition's IPAM).
- Another pod on the same node would receive its own distinct VFs and IPs.

So for the common **1 training pod per node** pattern:

- That pod has eth0 (primary CNI, Frontend)
- net1/net2 (Backend A/B VFs)
- net3/net4 (Storage A/B VFs).
- The node itself can host many VFs in total; exact limits depend on NIC/firmware, but **your 2–4 VFs per node** use-case is well within modern NIC capabilities.