Converged DAOS on HGX B200/B300

AI-Assisted Engineering of a High-Performance Storage Tier for ML

Alexander Timofeyev · **ML Infrastructure, LinkedIn**
linkedin.com/in/alexandertimofeyev

DAOS User Group · 2026

LinkedIn

An eight-section walk-through

Agenda

- Why converged AI + storage matters
- Why DAOS — and what "converged" means here
- Hardware, network, and CPU/NUMA design
- Operational engineering — what hurt
- Validation and benchmarks
- Kubernetes integration direction
- How an LLM coding assistant changed our loop
- Lessons, surprises, recommendations

GPUs are starving for bytes

Benefits

No separate storage fleet — saves CapEx, power & infrastructure.

Addresses multi-hop penalty — Disaggregated storage adds hops/latency.

Simpler K8s topology — one node pool, one DaemonSet.

Challenges (and how they're addressed)

Shared CPUs → DAOS cores pinned via first_core, excluded from K8s scheduling.

Shared network → NOT an issue: NCCL on dedicated IB fabric, DAOS on RoCE

Shared failure domain → EC erasure coding with topology-aware placement

Shared memory → Explicit scm_size keeps DAOS allocation bounded

Storage vs compute SLA asymmetry

GPUs are starving for bytes

Disaggregated storage adds a fabric and hops

Per-NIC ceiling. A modern HGX node can read ~50 GB/s from a single 400G link — saturable in microseconds.

Extra hops in a separate fabric. Storage in its own rack: client → client-ToR → spine → storage-ToR → storage. 3–5 hops, plus a fabric to size and cable.

Two failure domains. Two clusters, two on-call rotations, two software roadmaps.

Converged keeps storage on the same fabric, with fewer hops

Same RoCE for both. DAOS RPCs share the fabric we already operate for collectives — no separate storage net.

Fewer hops, partial loopback. Most I/O still traverses the ToR (placement is hash-based and global); ~1/N of operations are loopback-local on the same NIC.

Same hardware footprint. 8 NVMe per GPU node already in the BOM; converged drives them rather than leaving them as local scratch.

The bet: complexity at the node, simplicity & performance at the cluster.

DAOS as the Tier-0 scratch layer for ML

Starting scope: a Tier-0 ephemeral / scratch tier for ML training (staging, dataloaders, checkpoints). Durable data and governance stay in the existing lake. Scope expands as we adopt.

User-space data path

OS bypass via OFI + SPDK + PMDK. No kernel I/O on the hot path. CPU spent on storage is bounded and predictable.

MD-on-SSD (no PMem)

Phase 1 (v2.6+) lets us run on commodity NVMe-only hardware. SCM lives in DRAM (tmpfs), bulk on NVMe.

RDMA-native fabric

Mercury / OFI verbs with optional UCX. Same RoCE we already run for collective comms — no second fabric to operate.

Object-class flexibility

Per-container EC vs replicated vs single. Training-data and checkpoint redundancy decided independently.

What we run DAOS on

Per-node hardware budget

Compute AMD EPYC 9V45, 192 cores / 384 threads, dual NUMA

GPUs NVIDIA HGX B200 / B300 — 8× GPU per node

Network 2× ConnectX 400G RoCE (one per NUMA)

NVMe 8× Gen5 NVMe behind PEX890xx PCIe switches

Memory TBs of DDR5 — tmpfs SCM

OS Azure Linux 3 (Mariner-derivative)

DAOS v2.6.5-rc1 with cherry-picked SWIM patch

Engines / node 2 (one per NUMA), 16 targets each

Why the topology matters

Two NUMAs, two NICs. Lets each engine speak to the NUMA-local NIC; no cross-socket QPI on the I/O path.

PEX890xx PCIe switches. Eight NVMe drives live behind two upstream switches. Default IOMMU groups are too coarse for VFIO — see the engineering section.

No PMem. MD-on-SSD changes the metadata sizing calculus; we pin scm_size explicitly to keep DRAM for GPU clients.

One RoCE fabric, two roles

Per node

2× ConnectX-7 / 400G RoCE. Both NICs land on the same ToR. NUMA-aligned (one per socket). Carries DAOS storage traffic AND in-band management on the same fabric.

8× ConnectX-7 / 400G IB NDR. Separate compute fabric for NCCL collectives. Zero contention with storage traffic.

Macvlan-bond on the RoCE pair. Active-backup for the management overlay; physical NICs keep their IPs/GIDs for line-rate RDMA. FI_VERBS_GID_IDX=5 is the deterministic post-race answer.

Combined RoCE in-band+storage vs DGX-style dedicated

CAPEX. Roughly half the storage-side switches. One VLAN scheme, one cabling pass, one set of optics.

OPEX. One fabric to monitor, one PFC/ECN tuning surface, one ops team. DGX SuperPOD splits these; we don't.

BW headroom. 2× 400G = 100 GB/s per node; NVMe ceiling is ~52 GB/s read. Network has the headroom; isolation isn't free, but it isn't load-bearing here.

What we trade. Some storage QoS isolation. Mitigation: PFC priority lanes fpr RDMA; jumbo MTU 9000 on storage, 1500 on in-band. Path to dual-rail open if BW pressure grows.

Where the storage CPU lives

Engines busy-poll. Pin to physical cores at NUMA edges, leave HT siblings idle, reserve at kubelet level.

AMD EPYC 9V45 · 192 cores / 384 threads · dual NUMA

NUMA	Component	Logical CPUs	Cores
0	OS / K8s / kubelet	0–3, 192–195	4
0	**DAOS engine 0**	**4–23, 196–215**	**20**
0	Available to K8s/ML	24–95, 216–287	72
1	OS / K8s / kubelet	96–99, 288–291	4
1	**DAOS engine 1**	**100–119, 292–311**	**20**
1	Available to K8s/ML	120–191, 312–383	72

Four problems we did not expect

Each took one to several days. None appears in any vendor doc as a thing to watch for. They're the value of building this on real hardware.

RDMA-CM stale state across restarts

After ~20 DAOS start/stop cycles, HCA firmware's CM table fills with stale QPs. ib_write_bw fails; pool create hangs. Module reload is not enough — only a full reboot clears it. Worst at K8s scale where pod restart churn is daily.

VFIO on PEX890xx PCIe switches

IOMMU groups collapsed; eight NVMe shared one group with their upstream switch ports. Fix lives in BIOS — AmdSetup.ACSEnable — pushed via Redfish. Stock kernel quirks don't help.

SWIM false positives under load

Under fabric contention, SWIM ping latency spiked past the failure threshold and triggered phantom rank failures. DAOS-17111 fix is master-only; we cherry-picked four PRs and rebuilt libcart for v2.6.5-rc1.

`first_core` is absolute, not NUMA-relative

Engine 1 on NUMA 1 wants first_core: 100, not 4. `pinned_numa_node` controls memory affinity and NIC selection but doesn't shift `first_core` — they're independent. Silent misalignment — engines run, performance drops, no error in the log.

Limitations we're working around

Workarounds

- **daos_server.yml — three settings that matter.** crt_timeout: 600 (default 60 → pool-create timeouts at scale); log_mask: ERR (DEBUG burns ~20× the IOPS); SWIM_SUSPECT_TIMEOUT=120000 (flags 5-second NIC bounces as transient, not dead).
- **SWIM DAOS-17111 cherry-picked onto v2.6.5-rc1.** Four upstream PRs (#15894, #15924, #15945, #16519) — master-only, never backported. We rebuild libcart manually and overlay it in the container image. Retire when v2.8.x lands.
- **UCX dc_x is what we want at scale — and it's broken for us.** ofi+verbs;ofi_rxm is stable but $O(n^2)$ memory at 1024 ranks (~131 GB cluster-wide). UCX dc_x solves that with O(1) — but hits DAOS-18828 (DC wireup race) on our RoCE setup. Confirmed on UCX 1.19 and 1.20. We stay on verbs.
- **No IPv6 on the control plane.** DAOS 2.6.5 hardcodes tcp4 and 0.0.0.0 on Mercury. FQDN + FI_VERBS_GID_IDX=5 silently falls back to v4 on dual-stack hosts. We deploy v4-only and flag it.
- **Unprivileged K8s — almost.** daos-server runs as non-root. daos_server_helper is setuid root by design and still requires CAP_SYS_ADMIN — DAOS gates iova_mode=va on geteuid != 0 only.

What we measured and how

Always isolate one variable. Stonewall every IOR. Re-run cold and warm. Single-client and aggregate are different questions.

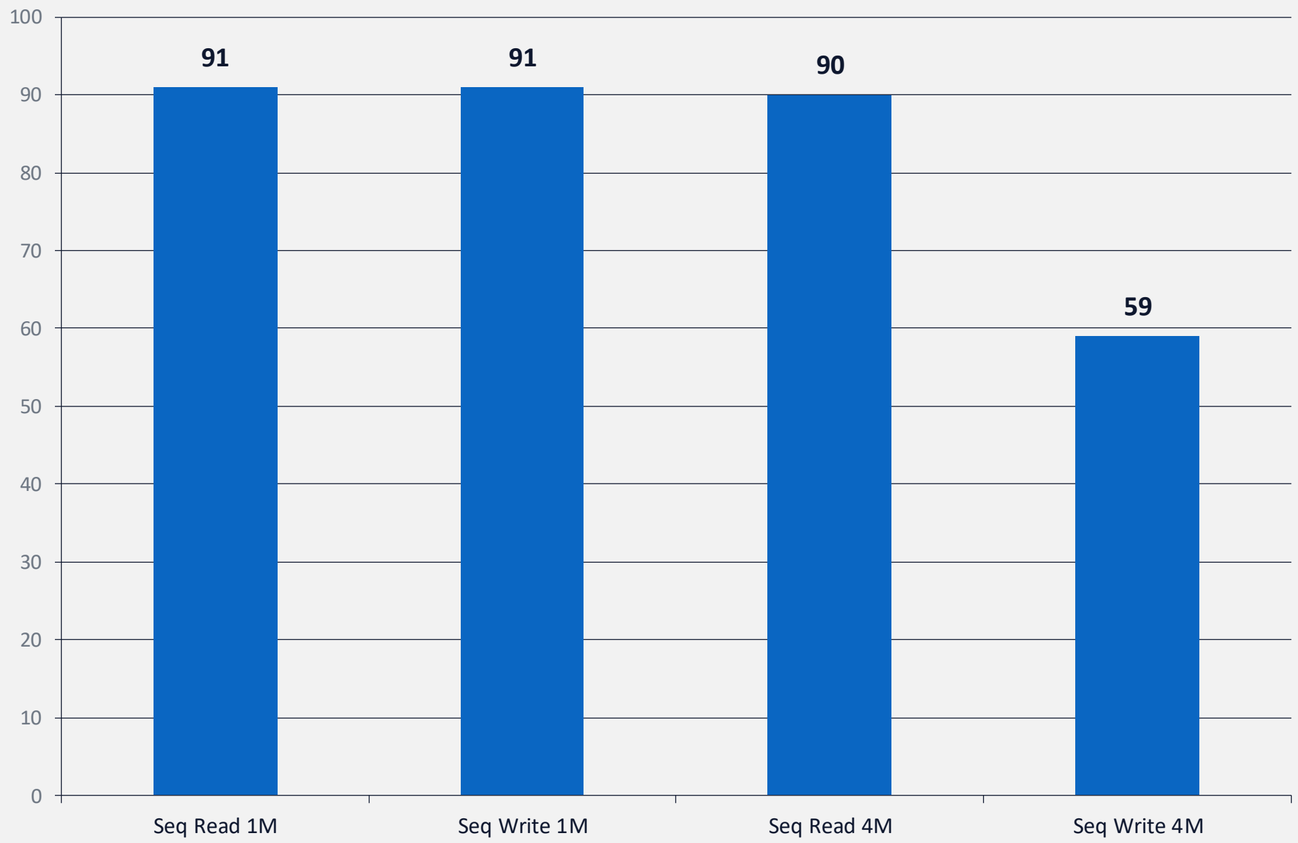
Five layers, one variable at a time

Layer	Tool	What we control for
Fabric	ib_write_bw	Per-NIC line rate, cross-subnet latency, GID selection
Storage device	fio (libaio + dfs)	Per-target ceiling vs aggregate; SCM vs NVMe placement
DAOS pool	fio --ioengine=dfs	Object class (SX vs EC vs S1); chunk size; rd_fac
Application	IOR with -F -D 60, MPI	Stonewall to remove startup bias; rank scaling; oclass
End-to-end	PyTorch training, AVRO dataloader	Native DFS vs dfuse POSIX; 10-step smoke; loss match

One client, three servers

Single client saturates one NIC and gets within 3% of theoretical fabric line rate. Random IOPS at 3.3M (aggregated across ranks from one client host); QD=1 single-rank latency 55–86 µs.

Per-client throughput · DAOS 2.6.5-rc1, EPYC 9V45



Fourteen nodes, co-located, no PMem

Reads NVMe-bound at 89% of theoretical. Writes NVMe-bound. IOPS target-count bound. On all three axes, the network is comfortable.

605 GiB/s

Network utilization on the write workload. Even saturating NVMe writes, the fabric is at one-third capacity. Converged architecture argument.

464 GiB/s

Aggregate read · SX, file-per-process, cache defeat. 89% of NVMe theoretical (605 / 678 GiB/s).

20.5 M IOPS

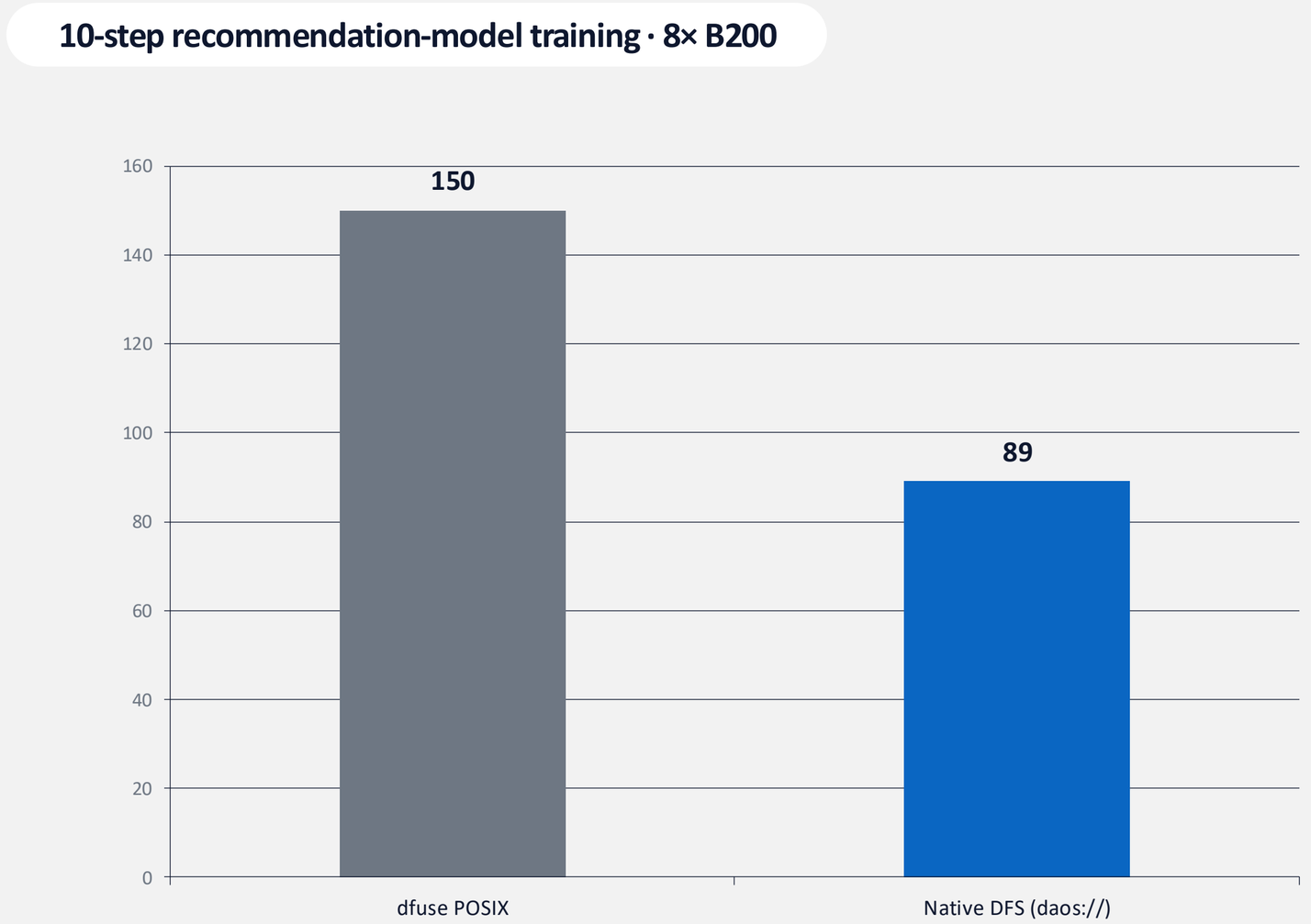
Peak write · 458 GiB/s sustained 300 s. SPDK NVMe steady-state write rate is the ceiling.

33% NIC

Random write 4K · S1 oclass, 1344 ranks, 60 s stonewall. Scales linearly to ~1300 ranks.

PyTorch on DAOS — does it actually train?

Native DFS dataloader (daos:// URI, no FUSE) is 40% faster than dfuse POSIX. Loss matches end-to-end. New bottleneck is the parser, not the storage. Async checkpoint: client returns after RDMA completion, SPDK persists to NVMe in the background; training step doesn't block on the flush.



Where the K8s story is going

Pod stack

daos-server as a DaemonSet. Non-privileged user runs the engine; CAP_SYS_ADMIN still required by the helper until upstream iova-mode patch lands.

VFIO device plugin. Static binding — each pod claims a fixed list of /dev/vfio/<group>.

daos-agent DaemonSet alongside. Lightweight, serves both NUMAs.

Training pods talk DFS, not PVCs. Local-storage CSI drivers assume kernel block devices; DAOS bypasses that layer entirely.

What's still in flight

Unprivileged daos_server_helper. Upstream DAOS-12648 — ~15 LOC patch to broaden iova_mode check.

Hot rolling restart of engines. Without dropping in-flight pool RPCs. DaemonSet rolling-update + RPC drain pattern under prototype.

GID-aware fabric pinning. Cleaner than the FI_VERBS_GID_IDX=5 trick; upstream-track if achievable.

AI-assisted engineering of a storage tier

Honest framing: an AI coding tool didn't replace expertise. It redistributed where my hours went. Four buckets below; three concrete examples in the notes.

Where it accelerated

Vendor-doc synthesis (six PDFs in thirty seconds). Boilerplate: systemd units, Dockerfiles, networkd configs. Throwaway log analysis. Recreating a failed benchmark recipe from a stack trace.

Pair-programming on hardware

Example 1 · VFIO BIOS: two days chasing a kernel-quirk theory. Assistant reread dmesg and asked, 'have you checked the AMD ACS BIOS attribute?' Found via Redfish. Fixed. ~2 days saved.

Knowledge that survives me

Example 2 · Memory-file discipline. Every gotcha gets written down as we hit it. When the next operator walks into this cluster, the institutional knowledge isn't just in my head.

Where it didn't help

Architecture tradeoffs · reading what the hardware is actually doing · knowing when a benchmark is lying · verifying RDMA moved bytes vs claimed throughput · the call when to stop and reboot.

Six rules of thumb

Lessons

- **Check Redfish BIOS attributes before assuming kernel quirks.** Many 'kernel doesn't expose this' turned out to be 'BIOS doesn't expose this'.
- **Bond macvlans, not NICs.** Bonding physical NICs costs you dual-NIC RDMA. Bond virtual macvlans and let NICs keep their identities.
- **Pick a deterministic GID index.** Stop fighting boot ordering. Index 5 works post-race; index 3 only works if you reload mlx5_ib (which kicks NCCL).
- **Pin engines to physical cores, never to HT siblings.** Sibling thread contention costs more than the SMT throughput gain.
- **Build a native DFS dataloader path early.** dfuse is a fine first step; the 40% headroom pays for the C++ extension.
- **Write down every gotcha as you hit it.** Future-you and the next operator will both thank you.

Thank you.

Alexander Timofeyev · ML Infrastructure, LinkedIn

linkedin.com/in/alexandertimofeyev

Backup slides

Four-step procedure

Recipe

- **Set AmdSetup.ACSEnable = Enabled.** The AMD root port gains an ACS Extended Capability; IOMMU groups go from coarse subtree-wide to single-device per NVMe.
- **Kernel cmdline: iommu=pt.** Regen both grub.cfg paths on EFI hosts. Do NOT add pcie_acs_override; stock kernel ignores it.
- **PATCH via Redfish to /redfish/v1/Systems/1/Bios/Settings.** ETag must come from the parent /Bios endpoint. Body uses nested AmdSetup.ACSEnable; flat form returns 400.
- **Verify post-reboot.** lspci -vvv on the AMD root port shows ACSCap; per-NVMe IOMMU groups are single-device; dmg storage scan succeeds.

Why we patched libcart on v2.6.5-rc1

Patch detail

- **Symptom.** Phantom rank failures under concurrent IOR + small-IO workloads.
- **Root cause.** SWIM ping latency spikes past the failure threshold under fabric contention.
- **Upstream fix.** Four PRs (#15894, #15924, #15945, #16519) on master; never backported to v2.6.5.
- **What we did.** Cherry-picked the four commits onto v2.6.5-rc1; manually built libcart.so with gcc.
- **Conflicts.** One trivial copyright, one logic conflict (PR #16519 supersedes the earlier skip path in #15894).
- **Verification.** 359/359 exported symbols match stock 2.6.5; ldd subset matches; smoke + IOR pass.
- **Operational.** Retire the patched libcart when v2.8.x lands with the fix.

Per-node ceiling (measured)

Read BW. ~100 GiB/s aggregate · 2× 400G NICs (~52 GiB/s per engine, NIC-bound)

Write BW. ~33 GiB/s · NVMe sustained-write ceiling

Random IOPS. ~750K read · ~1.5M write per node

DAOS targets. 32 (16 per engine × 2 engines)

Engines. 2 (one per NUMA)

What changes at 512 nodes

Aggregate read ceiling. ~50 TB/s · NVMe-bound at 8× drives × 512 nodes

SWIM membership. O(log N) cost — fine, but watch tail under load

Pool service replicas. svc_rf=2 covers most failure modes; raise to 3 above ~256 nodes

Rebuild domain. EC stripes stay below 16-wide to avoid combinatorial fault domains

Operational gap. Rolling restart of engines without dropping pool RPCs (in prototype)

Converged is the wrong answer if...

Counter-cases

- **Your write rate exceeds per-node NVMe write capacity.** Disaggregated scales storage NVMe count independently of GPU node count; converged can't.
- **You don't operate the GPU nodes yourself.** Co-location ties storage SLO to GPU SLO; cross-team failure modes get messy.
- **You have PMem hardware.** MD-on-SSD's tmpfs metadata is not equivalent. Use the PMem.
- **You can't pin CPUs.** Engine busy-poll competes with everything unpinned. K8s reservation must be enforceable.
- **Your training is dataloader-bound but I/O-light.** Profile the parser first; you might be solving the wrong problem.

Cost framing · converged vs disaggregated

What converged saves

- Separate storage cluster: BOM, racks, power, cooling
- Second fabric — or fabric port density on the existing one
- Two ops teams, two on-call rotations
- One full data copy per training job (staging)

What converged costs

- ~20% of physical cores permanently reserved for storage
- Storage SLO bound to GPU-node availability
- More complex BIOS / IOMMU / networking per node
- Harder rolling-upgrade story — restart impacts local clients

Five common modes

Failure modes

- **NVMe drive fault.** Target stops; SWIM marks rank degraded; pool keeps serving from EC; rebuild kicks at ~30% CPU.
- **NIC fault.** RDMA fails on that NIC; engines on that NUMA stall; bond preserves SSH; partner NIC serves its NUMA.
- **Engine SIGKILL (OOM).** Almost always auto-sized scm_size eating all DRAM. Pin scm_size explicitly — 64 GiB per engine in our config.
- **Whole-node loss.** Two ranks gone; pool SvcReps survive when PS replicas are spread across nodes (default placement with svc_rf ≥ 1); rebuild starts; stale agent caches need SIGUSR2.
- **RDMA-CM stale state.** After enough start/stop cycles, fresh starts hang on QP setup. Reboot is the only fix we've found.

Six concrete steps · ordered by what bites earliest

Recommendations

- **Start with three nodes, not one.** Single-node DAOS can't exercise SWIM, EC, or pool replication.
- **Pin scm_size explicitly from day one.** Auto-sizing claims all DRAM; engine OOM under load is the most common new-deployment crash.
- **Build a native DFS dataloader path early.** Otherwise dfuse-only deployments will mislead your performance work.
- **Containerize the server, then the agent.** Container images make rolling out config changes (especially DAOS-17111) tractable on N nodes.
- **Write down every gotcha as you hit it.** BIOS toggles, GID indexes, magic env vars — they don't survive in your head past three weeks.
- **Talk to the DAOS team about your fabric early.** Provider choice (verbs vs UCX vs cxi) materially changes your debugging surface.