GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

NVIDIA Topograph Kubernetes Slurm: Private GPU Cluster Pilot

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-11-10 · nvidia topograph · topograph kubernetes · topograph slurm · topology aware scheduling · gpu cluster topology · slurm topology.conf · kubernetes node labels · kai scheduler kueue · infiniband topology discovery

Original article: https://gpusmith.com/articles/en/nvidia-topograph-private-gpu-clusters-adoption-gate



In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes: The September 2026 Release and What Followed
4. Provider-to-Scheduler Decision Matrix
5. Node-Local Affinity Versus Cross-Node Fabric Placement
6. Choosing the Scheduler Layer
7. Implementation Considerations and Process Changes
8. Data Analysis and Evidence
9. Implications and Future Directions
10. Frequently Asked Questions (FAQs)
11. Conclusion

Executive Summary

NVIDIA Topograph is an open-source service that discovers a cluster's network topology from a provider (a cloud API or an on-premises fabric tool) and translates it for a scheduler through an engine, publishing Kubernetes node labels, Slurm topology files (docs.nvidia.com), or Slinky ConfigMaps (raw.githubusercontent.com). It does not place jobs. Engines "only translate" and never discover topology themselves (raw.githubusercontent.com), so every placement gain depends on a separate scheduler honoring the output. This report, current as of November 2026, gives private GPU cluster operators an adoption gate for deciding whether to pilot it.

The release picture needs pinning. NVIDIA's developer blog is dated 22 September 2026 (developer.nvidia.com), while the repository's latest GitHub release, v1.1.0, was published on 26 October 2026 (api.github.com) and added Crusoe and Firebird providers (api.github.com). The blog's provider sentence names Google Cloud, Lambda, Nebius, Nscale and OCI, although its provider support table also lists Crusoe, so Crusoe support was already described before v1.1.0 shipped the provider code (developer.nvidia.com), whereas the README lists AWS, Crusoe, GCP, OCI, Nebius, Nscale, Lambda and NetQ (raw.githubusercontent.com). Treat the repository tag, not the blog, as the support list.

Three distinctions drive the decision. First, node-local alignment (NUMA, PCIe, NVLink inside a server) belongs to the kubelet Topology Manager and the device stack, and Topograph is described as complementary to it (raw.githubusercontent.com). Second, the default Kubernetes scheduler treats Topograph labels as a preference scored one pod at a time (raw.githubusercontent.com), whereas KAI Scheduler offers a hard required level (raw.githubusercontent.com) and Kueue offers required and preferred annotations (kueue.sigs.k8s.io). Third, non-MNNVL (Multi-Node NVLink) InfiniBand clusters depend on Topograph with an InfiniBand provider as the only network topology source, because the GPU clique label is absent (raw.githubusercontent.com).

Refresh is event-driven. Labels refresh when generation runs after a watched change (developer.nvidia.com), changes to node metadata do not trigger the observer (docs.nvidia.com), and the Slurm trigger fires only on node up and down events (docs.nvidia.com). A healthy pod also does not prove current topology

(docs.nvidia.com). The pilot therefore needs a wiring-change acceptance test, not just an install check.

Published evidence is mixed and not specific to Topograph. A production study reports a 10.6% end-to-end gain from topology-aligned scheduling on more than 9,600 GPUs (arxiv.org), while an older fat-tree study found minimal network-placement impact on a tapered cluster (www.sc17.supercomputing.org). No source reviewed quantifies Topograph itself, so any throughput or cost claim requires the buyer's own paired runs.

Introduction and Background

Large distributed training and inference jobs spend a measurable share of their time in collective communication, and where the ranks land in the fabric changes which links carry that traffic. NVIDIA's documentation states the premise plainly: nodes connected to the same leaf switch can communicate with lower latency and higher bandwidth (docs.nvidia.com). Slurm has long supported topology-aware allocation from a configuration file, but the maintainers of Topograph argue that node churn makes a static file unreliable (sc25.supercomputing.org). Their SC25 workshop abstract states that multi-node workloads should be scheduled on network-adjacent nodes (sc25.supercomputing.org).

This report examines how a GPU cluster network topology discovery service for Kubernetes and Slurm behaves in a private deployment: what Topograph does, what it explicitly does not do, and how to run an auditable pilot. It answers the search queries that bring operators to the tool: how Topograph works with Kubernetes and Slurm, how it relates to `topology.conf`, how it behaves on bare-metal InfiniBand, which node labels it writes, and how topology-aware scheduling differs between the default Kubernetes scheduler and gang-aware schedulers.

Two terms recur. A **provider** is the source of topology (a cloud API, NetQ, InfiniBand tooling), and an **engine** is the target format (Kubernetes, Slurm, Slinky, Node Feature Discovery, or a JSON graph). Discovery is asynchronous: a request is submitted, validated, and polled for a result.

This report is built on dated public specifications. It contains no benchmark runs, client results or partnership claims. Where the evidence is vendor-authored, it is labeled as such. Readers who act on commands below should validate them against the chart and documentation version they pin, and should involve their network and security engineers before granting any permissions.

Key Changes: The September 2026 Release and What Followed

What was published on 22 September 2026

NVIDIA's developer blog, published on 22 September 2026, presents Topograph as a bridge from provider-reported topology to scheduler-specific formats. The repository README frames the same role: Topograph sits between topology sources and schedulers and replaces neither. The latest GitHub release, v1.1.0, is dated 26 October 2026 (api.github.com). The earlier v1.0.0 release was published on 18 August 2026, before the blog (api.github.com).

The practical consequence is that a pilot started from the blog alone may not match a pilot started from the current tag. Record the exact release tag, chart version and image digest in the pilot file. The README clones from github.com/dsx-ai-factory/topograph, whereas the release API and raw documentation links cited here

use NVIDIA/topograph, so record the precise path you install from and confirm that it matches the release you pinned.

Provider inventory: three sources, three lists

The blog's prose names the cloud providers with working integrations as Google Cloud, Lambda, Nebius, Nscale and OCI, as noted above, but its support table also lists Crusoe, which the footnote says reads labels from Crusoe Managed Kubernetes nodes. The README names AWS, Crusoe, GCP, OCI, Nebius, Nscale, Lambda and NetQ (raw.githubusercontent.com). The main-branch overview additionally describes a provider for sites that inventory fabric cabling in Nautobot. The v1.1.0 release notes list the addition of the Crusoe provider (api.github.com) and a Firebird provider with simulation (api.github.com).

These lists are not contradictions so much as snapshots taken on different dates. The engineering response is to build the provider decision from the pinned tag's documentation, and to treat the blog as a dated introduction.

A label contract that Topograph owns

On Kubernetes, the documented default keys are `fabric.topograph.run/tier-N` and `accelerator.topograph.run/domain`, with an optional sub-domain key. Tier 0 is closest to the compute node and tier numbers increase with distance (raw.githubusercontent.com). Label values of 63 characters or fewer are used as-is, and longer values are hashed, which preserves locality but not the original identifier (raw.githubusercontent.com). Not every provider produces both topology types.

One point corrects a common assumption. Topograph does not write the GPU Operator's clique label. There is no default accelerator-domain source label, and an operator who wants the clique label to be authoritative must set it explicitly.

Event-driven refresh, not polling

The Node Observer is a controller that monitors configured node and pod changes. Events include node adds and deletes and trigger-pod readiness transitions (docs.nvidia.com). Bursts are coalesced, so with the default 15 second delay a result is never available sooner than 15 seconds after node churn stops (docs.nvidia.com). NVIDIA's own caveat is that whether a fabric change becomes visible depends on the provider and its triggering events. The "Change detection, stale labels, rollback and owners" checklist below converts these statements into an acceptance test.

Provider-to-Scheduler Decision Matrix

Choosing a provider for a private cluster

Private clusters are usually one of four shapes: InfiniBand without NetQ, Spectrum-X or InfiniBand with NetQ, a multi-node NVLink (MNNVL) rack system, or a colocation or cloud lease where the vendor already exposes topology. NVIDIA's InfiniBand provider documentation recommends NetQ over direct discovery where NetQ is deployed, calling it the standard approach for Spectrum-X (raw.githubusercontent.com).

For the InfiniBand bare-metal variant, the provider discovers the switch tree using `ibnetdiscover` (docs.nvidia.com), is intended for bare-metal clusters such as Slurm, needs `pdsh` on the node running Topograph (docs.nvidia.com), and requires no cloud credentials. All InfiniBand variants are single-region only (raw.githubusercontent.com). The Kubernetes variant is different in a way that matters to security reviewers: deployments typically run the broker privileged (docs.nvidia.com).

The NetQ provider queries the NetQ API for fabric topology and NVLink domain data (raw.githubusercontent.com), requires a running NetQ server reachable from the Topograph host, and treats NVLink domain discovery as best-effort, returning only the switch tree if that fails.

Table 1 below maps provider families to their topology source, whether they see cross-node fabric or only an NVLink partition, and the scheduler outputs that apply. It summarizes documentation for the v1.1.0 documentation set and should be re-derived from the tag you pin.

Provider family	Topology source	Cross-node fabric?	NVLink domain?	Typical engine and scheduler	Pilot prerequisite
infiniband-bm	`ibnetdiscover` through `pdsh`	Yes, switch tree	Optional add-on for NVLink-connected GPUs	Slurm `topology.conf` (docs.nvidia.com)	`pdsh` and `sudo ibnetdiscover` on nodes
infiniband-k8s	Same discovery from a Kubernetes broker	Yes, switch tree	Optional add-on	Kubernetes labels for native affinity, KAI or Kueue	Privileged broker and host `/sys/class` mount
netq	NetQ API	Yes	Best-effort	Any engine; recommended for Spectrum-X	Reachable NetQ server and account
dra	Existing clique node label	**No** (docs.nvidia.com)	Partition membership only (raw.githubusercontent.com)	Slinky block topology only	GPU Operator clique label present
aws, gcp, oci, nebius, lambda, nscale	Each cloud's own topology API (raw.githubusercontent.com)	Provider-reported tiers	Where the API exposes it	Slurm, Kubernetes, Slinky per provider page	Cloud IAM read permission; OCI IMDS variant is Slurm-only
crusoe	Crusoe-published node labels, because a guest VM cannot run `ibnetdiscover` (docs.nvidia.com)	Provider-reported	Provider-reported	Kubernetes	Managed Kubernetes nodes carrying those labels

Two lessons come out of *Table 1*. The **dra** provider is not a general Kubernetes MNNVL topology provider: it does not discover the switch fabric between partitions, so it cannot guide placement when a workload spans more than one NVLink partition. And for the common private case of InfiniBand without MNNVL, NVIDIA states that Topograph with an InfiniBand provider is the only source of network topology for scheduling decisions (raw.githubusercontent.com).

Mapping provider output to scheduler artifacts

The engines translate one canonical graph into different artifacts. The decision is mostly about which artifact your scheduler can consume.

- **Kubernetes engine:** writes node labels directly through the Kubernetes API. It lists nodes once and patches only changed labels. This is the engine for native pod affinity, KAI Scheduler and Kueue.
- **Node Feature Discovery (NFD) engine:** publishes NodeFeatureGroup resources instead of labels. Use it only when a downstream component already consumes NFD. The NodeFeatureGroup API is alpha and must be explicitly enabled (raw.githubusercontent.com), and a scheduler still has to decide which leaf group to use.
- **Slurm engine:** produces tree or block topology through a plugin parameter. When block sizes are not configured, Topograph derives them from the discovered accelerator domains. If no output path is set, the text is returned in the HTTP response.
- **Slinky engine:** updates Slurm topology configuration stored in Kubernetes ConfigMaps. Slinky also reads a node annotation to place NodeSet pods in the topology (slinky.schedmd.com), and a topology file must still exist for dynamic topology to work (slinky.schedmd.com).
- **Graph engine:** returns per-instance network layers and accelerator labels as JSON, which is useful as the "discovered" side of the audit worksheet below.

Node-Local Affinity Versus Cross-Node Fabric Placement

Node-local: GPU, CPU and NIC proximity

Inside a server, the question is which GPUs, CPU cores and NICs share a PCIe switch, a NUMA node or an NVLink mesh. `nvidia-smi topo -m` prints that matrix. Its legend distinguishes NVLink connections (docs.nvidia.com), a single PCIe bridge, and paths that cross the inter-socket interconnect. Another mode shows CPU and memory affinity and the NUMA node for each GPU.

In Kubernetes, the Topology Manager is a kubelet component that coordinates resource alignment on one node (kubernetes.io). Its default policy performs no topology alignment (kubernetes.io), while the restricted policy rejects a pod whose alignment is not preferred (kubernetes.io). The feature has been stable since Kubernetes v1.27 (kubernetes.io). NCCL, the collective library, detects intra-host topology itself, and its topology file must describe a single host (docs.nvidia.com).

Topograph's documentation contrasts the two layers directly: Topograph labels tell the scheduler which nodes belong together on the network, and both Topograph and the Topology Manager can be active simultaneously. A pilot should therefore confirm the node-local policy separately, because a correct fabric label cannot repair a pod whose GPU and NIC straddle NUMA nodes.

Cross-node: leaf, spine and NVLink domain

Cross-node placement concerns which servers share a leaf switch, which share a spine, and which share an NVLink domain. On GB200 NVL72 systems, NVIDIA describes 72 GPUs in a single NVLink domain with 130 TB/s of GPU communication (www.nvidia.com). The Kubernetes DRA (Dynamic Resource Allocation) driver for

NVIDIA GPUs provisions ephemeral multi-node NVLink fabrics using IMEX (dra-driver-nvidia-gpu.sigs.k8s.io) and requires Kubernetes 1.32 or later (dra-driver-nvidia-gpu.sigs.k8s.io).

The clique label that identifies an MNNVL domain is written by NVIDIA's GPU stack, not by Topograph, and is scoped to nodes inside such a domain. NVIDIA's Topograph reference states that the label cannot tell a scheduler which racks share a top-of-rack switch (raw.githubusercontent.com), so rack and leaf placement still needs fabric tier labels. The label is also refreshed on an interval rather than propagated instantly (raw.githubusercontent.com). On hardware where the GPU fabric never reports completion, the clique label is absent and the InfiniBand provider becomes the only source.

The separation matters for acceptance testing. A job can pass a node-local test (every GPU paired with the nearest NIC) and fail a fabric test (its eight nodes sit under four different leaves), and the reverse is equally possible.

Choosing the Scheduler Layer

Default Kubernetes scheduler: a preference, one pod at a time

With Topograph labels, native pod affinity can express "prefer the same leaf". Required affinity rules are hard filters, since the Pod is not scheduled unless the rule is met (kubernetes.io), whereas preferred rules are scored (kubernetes.io). Topology spread constraints have the same split: `DoNotSchedule` is hard (kubernetes.io) and `ScheduleAnyway` is soft (kubernetes.io). Topograph's Kubernetes engine page notes that the default scheduler places one pod at a time, so placement depends on where earlier pods landed (raw.githubusercontent.com), and it states that there is currently no fully network-aware scheduler.

Kubernetes is adding native alternatives. The standard scheduler evaluates Pods sequentially (kubernetes.io), but gang scheduling, in which none of the Pods are bound unless the group fits (kubernetes.io), is documented as beta since Kubernetes v1.37 and disabled by default (kubernetes.io). Native topology-aware scheduling is documented as alpha since v1.36 and disabled by default (kubernetes.io). It enforces a topology constraint strictly (kubernetes.io), allows a single constraint per PodGroup (kubernetes.io), and does not trigger preemption (kubernetes.io). Topograph's own label reference describes the same KEP as alpha in v1.37 (docs.nvidia.com), a version discrepancy between two NVIDIA-adjacent pages that a pilot should resolve against the cluster's actual Kubernetes release.

KAI Scheduler and Kueue: gang placement with levels

KAI Scheduler derives topology from node labels and its own Topology custom resource (raw.githubusercontent.com), which carries an alpha API version. A required placement will not place the workload on nodes lacking the specified label (raw.githubusercontent.com), and a preferred placement falls back to a higher-level domain. Its multi-level documentation ties PodGroup gang semantics to these constraints. NVIDIA Run:ai documentation says users may apply labels manually or use discovery tools, and that labels are applied at the Kubernetes level and managed outside Run:ai (run-ai-docs.nvidia.com).

Kueue topology-aware scheduling has been beta since Kueue v0.14 (kueue.sigs.k8s.io). It assumes node labels are set by the cloud provider or by administrators of on-premises clusters (kueue.sigs.k8s.io), which is where Topograph's Kubernetes engine fits. A required annotation forces all pods into one domain at the named

level (kueue.sigs.k8s.io), while the preferred annotation is a preference rather than a requirement (kueue.sigs.k8s.io) and escalates to the next level up when the PodSet does not fit (kueue.sigs.k8s.io).

Slurm: tree, block and per-partition topology

Slurm's default topology plugin is flat (slurm.schedmd.com), so topology-aware behavior is something you turn on. Tree topology describes leaf switches with their nodes and recommends one top-level switch (slurm.schedmd.com); users cap leaf switches with `--switches` (slurm.schedmd.com). Block topology, intended for NVLink domains, defines enforceable block sizes (slurm.schedmd.com), each a power of two larger than the prior (slurm.schedmd.com), and jobs use `--segment` (slurm.schedmd.com). An NVIDIA presentation at SchedMD's SLUG24 states that the block plugin was introduced in Slurm 23.11 (slurm.schedmd.com). Slurm 25.05 introduced multiple topologies selectable per partition (slurm.schedmd.com), and nodes can be dynamically added to and removed from topologies (slurm.schedmd.com).

Slurm itself cautions that best-fit allocation on currently free nodes can use more switches than the optimum (slurm.schedmd.com). File changes take effect upon restart, signal or reconfigure (slurm.schedmd.com), which is why the Topograph Slurm path registers a trigger that runs whenever a node goes down or comes up (docs.nvidia.com).

Volcano as a different architecture

Volcano is a separate scheduler with its own model and should not be treated as equivalent. Its HyperNode resource represents a network performance domain, typically mapped to a switch (volcano.sh). It supports a hard mode that keeps tasks in one HyperNode (volcano.sh) and a soft mode that does so as much as possible (volcano.sh). It can discover HyperNodes from data sources itself (volcano.sh), polling at a default of one hour (volcano.sh). Operators who already run Volcano should compare its discovery to Topograph's rather than stack both.

Table 2 summarizes the scheduler layer decision.

Layer	How it learns topology	Hard vs soft	Gang aware?	Stated maturity
Default kube-scheduler + affinity	Topograph node labels	Required is hard, preferred is scored	No; pods placed one at a time	Core scheduler behavior
Kubernetes native TAS and PodGroup	Node label key supplied in PodGroup	Strict only, one constraint	Yes, beta and off by default	Topology part alpha
KAI Scheduler	Node labels plus Topology CRD	Required and preferred levels	Yes, PodGroup	CRD alpha
Kueue TAS	Node labels set by admins	Required and preferred annotations	Not stated in the pages reviewed	Beta since v0.14
Slurm	`topology.conf` or `topology.yaml`	`--switches`, `--segment`	Allocation of a job's nodes is native to Slurm	Block since 23.11

Layer	How it learns topology	Hard vs soft	Gang aware?	Stated maturity
Volcano	HyperNode CRD, own discovery	Hard and soft modes	Job-level constraint on tasks	Separate project

The table supports a plain rule: if jobs must start together inside one fabric domain, a preference in the default scheduler is not enough, and the choice is between a gang-aware layer (KAI, Kueue, or native PodGroup scheduling once enabled) and Slurm. If jobs tolerate fragmentation but benefit from proximity, a preferred affinity is the lowest-risk first step.

Implementation Considerations and Process Changes

Pin prerequisites and verify permissions before running commands

The README states that Kubernetes installs need Kubernetes 1.27 or later and Helm 3.10+ or 4.x (raw.githubusercontent.com). The documented install uses the NGC Helm repository (docs.nvidia.com) followed by a `helm install` of the `topograph/topograph` chart and a `helm test` hook. A minimal sequence, to be checked against the chart README of your pinned version, is:

```
helm repo add topograph https://helm.ngc.nvidia.com/nvidia/topograph
helm install topograph topograph/topograph \
  --namespace topograph --create-namespace \
  --set provider.name=<provider> --set engine.name=k8s
helm test topograph --namespace topograph
```

Security review should precede the install because the service's design places controls on the operator. The API server does not implement built-in authentication (raw.githubusercontent.com), so access control belongs at the network layer. A NetworkPolicy has no effect unless the container network interface enforces it (raw.githubusercontent.com). The node data broker has cluster-wide node update permission in the chart's RBAC, and an optional admission policy validates that the node being modified matches the node name claim (raw.githubusercontent.com). Cloud-hosted deployments should treat instance metadata access as a credential-exposure vector. On large first reconciliations, the Kubernetes client defaults to 5 queries per second with a burst of 10 (raw.githubusercontent.com), which the chart exposes as tunables.

Cloud providers add their own read permissions: the AWS provider uses the EC2 instance topology API, the GCP provider the Compute Engine instance list, and Nebius its instance list. Grant the narrowest role the pinned provider page documents.

Stage one: simulation without production hardware

Model files can simulate discovered topology without querying a real cloud API or NetQ instance (docs.nvidia.com). A helper script creates or reuses a local kind cluster, installs KWOK, and applies the generated virtual nodes. The test provider simulates the API lifecycle itself, and the `infiniband-sim` variant replays saved `ibnetdiscover` output with no hardware or credentials. The README points to a no-cloud demo script.

Use simulation to prove three things before touching production: that your scheduler configuration consumes the label keys, that required constraints reject an oversized gang cleanly, and that removing a node from the model removes stale labels.

Stage two: controlled production pilot

A controlled pilot uses a labeled subset of nodes, one scheduler path at a time, and no change to the existing production queue. The sequence below is the minimum.

1. **Capture desired topology.** From the network team's cabling plan, list each node's leaf, spine and (for MNNVL) rack domain.
2. **Capture discovered topology.** Record Topograph output (labels, `topology.conf`, or Graph engine JSON) and independent raw evidence such as `ibnetdiscover` output.
3. **Diff.** Any node whose discovered leaf differs from desired is a finding, regardless of which side is wrong.
4. **Verify node-local policy.** Record `nvidia-smi topo -m` and the kubelet Topology Manager policy for every pilot node.
5. **Run paired jobs.** Run the same job through the baseline and the topology-aware path, alternating order, on the same node pool.
6. **Run the change test** described in the next section.

For fabric measurement, `nccl-tests` reports bus bandwidth, which is meant to reflect the speed of the hardware bottleneck (github.com) and can be compared with hardware peak independently of rank count (github.com). A multi-node build requires MPI, and the project's example launches 64 ranks over 8 nodes (github.com). Discard warmup iterations, which are not timed (github.com).

Table 3 is a pilot run record template. The cells are placeholders to be filled from your own runs; the report supplies no measured values.

Run	Scheduler path	Job and GPU count	Placement observed (leaf and domain labels)	Network path evidence	Duration	Throughput or latency	Variance across repeats
A1	Baseline, no topology rule	Identical job, N GPUs	Record per-node labels	`nccl-tests` busbw, NCCL log	Record	Record	Record
A2	Preferred affinity	Identical job, N GPUs	Record per-node labels	Same	Record	Record	Record
A3	Required level (KAI or Kueue)	Identical job, N GPUs	Record per-node labels	Same	Record	Record	Record
A4	Slurm tree or block	Identical job, N GPUs	`scontrol show topology` excerpt	Same	Record	Record	Record

The interpretive rule for *Table 3* is to compare rows only when the job, GPU count and node pool are identical, and to repeat each row enough times to show run-to-run variance before attributing any difference to placement. A difference smaller than the variance is not evidence of a gain.

Change detection, stale labels, rollback and owners

Treat the following as an acceptance checklist, each item tested by an actual change in a maintenance window or in simulation.

- **Node added or removed.** Expected to trigger regeneration. Verify new labels appear after the aggregation delay.
- **Node metadata changed.** Does not trigger the observer (docs.nvidia.com); verify you issue a manual request, and note that results are kept only for the last 100 request IDs (docs.nvidia.com).
- **Switch re-cabled with no node event.** The Slurm trigger does not detect arbitrary rewiring (developer.nvidia.com). Verify a scheduled or manual regeneration closes this gap, and compare the result with fresh `ibnetdiscover` output.
- **Selectors match nothing.** The topology never regenerates (docs.nvidia.com). Verify selectors against real node labels.
- **Continuous churn.** The queue can starve (docs.nvidia.com). Verify behavior under an autoscaling or repair storm.
- **Provider failure.** The server makes up to five generation attempts (docs.nvidia.com), so failures may not appear where you expect. Alert on API errors, not only on pod health, because the health endpoint does not test provider reachability, credentials or the last generation result (docs.nvidia.com).
- **Metrics hygiene.** The `topograph_missing_topology` gauge is only ever set, never cleared, so a nonzero reading may be stale.
- **MNNVL clique lag.** The clique label is refreshed on an interval; verify scheduling tolerates the lag.

Topograph states that its output reflects reported rather than intended topology (developer.nvidia.com). A mis-cabled but self-consistent fabric will therefore yield labels that match the wiring and disagree with the design, which is why the desired-versus-discovered diff is a separate step.

Rollback is the operator's design, because the sources reviewed document no rollback procedure. It is an inference, not a documented guarantee, that labels are inert until a scheduler rule references them, so the lowest-risk rollback is to remove or relax the scheduler rule (required to preferred, then to none) before removing the service. For Slurm, keep the previous `topology.conf` and reload it, since file changes take effect on restart or reconfigure.

Define exit criteria before the pilot begins, so that the result is not argued after the fact:

- **Label accuracy:** zero unexplained differences between desired and discovered topology on the pilot nodes.
- **Change coverage:** every change type in the checklist either triggers a refresh within the agreed window or has a documented manual step.
- **Performance:** paired-run differences exceed the measured run-to-run variance, or the pilot is recorded as inconclusive.

- **Safety:** the rollback step has been exercised at least once.

Assign owners explicitly:

- **Network or fabric team:** owns the desired topology, cabling changes and the change calendar.
- **Platform team:** owns the Topograph release, chart values, observer selectors and alerts.
- **Scheduler team:** owns KAI, Kueue, Slurm configuration and the required versus preferred policy.
- **Security team:** owns RBAC review, network exposure and provider credentials.
- **Workload owners:** own the paired-run job definitions and the acceptance sign-off.

Data Analysis and Evidence

Hardware and latency numbers that frame the problem

Per-GPU NVLink bandwidth is 900 GB/s on the fourth generation in H100 (www.nvidia.com) and 1.8 TB/s on the fifth generation in GB200 NVL72 (www.nvidia.com). Network links are slower by an order of magnitude: Quantum-2 InfiniBand switches provide 64 ports of 400 Gb/s (docs.nvidia.com), Quantum-X800 provides 144 ports of 800 Gb/s (www.nvidia.com), and ConnectX-8 supports up to 800 Gb/s (networking-docs.nvidia.com). Spectrum-X is described as scaling in two-tier leaf and spine topologies (www.nvidia.com). Converting units, 400 Gb/s is 50 GB/s and 800 Gb/s is 100 GB/s per port, so a job that crosses the fabric instead of staying inside an NVLink domain moves from a path rated in the hundreds of GB/s per GPU to one rated at 50 to 100 GB/s per port. Rated figures are not measured throughput, NVLink ratings are aggregate per GPU while network ratings are per-port line rates, so the two are not like-for-like and the order-of-magnitude contrast is indicative only, and real gaps depend on the collective and message size.

Topograph's own operating numbers are small and documented: a 15 second aggregation delay, a default Kubernetes client limit of 5 queries per second with a burst of 10 (raw.githubusercontent.com), and a result cache of 100 request IDs (docs.nvidia.com). NVIDIA's InfiniBand documentation also states that manual topology maintenance is feasible at roughly 32 nodes with a stable network and a careful operator (docs.nvidia.com), and that manual maintenance becomes the dominant source of scheduling misplacement at larger scale. Both are vendor statements rather than measurements.

What independent studies show, and what they do not

No source reviewed measures a Topograph-generated label set against a baseline. The closest evidence concerns topology-aware placement in general, and the setups differ.

- **Production LLM training (peer-reviewed).** The Arnold scheduler, presented at NeurIPS 2025 (arxiv.org), reports a 10.6% end-to-end gain on more than 9,600 GPUs (arxiv.org), an average speedup of 5.7% and 10.6% at the two scales studied (arxiv.org), and a reduction of up to 1.67 times in communication-group spread in simulation (arxiv.org).
- **Sensitivity to spread.** The same study measured collective bandwidth falling by up to 17% for collectives and 70% for point-to-point operations as groups spanned more units (arxiv.org), yet found negligible variation of 0.3% within a single unit (arxiv.org). The gain comes from avoiding wide spread, not from fine leaf-level tuning.

- **Communication-aware Slurm allocation.** A 2020 workshop paper reports a 9% average reduction in execution time for communication-intensive jobs (dl.acm.org) and a 31% reduction in average wait time on three supercomputer job logs (dl.acm.org); these are log-driven results on older systems.
- **Simulation.** A network-sensitive GPU scheduler reports up to a 69% makespan improvement in a trace-driven simulator (arxiv.org), which is not a hardware result.
- **Interference.** An SC18 study measured up to a 114% run time increase in the worst case for fixed placements under inter-job interference (www.cs.umd.edu).
- **Counter-evidence.** A 2017 poster on a 3-level fat-tree with a 2:1 taper found minimal placement impact for the applications tested (www.sc17.supercomputing.org) and noted that added locality constraints can reduce system utilization (www.sc17.supercomputing.org).

The pattern is consistent with a conditional hypothesis: placement matters when jobs are large, communication-bound and the fabric is oversubscribed or shared, and matters little when the fabric is non-blocking relative to the job or the jobs are small. The authors of the studies above did not test Topograph, and their hardware differs from yours. NVIDIA's blog states a qualitative problem (poor placement fragments topology domains), but the text reviewed contains no measured throughput figure for Topograph. For a buyer, the defensible position is that the expected size of any gain is unknown until paired runs on the buyer's fabric are complete.

Meta's engineering description of its RoCE training network explains a related design goal, reducing cross-zone traffic and collective completion time through placement (engineering.fb.com), which supports the mechanism but not a number for other clusters.

Implications and Future Directions

The adoption gate is operational, not technical. Installing the service is a short exercise; the harder questions are whether the provider's reported topology is accurate, whether refresh covers your change events, and whether a scheduler layer exists that can act on the output. A cluster with a single flat fabric, small jobs and a non-blocking network has little to gain based on the counter-evidence above.

Kubernetes is moving the scheduler layer. Gang scheduling reached beta in v1.37 (kubernetes.io) but is off by default, and native topology awareness remains alpha. If a cluster's Kubernetes version and feature gates will not reach that state during the planning horizon, KAI or Kueue is the practical gang layer today. Decisions should be tied to the features actually enabled in your own cluster.

Slurm keeps evolving. Slurm 26.05 introduced ring topologies (slurm.schedmd.com), so the set of topology types a generator must emit is not fixed. **Slurm-on-Kubernetes deserves its own test.** NVIDIA documents Slinky integration that discovers block topology using DRA and GPU Operator labels (developer.nvidia.com). Because the DRA provider sees only NVLink partitions, a Slinky cluster that spans partitions needs a provider that sees the fabric too.

Independent validation will matter more. NetQ includes topology validation against a user-provided blueprint (docs.nvidia.com), and UFM can compare the current topology with a master topology (networking-docs.nvidia.com). Pairing such drift checks with Topograph output is a stronger control than either alone, and it also gives auditors evidence of intended versus reported wiring.

The desired-versus-discovered worksheet, the paired-run protocol and the owner matrix give structure, but the evidence that decides adoption has to come from the buyer's own cluster.

Frequently Asked Questions (FAQs)

What is NVIDIA Topograph, and does it schedule jobs?

It discovers cluster topology and translates it into a scheduler's format. It sits between topology sources and schedulers and replaces neither. Placement is performed by Kubernetes, KAI, Kueue or Slurm.

How does Topograph work with Kubernetes topology-aware scheduling?

The Kubernetes engine writes `fabric.topograph.run/tier-N` and accelerator-domain labels. The default scheduler can use them in affinity rules as a preference, while KAI and Kueue can apply required levels to a gang.

How does Topograph relate to Slurm `topology.conf`?

The Slurm engine generates tree or block topology. A node up or down trigger can call it, which by design reacts to node events rather than to cabling changes. Slurm 25.05 added per-partition topologies.

Does it work on bare-metal InfiniBand without MNNVL?

Yes through the InfiniBand provider, which uses `ibnetdiscover` and is the only network topology source when the clique label is absent. It is single-region only.

Which node labels does it write, and what are NFD resources?

Fabric tier labels and accelerator domain labels. The NFD engine instead publishes NodeFeatureGroup resources and is only for consumers of NFD.

How can operators confirm the labels match the wiring after a change?

Diff labels against fresh `ibnetdiscover` output and the cabling plan after a controlled change, as in the checklist above. Regeneration is triggered by events such as node adds and deletes.

What does the default scheduler lack compared with KAI or Kueue?

It places one pod at a time, so a required gang-wide domain needs a gang-aware layer, where KAI ties PodGroup gang semantics to topology constraints.

Will it improve throughput or lower cost?

That cannot be promised from public sources. Published placement studies show effects from negligible to double digits, under different conditions. Measure with paired runs.

Conclusion

Topograph is best understood as a topology reporting layer with scheduler-specific outputs. Its value is bounded by three factors: the accuracy of the provider's reported topology, the coverage of its refresh triggers, and the existence of a scheduler path that can act on the result. NVIDIA says as much about reported versus intended topology.

For a private cluster, the pilot decision reduces to a short test. Pin a release and record the provider list from that tag. Choose the provider by fabric: NetQ where deployed, InfiniBand otherwise, and do not substitute the DRA provider when the fabric between partitions matters. Choose the scheduler layer by whether jobs need a hard gang-wide domain or only a preference. Run the change-detection checklist and the desired-versus-discovered diff before the paired performance runs. If the diff is clean, the labels track a deliberate wiring change, and paired runs show a difference larger than their variance, adoption is justified by the buyer's own evidence. If any of those fail, the cluster has learned something useful about its topology data before it learned it from a stalled job.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://docs.nvidia.com/topograph/architecture/config-and-api.md
2. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/engines/slinky.md
3. https://raw.githubusercontent.com/NVIDIA/topograph/main/README.md
4. https://developer.nvidia.com/blog/topology-aware-workload-scheduling-with-nvidia-topograph
5. https://api.github.com/repos/NVIDIA/topograph/releases
6. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/engines/k8s.md
7. https://raw.githubusercontent.com/NVIDIA/KAI-Scheduler/main/docs/topology/README.md
8. https://kueue.sigs.k8s.io/docs/concepts/topology_aware_scheduling
9. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/reference/node-labels.md
10. https://docs.nvidia.com/topograph/architecture/config-and-api
11. https://docs.nvidia.com/topograph/engines/slurm.md
12. https://docs.nvidia.com/topograph/architecture/architecture
13. https://arxiv.org/abs/2509.15940
14. https://www.sc17.supercomputing.org/SC17%20Archive/src_poster/poster_files/spost153s2-file2.pdf
15. https://docs.nvidia.com/topograph/getting-started/overview
16. https://sc25.supercomputing.org/proceedings/workshops/workshop_pages/ws_canolt102.html
17. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/providers/infiniband.md
18. https://docs.nvidia.com/topograph/providers/infini-band.md
19. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/providers/netq.md
20. https://docs.nvidia.com/topograph/providers/dra.md
21. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/providers/aws.md

22. https://docs.nvidia.com/topograph/providers/crusoe.md
23. https://raw.githubusercontent.com/NVIDIA/topograph/main/docs/engines/nfd.md
24. https://slinky.schedmd.com/slurm-operator/v1.2.0/usage/topology.html
25. https://docs.nvidia.com/deploy/nvidia-smi/index.html
26. https://kubernetes.io/docs/tasks/administer-cluster/topology-manager
27. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html
28. https://www.nvidia.com/en-us/data-center/gb200-nvl72
29. https://dra-driver-nvidia-gpu.sigs.k8s.io/docs
30. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node
31. https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints
32. https://kubernetes.io/docs/concepts/scheduling-eviction/podgroup-scheduling
33. https://kubernetes.io/docs/concepts/scheduling-eviction/gang-scheduling
34. https://kubernetes.io/docs/concepts/workloads/workload-api/topology-aware-scheduling
35. https://docs.nvidia.com/topograph/reference/node-labels-and-annotations
36. https://run-ai-docs.nvidia.com/self-hosted/platform-management/aiinitiatives/resources/topology-aware-scheduling
37. https://slurm.schedmd.com/slurm.conf.html
38. https://slurm.schedmd.com/topology.html
39. https://slurm.schedmd.com/sbatch.html
40. https://slurm.schedmd.com/topology.conf.html
41. https://slurm.schedmd.com/SLUG24/NVIDIA-Craig_Tierney.pdf
42. https://volcano.sh/docs/keyfeatures/networktopologyaware
43. https://docs.nvidia.com/topograph/getting-started/install-on-kubernetes.md
44. https://docs.nvidia.com/topograph/architecture/modeling-and-api-simulation
45. https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md
46. https://github.com/NVIDIA/nccl-tests
47. https://www.nvidia.com/en-us/data-center/h100
48. https://docs.nvidia.com/networking/display/QM97X0PUB/Introduction
49. https://www.nvidia.com/en-us/networking/products/infiniband/quantum-x800
50. https://networking-docs.nvidia.com/connectx8hw/introduction
51. https://www.nvidia.com/en-us/networking/spectrumx
52. https://docs.nvidia.com/topograph/providers/infini-band
53. https://arxiv.org/pdf/2509.15940
54. https://dl.acm.org/doi/10.1145/3409390.3409410
55. https://arxiv.org/abs/2401.16492
56. https://www.cs.umd.edu/~bhatele/pubs/pdf/2018/sc2018b.pdf
57. https://engineering.fb.com/2024/08/05/data-center-engineering/roce-network-distributed-ai-training-at-scale
58. https://kubernetes.io/blog/2026/09/08/kubernetes-v1-37-advancing-workload-aware-scheduling

59. https://developer.nvidia.com/blog/running-large-scale-gpu-workloads-on-kubernetes-with-slurm
60. https://docs.nvidia.com/networking-ethernet-software/cumulus-netq-53/Validate-Operations/Validation-Checks
61. https://networking-docs.nvidia.com/ufmsdnappswum/4183/network-map

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.