

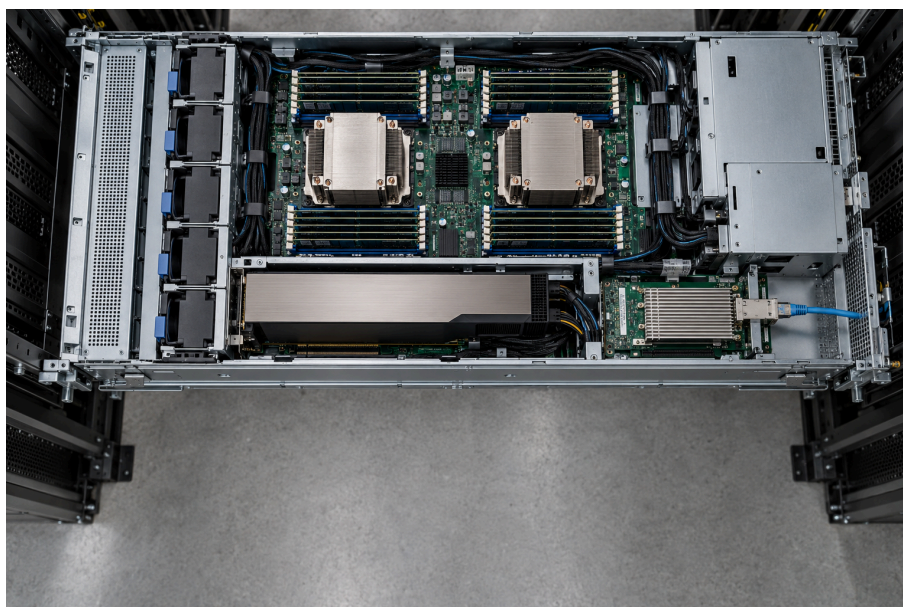


Kubernetes 1.37 Pod-Level Resource Managers: GPU NUMA Gate

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 · 2026-10-27 · [kubernetes 1.37 pod-level resource managers](#) · [podlevelresource managers](#) · [kubernetes pod-level resources](#) · [gpu numa alignment](#) · [cpu manager](#) · [memory manager](#) · [topology manager](#) · [gpu kubernetes](#) · [sidecar resources](#)

Original article: <https://gpusmith.com/articles/en/kubernetes-pod-level-resource-managers-gpu-numa>



In this article

1. [Executive Summary](#)
2. [Introduction and Background](#)
3. [Key Changes](#)
4. [Implementation Considerations and Process Changes](#)
5. [GPU NUMA Locality and the Canary Manifest](#)
6. [Admission, Observability and Rollback](#)
7. [Data Analysis and Evidence](#)
8. [Implications and Future Directions](#)
9. [Conclusion](#)

Executive Summary

Kubernetes 1.37 Pod-Level Resource Managers is a **Beta, default-off** kubelet feature that lets Topology Manager, CPU Manager and Memory Manager use a Pod's central processing unit (CPU) and memory budget. (kubernetes.io) For graphics processing unit (GPU) services with sidecars, its attraction is a hybrid allocation: exclusive resources for the main application, with helpers sharing the remaining Pod pool. The official tutorial illustrates a **4-CPU, 4Gi Pod** containing an exclusive **2-CPU, 2Gi main container** and shared helpers. (kubernetes.io)

The adoption test must extend beyond that allocation. The feature does not establish that the selected GPU and network interface controller (NIC) are local to the resulting CPU and memory placement. NVIDIA's device-plugin implementation and Dynamic Resource Allocation (DRA) documentation expose device locality through their own discovery paths. DRA can constrain selected GPUs to a shared published non-uniform memory access (NUMA) node, but that does not demonstrate CPU pinning or GPU/NIC coallocation. Compare allocated device identities against host topology, effective CPU sets and actual resident pages.

(raw.githubusercontent.com) (dra-driver-nvidia-gpu.sigs.k8s.io) (cdn.kernel.org) (man7.org)

Restrict initial deployment to a canary node pool and resolve the supported software stack before production adoption. NVIDIA's fetched **Network Operator v26.7.0** support page lists Kubernetes **1.32 through 1.36**, so it does not establish support for this target release. (docs.nvidia.com) Managed-service CPU or topology settings likewise do not, by themselves, confirm access to the gate. (learn.microsoft.com) Observability and recovery are acceptance requirements: **1.37** checkpoints improve downgrade parsing, but downgrading to **1.36** loses active Pod-level assignments. (kubernetes.io) Drain and recreate workloads through the tested recovery process.

Finally, placement is evidence of an allocation, not a benchmark result. Hold model, workload shape, offered load, warmup and cache state constant; measure service latency and throughput alongside CPU use and memory locality. PyTorch documents synchronization and repetition requirements, while vLLM exposes load controls and warns about cache reuse in repeated benchmarks. No feature-specific throughput gain, cost reduction or measured sidecar saving is claimed here. **Adopt, restrict or defer** according to the supported stack's observed admission, allocation, monitoring, performance and rollback behavior. (docs.pytorch.org) (docs.vllm.ai)

Introduction and Background

A GPU service with a model server, data loader, proxy and telemetry helpers needs more than a device allocation. Its host threads and memory pages must also land where the application expects. **Non-uniform memory access (NUMA)** describes a machine in which locality matters to memory access. A topology tool describes that structure; it does not establish application speed. The hwloc documentation explicitly separates the platform's structural model from a performance model. (www.open-mpi.org)

The practical question is whether a shared budget can replace unnecessarily isolated helper allocations while preserving the main application's locality. GPU Smith's perspective in this report is an engineering acceptance decision: enable the feature only where configuration, placement, accounting and recovery have been demonstrated. It is a framework for evaluating a node pool, rather than a claim of measured savings.

Kubernetes 1.37 pod-level resource managers extend kubelet placement decisions to a Pod's `.spec.resources`. The official announcement describes a **Beta, disabled-by-default** feature affecting Topology Manager, CPU Manager and Memory Manager. It is distinct from merely allowing a Pod-level resource declaration. (kubernetes.io)

Device placement remains another part of the system. NVIDIA's device plugin documentation describes specific CPU Manager interoperability settings, while the network device plugin separately advertises network resources. Neither should be assumed configured merely because a Pod has a CPU and memory budget. (github.com) (raw.githubusercontent.com)

This report uses documentation retrieved on **October 27, 2026**, with version-specific evidence identified where relevant. It reports no laboratory benchmark of the feature. The acceptance procedure therefore combines a topology inventory with an identical baseline/canary workload. Framework documentation reinforces that timing requires warmup, controlled thread counts and accelerator synchronization, while serving benchmarks require a defined arrival process. (docs.pytorch.org) (docs.vllm.ai)

Key Changes

A Pod budget becomes an allocation input

The important distinction is between **PodLevelResources**, which permits the budget, and **PodLevelResourceManagers**, which lets resource managers consume it. The feature-gate reference lists the former as Beta and enabled by default since **1.34**, and the latter as Beta and disabled by default in **1.37**. (kubernetes.io) The tutorial requires both gates for the control plane and workers. (kubernetes.io)

For this use case, the meaningful improvement is allocation flexibility. A performance-sensitive main container can have an exclusive allocation while helpers consume a shared remainder. An operator should evaluate that model against the application's actual worker and helper behavior. Application runtimes can impose thread affinity themselves, so a larger allowed CPU set does not prove that useful work spreads across it. (www.glennklockwood.com)

Pod scope and container scope answer different questions

Pod scope reserves and aligns a common Pod allocation, then partitions it into exclusive slices and a Pod-isolated shared pool. **Container scope** allocates eligible exclusive containers from the node while other containers use the node shared pool. The latter does not provide the same isolated helper pool, even though a Pod budget constrains aggregate consumption. (kubernetes.io)

This choice should be made at node-pool design time, with the application's locality objective written down. If helpers handle critical preprocessing, confining the whole Pod may be useful. If only the main process requires isolation, a node-shared helper model may be acceptable. In either case, an allowed-node mask and actual resident memory are separate observations: the kernel exposes effective permitted nodes, while process tools inspect page distribution. (cdn.kernel.org) (man7.org)

The feature is not a GPU or NIC binding policy

Pod-level budgets cover **CPU, memory and hugepages**, rather than a GPU or network interface allocation. NVIDIA's pinned device-plugin implementation supplies topology information only when it discovers a NUMA node. A missing hardware locality value therefore cannot safely be treated as agreement with the kubelet's placement. (kubernetes.io) (raw.githubusercontent.com)

Dynamic Resource Allocation (DRA) supplies another device-selection path. NVIDIA's DRA documentation describes a NUMA attribute and a constraint for matching the published NUMA value among selected GPUs. That is a device-selection constraint; it does not demonstrate CPU pinning or GPU/NIC coallocation. The acceptance test must compare the resulting allocations rather than infer them from the presence of a constraint. (dra-driver-nvidia-gpu.sigs.k8s.io)

For network interfaces, verify what is actually advertised. The upstream single-root input/output virtualization (**SR-IOV**) device plugin has an **excludeTopology** setting that suppresses NUMA advertisement. Its own documentation also separates device allocation from driver binding and network plumbing. These are concrete reasons to inspect the deployed configuration rather than equate device visibility with aligned execution. (raw.githubusercontent.com)

Implementation Considerations and Process Changes

Establish eligibility before changing a node

The provider context includes **Amazon Elastic Kubernetes Service (EKS)**, **Azure Kubernetes Service (AKS)** and **Google Kubernetes Engine (GKE)**. *Table 1* combines feature eligibility with the scope decision. The provider rows show why exposed settings alone are insufficient evidence of gate support.

Decision or prerequisite	Documented behavior	Acceptance requirement
CPU and memory policies	CPU static; Memory Static. The feature is Linux-only; Windows Pod allocations are a no-op. (kubernetes.io)	Capture effective configuration and startup logs.
Topology policy and scope	Tutorial permits single-numa-node, restricted or best-effort; it excludes none. (kubernetes.io)	Choose strict single-node alignment for a canary whose locality contract requires it.
Allocation scope	Pod scope has exclusive slices and an isolated shared remainder; container scope has independent exclusive allocations and node-shared helpers. (kubernetes.io)	Choose whether helpers require Pod isolation or may share the node pool.
AKS controls	Published controls include static CPU and topology policy; the configuration interface exposes a selected subset. (learn.microsoft.com)	Obtain confirmation for gate, scope and memory policy before rollout.
GKE controls	Documentation exposes Memory Manager and separate topology policy/scope settings. (docs.cloud.google.com)	Confirm the chosen interface, machine type and gate support.

The table describes necessary checks, not a certification matrix. A provider's configuration surface can expose individual managers without documenting this Beta feature. Preserve a distinction between **upstream eligibility**, **distribution configurability** and **the tested node image**. EKS platform versions, for example, describe control-plane flags and patch versions together, rather than just a minor release label. (docs.aws.amazon.com)

Static CPU policy requires a nonzero system CPU reservation. Static Memory Manager also requires NUMA reservations whose total agrees with the kubelet's system, Kubernetes and eviction memory reservations. These are node-specific inputs, so copying another machine's reservation map is not a valid configuration procedure. (kubernetes.io)

Verify exact versions and the supported integration path

The upstream releases page retrieved for this report listed **1.37.1**, dated **September 15, 2026**, as the latest patch for that branch. (kubernetes.io) EKS separately listed **1.37.0 / eks.4**, released **October 1, 2026**. (docs.aws.amazon.com) These are distinct published records, not interchangeable descriptions of the binary running on a worker. Record the actual kubelet version, node image and provider platform.

A second version boundary concerns networking. NVIDIA's **Network Operator v26.7.0** platform page lists Kubernetes **1.32 through 1.36**. That page does not establish vendor support for this report's target release. A successful upstream canary would not close that support question. (docs.nvidia.com)

Use the following deployment record:

- **Control plane:** Record version, platform and approved feature-gate configuration. EKS documents platform capabilities separately. (docs.aws.amazon.com)
- **Workers:** Record the node image and selected kubelet settings. AKS exposes a subset rather than arbitrary configuration. (learn.microsoft.com)
- **Memory interface:** Preserve the casing and schema of the selected provider interface. GKE documents its Memory Manager configuration separately. (docs.cloud.google.com)
- **GPU path:** Record plugin or DRA driver version and deployment values. NVIDIA's operator documentation identifies the driver it manages. (docs.nvidia.com)
- **CPU interoperability:** Check the deployed plugin's CPU Manager options, including privilege requirements, against its documentation. (github.com)
- **Network path:** Record the NIC plugin and whether topology advertisement has been disabled. (raw.githubusercontent.com)

For self-managed nodes, **Talos v1.13** documents kubelet extra flags and configuration overrides, with restrictions on some fields. This is a configuration mechanism, not a statement of support for this gate or target Kubernetes release. Persist and validate the desired state through the chosen operating system's supported interface. (docs.siderolabs.com)

No inspected EKS, AKS or GKE page explicitly confirmed customer enablement of PodLevelResourceManagers. Their documented CPU and topology controls remain useful evidence for the surrounding configuration, but support for the gate remains a separate acceptance item. (learn.microsoft.com) (docs.cloud.google.com)

GPU NUMA Locality and the Canary Manifest

Build a locality oracle before changing placement

A **locality oracle** is the operator's recorded mapping from the allocated GPU and NIC to expected CPU and memory nodes. It is a proposed acceptance artifact, not a Kubernetes component. Start with the physical host inventory, then repeat the comparison for the devices actually assigned to the canary.

For NVIDIA hardware, `nvidia-smi topo -m` reports GPU/NIC relationships and CPU/memory affinities. The nearest CPU NUMA node, nearest memory NUMA node and a GPU's own NUMA identifier are different properties; preserve the command used with the result. (docs.nvidia.com)

A practical inventory uses commands such as:

```
nvidia-smi --query-gpu=uuid,pci.bus_id --format=csv
nvidia-smi topo -m
nvidia-smi topo -C -i <gpu-index>
nvidia-smi topo -M -i <gpu-index>
ethtool -i <network-interface>
cat /sys/class/net/<network-interface>/device/numa_node
cat /sys/bus/pci/devices/<normalized-pci-address>/numa_node
numactl --hardware
lstopo -p
```

Record GPU universally unique identifiers (UUIDs) and Peripheral Component Interconnect (PCI) addresses with the inventory. GPU topology commands come from NVIDIA's reference; the NIC-to-PCI and NUMA checks are documented in Intel's configuration guide. Linux documents a PCI NUMA value of **-1** as unknown, not as node zero. Normalize PCI address formatting before comparing records. (docs.nvidia.com) (cdrdv2-public.intel.com) (www.kernel.org)

Also capture the transport's actual device selection. **Unified Communication X (UCX)** can restrict network devices with `UCX_NET_DEVICES`, and its informational logging identifies selected transports and devices. A host-level local NIC mapping is insufficient if the application communicates through a different interface. (openucx.readthedocs.io)

Keep the CPU numbering convention explicit. `hwloc` logical object indexes differ from operating-system indexes, and taskset masks refer to logical CPUs. Simultaneous multithreading (**SMT**) can put multiple logical CPUs on one physical core. Consequently, counting CPU IDs is not a physical-core capacity calculation. (www.open-mpi.org) (man7.org) (kernel.org)

Table 2 specifies what the acceptance record must reconcile. Fill it with observed values; no illustrative CPU ID below is presented as a measurement.

Object	Evidence to capture	Required comparison
Allocated GPU	UUID, PCI address, affinity and published device topology. (docs.nvidia.com) (raw.githubusercontent.com)	Allocated device identity agrees with the host inventory.

Object	Evidence to capture	Required comparison
Allocated NIC	Interface, PCI bus address and NUMA node. (cdrdv2-public.intel.com)	Chosen communication interface matches the intended local path.
DRA device attribute	ResourceSlice NUMA attribute; absence remains unknown. (dra-driver-nvidia-gpu.sigs.k8s.io)	Nonnegative published locality agrees with host sysfs.
Main CPU allocation	PodResources IDs, effective cpuset, thread masks and operating-system (OS) topology. (cdn.kernel.org) (man7.org)	Granted CPUs and application threads satisfy the locality contract.
Memory placement	Effective allowed nodes plus resident-page distribution. (cdn.kernel.org) (man7.org)	Allowed nodes and observed pages are evaluated separately.
Accounting	Full Pod allocation and exclusive-container subsets. (kubernetes.io)	Shared pool is a set difference, not an additional allocation.

This reconciliation prevents topology labels from becoming circular evidence. For example, firmware supplies the initial PCI NUMA value, and Linux permits overrides. Retain the provenance of any correction. AMD operators can use ROCm SMI's device NUMA query in the same inventory design; that API is not evidence of measured application performance. ([www.kernel.org](#)) ([rocm.docs.amd.com](#))

When recording that inventory, preserve device identities independently of a utility's display order. A vendor NUMA query identifies a locality property, while a topology viewer's OS-versus-logical numbering identifies how CPU masks must be interpreted. Retain both conventions in the acceptance artifact so another engineer can reproduce the comparison. ([rocm.docs.amd.com](#)) ([www.open-mpi.org](#))

Use a small, explicit hybrid canary

The prepared node configuration must explicitly select the intended scope. The following fragment belongs in an otherwise valid, node-specific kubelet configuration with the required CPU and memory reservations already established; it is not a complete replacement configuration. Its manager and gate settings follow the official tutorial. ([kubernetes.io](#))

```
featureGates:
  PodLevelResources: true
  PodLevelResourceManagers: true
cpuManagerPolicy: static
memoryManagerPolicy: Static
topologyManagerPolicy: single-numa-node
topologyManagerScope: pod
```

The following allocation-only canary follows the official tutorial's resource structure. The Pod reserves **4 CPU units and 4Gi**; the main container receives an exclusive **2 CPU units and 2Gi**; restartable helpers share the remainder. The images are demonstration placeholders, so this manifest tests allocation rather than GPU throughput. ([kubernetes.io](#))

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: pod-numa-allocation-canary
spec:
  nodeSelector:
    example.com/pod-managers: "canary"
  resources:
    requests:
      cpu: "4"
      memory: "4Gi"
    limits:
      cpu: "4"
      memory: "4Gi"
  initContainers:
  - name: metrics
    image: registry.k8s.io/pause:3.9
    restartPolicy: Always
  - name: logging
    image: registry.k8s.io/pause:3.9
    restartPolicy: Always
  containers:
  - name: main
    image: registry.k8s.io/pause:3.9
    resources:
      requests:
        cpu: "2"
        memory: "2Gi"
      limits:
        cpu: "2"
        memory: "2Gi"

```

The Pod budget must satisfy **Guaranteed quality of service (QoS)** eligibility. For an exclusive main-container slice, CPU and memory requests must equal their respective limits, with a positive integer CPU request. This eligibility rule matters when converting an existing fractional-CPU service; moving its budget to the Pod does not make every container exclusive. (kubernetes.io)

Label only the prepared canary pool with the example selector. For the workload test, replace the placeholder with the exact baseline application and helper images. Retain its existing GPU request or DRA claim and network attachment. Do not add a second GPU allocation mechanism merely to enable the CPU/memory feature; validate the deployed device path's own configuration instead. (github.com) (docs.nvidia.com)

The baseline-to-canary change should be reviewable as a manifest diff: move the selected budget to `.spec.resources`, retain the main container's eligible exclusive slice, and change helpers to the shared model only where intended. Keep model, batch/concurrency settings and input/output distributions fixed. vLLM documents separate token-length and offered-load controls, which must stay comparable across runs. (docs.vllm.ai)

For diagnostic interpretation, keep CPU binding distinct from memory policy: `numactl --physcpubind` names CPUs, while `--membind` constrains memory allocation and can fail if the selected nodes lack capacity. These controls explain the observations; the acceptance procedure should inspect the kubelet's resulting limits rather

than silently override them inside the application. (man7.org)

Before treating the CPU mask as proof of placement, inspect every relevant thread. `taskset -a` covers all tasks associated with a process; an affinity setting alone also does not prove immediate migration. When investigating interference, distinguish the application's thread masks from the CPUs granted to its Linux control group (cgroup). (man7.org)

Admission, Observability and Rollback

Test failure paths as deliberately as the happy path

Table 3 is a proposed acceptance grid. It combines documented failure behavior with tests the operator must execute locally; a checked documentation row is not a completed hardware experiment.

Test	Expected result or observation	Release decision
Empty shared remainder	Exhaust CPU, then memory, with exclusive slices while retaining a shared helper. Admission should reject the configuration. (kubernetes.io)	Reject a rollout whose admission behavior differs.
Local capacity mismatch	Under strict single-node topology, demand that cannot fit the required NUMA placement should fail admission. (kubernetes.io)	Preserve the event and verify replacement behavior.
Unknown device locality	Missing or negative host locality must not be interpreted as an aligned allocation. (www.kernel.org) (dra-driver-nvidia-gpu.sigs.k8s.io)	Defer topology-dependent admission acceptance.
Helper pressure	Measure helper CPU time, throttling and main-request latency during a burst. CPU time comes from cgroup accounting. (cdn.kernel.org)	Reject a pool that passes idle tests but misses its workload objectives.
Network support	The fetched operator support range ends before the target Kubernetes release. (docs.nvidia.com)	Resolve support before enabling the production pool.
Restart and rollback	Capture allocation state before and after helper and kubelet restarts; account for documented checkpoint migration behavior. (kubernetes.io)	Require reconciled assignments and an exercised drained recovery.

Interpret a failure at the layer that produced it. An application programming interface (API)-admitted Pod is not proof of successful kubelet topology admission, and an admitted allocation is not proof of acceptable service latency. Linux affinity and memory tools provide different evidence from a request-duration histogram. Preserve all of them against the same Pod identity and node. (man7.org) (prometheus.io)

Schedule workload canaries through their normal controller so replacement behavior is explicit. Include a test of the owning controller's response to admission failure, rather than relying indefinitely on a bare diagnostic Pod.

Reconcile allocations without double-counting

Under pod scope, **PodResources** top-level `cpu_ids` and memory report the complete Pod allocation; exclusive containers report subsets, while shared containers leave those fields empty. Shared CPU IDs are the Pod set minus the union of exclusive-container sets. Under container scope, top-level fields remain empty. A collector must understand the selected scope. (kubernetes.io)

An empty helper CPU list therefore does not mean it has no CPU access. Verify its effective cgroup CPU set and memory-node mask. Then inspect actual resident pages with `numastat -p` or `numa_maps`. The kernel's permitted-node set is an allocation boundary, while resident-page observations describe where memory currently resides. (cdn.kernel.org) (man7.org)

Require these observability checks:

- **Allocation counters:** Collect `resource_manager_allocations_total` and `resource_manager_allocation_errors_total`, preserving source labels for Pod and node pools. (kubernetes.io)
- **Assignment types:** Inspect `resource_manager_container_assignments` for `node_exclusive`, `pod_exclusive` and `pod_shared`. (kubernetes.io)
- **Restart-aware rates:** Treat counters as cumulative and calculate rates before aggregation, preserving reset detection. (prometheus.io)
- **Latency distributions:** Keep histogram bucket definitions identical and retain sample counts; quantiles are estimates whose precision depends on the histogram. (prometheus.io)
- **Consumer compatibility:** Demonstrate that the deployed collector reconciles Pod and container allocations rather than adding overlapping sets.
- **Thread evidence:** Retain thread masks alongside cgroup masks when runtimes impose affinity. (www.glennklockwood.com)

Drain before changing the allocation regime

The checkpoint distinction matters. In **1.36**, active feature use writes a V3 format that older kubelets cannot parse; downgrade or gate disablement can require recovery. In **1.37**, V4 embeds the standard V2 structure, allowing a downgrade to restore standard container allocations, but a **1.36** kubelet loses active Pod-level assignments. Forward-compatible parsing is not preservation of the Pod allocation contract. (kubernetes.io)

Use a rehearsed, node-specific procedure:

- **Freeze rollout:** Stop new canary admissions and preserve configuration, allocation records and application evidence.
- **Evacuate:** Cordon and drain through the site's disruption and capacity process before changing policies or versions.
- **Stop kubelet:** Apply the documented policy-change procedure to a drained node. (kubernetes.io)
- **Handle state:** Follow the incompatible-checkpoint recovery described above only after evacuation and with kubelet stopped.
- **Restore configuration:** Reinstate the approved baseline configuration and image through the distribution's supported mechanism. (learn.microsoft.com)
- **Recreate and verify:** Recheck effective CPU sets, allowed nodes, resident pages and device identities before returning capacity to service. (cdn.kernel.org) (man7.org)

Require the rollback record to include recreated workloads and reconciled assignments. A kubelet process that starts successfully is only the beginning of the recovery check.

Data Analysis and Evidence

The quantitative question is **reserved capacity versus useful workload output**, not whether a NUMA label looks correct. CPU IDs, interval-average CPU use, memory residency and service performance are separate measurements. hwloc's distinction between a structural and a performance model is a useful boundary for the entire analysis. (www.open-mpi.org)

Calculate reservation slack without inventing savings

(Hypothetical Example) Assume two sidecars previously reserved one physical core each, while identical steady-state intervals measured **0.15** and **0.25 core-equivalents** of CPU use. Assume the proposed shared allocation reserves one physical core. Assume SMT is disabled for this fixture, so the reservation and usage units are directly comparable. These are explicit analytical assumptions, not benchmark observations.

- **Baseline reservation:** 2 physical cores.
- **Observed average use:** $0.15 + 0.25 = 0.40$ core-equivalents.
- **Baseline reservation slack:** $2.00 - 0.40 = 1.60$ core-equivalents.
- **Candidate reservation slack:** $1.00 - 0.40 = 0.60$ core-equivalents.
- **Reservation reduction:** 1.00 physical core, conditional on peak and latency acceptance.
- **Normalization:** Establish physical-core membership before converting CPU IDs into cores. (kernel.org) (www.open-mpi.org)

This arithmetic measures a reservation difference under the stated assumptions; it does not establish recoverable CPU capacity or lower GPU cost. For real data, derive average CPU consumption from the change in cgroup CPU usage over elapsed wall time, preserving the time units. If aggregating hierarchical cgroups, avoid counting parent usage and the same descendants again. (cdn.kernel.org)

Memory accounting needs comparable care. Record resident memory by node and memory type, then define a local-byte fraction for the intended application working set. Do not combine ordinary and huge-page counts without conversion. `memory.numa_stat` exposes byte quantities, while `numa_maps` page counts require attention to the page type. (cdn.kernel.org) (man7.org)

Compare identical work under controlled load

A proposed experiment should change the allocation model while holding the following controls constant:

- **Application identity:** Same model, weights, precision, runtime and container digests.
- **Workload shape:** Same request dataset and input/output token distributions. (docs.vllm.ai)
- **Offered load:** Same request rate, concurrency cap and arrival variability. (docs.vllm.ai)
- **Warmup:** Apply equivalent initialization and cache preparation before recording results. (docs.pytorch.org)
- **Timing:** Synchronize asynchronous accelerator work or use correctly synchronized device events for kernel timing. (docs.pytorch.org)
- **Repetition:** Use repeated measurements to quantify variation rather than selecting a favorable run. (docs.pytorch.org)

- **Cache state:** Control prompt-prefix reuse; vLLM warns that reused cached prompts can inflate throughput. (docs.vllm.ai)
- **Communication:** Keep collective sizes and warmup settings constant when using NVIDIA Collective Communications Library (NCCL) tests. (github.com)

Report inference **time to first token (TTFT)**, **inter-token latency (ITL)**, **time per output token (TPOT)**, completed request throughput and errors at the same load. These latency measures are not interchangeable: vLLM measures streamed-output intervals separately from its per-request TPOT calculation. For training, pair step-time measurements with the actual data pipeline and collective workload. (docs.vllm.ai) (github.com)

Distinguish throughput-only operation from a service constrained by latency. MLCommons' scenario guidance makes the workload model explicit, and its endpoint interface distinguishes Offline throughput from Online throughput that must meet latency targets. A placement improvement at an unconstrained saturation point is insufficient evidence for an interactive service. (docs.mlcommons.org) (endpoints.mlcommons.org)

For aggregate tail latency, preserve comparable histograms rather than averaging per-instance summary quantiles. Prometheus explicitly warns that averaging summary quantiles is statistically invalid. Report run duration, observation count, bucket definitions and variation with the result; this report imposes no universal test duration or improvement threshold. (prometheus.io)

Implications and Future Directions

The adoption decision should have three outcomes: **adopt for the tested workload**, **restrict to a dedicated pool**, or **defer**. Passing configuration and topology checks establishes eligibility. Passing controlled workload tests establishes fitness for the measured operating conditions. Neither automatically establishes fitness for another server design or application shape.

A dedicated pool is useful when the workload's affinity contract differs from general-purpose services. Keep the pool's desired configuration in the distribution's supported management interface, and verify that replacement nodes reproduce it. AKS documents selected configurable parameters; GKE separately exposes topology policy and scope. These interfaces require explicit configuration rather than assumptions about upstream defaults. (learn.microsoft.com) (docs.cloud.google.com)

The device path should remain part of change control. NVIDIA's pinned plugin code discovers GPU NUMA locality from sysfs; its DRA documentation publishes locality when the discovery result is usable. A driver or plugin upgrade should therefore trigger a comparison of device identity and topology metadata, not merely a check that a GPU resource remains visible. (raw.githubusercontent.com) (dra-driver-nvidia-gpu.sigs.k8s.io)

Future evaluation should prioritize repeatability over a universal performance percentage. Extend tests to helper bursts, workload restarts and mixed node occupancy. Inspect memory residency after warmup as well as allowed nodes: changing permitted memory nodes can trigger migration that is not necessarily complete. A steady-state NUMA mask alone is not a sufficient migration result. (cdn.kernel.org) (man7.org)

Use these decision criteria:

- **Adopt:** The tested stack is supported, admission is predictable, collectors reconcile allocations, and service objectives hold.
- **Restrict:** Placement is beneficial for selected workload shapes, but a broader pool lacks equivalent evidence.
- **Defer:** Gate configurability, vendor support, locality metadata or collector behavior remains unresolved.
- **Revalidate:** Repeat the contract after changes to node images, device drivers, runtime affinity or workload shape.

Broader deployment should wait when the documented support envelope and target version differ. The Network Operator support page inspected here is a concrete boundary to resolve, while EKS minor-version availability is evidence of a different capability. Treat each as its own dated fact rather than inferring an end-to-end support statement. (docs.nvidia.com) (docs.aws.amazon.com)

Conclusion

The defensible reason to enable Pod-level resource managers is a better allocation contract for a specific service: an exclusive main-container slice plus a deliberately sized helper pool, with CPU and memory placement that agrees with the devices actually assigned. The adoption decision should follow that contract through admission, execution, accounting and recovery.

Operators should begin with a small canary pool, a recorded GPU/NIC/CPU/memory mapping and a baseline workload that can be replayed. They should require evidence from effective CPU sets, thread affinity, permitted memory nodes and resident pages, rather than accepting any single topology display as proof. The feature changes how kubelet handles CPU and memory budgets; device selection and application behavior still need their own checks.

The business result remains conditional. A smaller reservation can be valuable even when peak throughput is unchanged, but it is useful only if the helper workload remains adequately provisioned and the service meets its objectives. A lower reservation is not itself a measured reduction in GPU cost.

Adopt where the complete, supported stack passes the acceptance procedure. Restrict deployment when evidence applies only to particular workloads or machines. Defer when vendor support, topology metadata, monitoring compatibility or rollback behavior is unresolved. A disciplined decision can therefore be positive, limited or negative without inventing a performance gain.

Source links

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

1. <https://kubernetes.io/blog/2026/09/15/kubernetes-v1-37-pod-level-resource-managers-beta>
2. <https://kubernetes.io/docs/tutorials/cluster-management/use-pod-level-resource-managers>
3. https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.17.1/internal/rm/nvml_devices.go
4. <https://dra-driver-nvidia-gpu.sigs.k8s.io/docs/reference/resourceslice-attributes>
5. <https://cdn.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html>
6. <https://man7.org/linux/man-pages/man8/numastat.8.html>

7. <https://docs.nvidia.com/networking/display/kubernetes2670/platform-support.html>
8. <https://learn.microsoft.com/en-us/azure/aks/custom-node-configuration>
9. <https://kubernetes.io/docs/reference/node/pod-level-resource-managers>
10. <https://docs.pytorch.org/docs/main/notes/cuda.html>
11. <https://docs.vllm.ai/en/latest/benchmarking/cli>
12. <https://www.open-mpi.org/projects/hwloc/doc/v2.14.0/faq.html>
13. <https://github.com/NVIDIA/k8s-device-plugin>
14. <https://raw.githubusercontent.com/k8snetworkplumbingwg/sriov-network-device-plugin/master/README.md>
15. <https://docs.pytorch.org/docs/main/utils/benchmark.html>
16. <https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates>
17. <https://www.glennklockwood.com/hpc-howtos/process-affinity.html>
18. <https://kubernetes.io/docs/concepts/resource-management/pod-level-resource-managers>
19. <https://kubernetes.io/docs/tasks/configure-pod-container/assign-pod-level-resources>
20. <https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.17.1/internal/rm/devices.go>
21. <https://learn.microsoft.com/en-us/azure/aks/custom-node-configuration-reference>
22. <https://docs.cloud.google.com/kubernetes-engine/docs/how-to/node-system-config>
23. <https://docs.aws.amazon.com/eks/latest/userguide/platform-versions.html>
24. <https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies>
25. <https://kubernetes.io/releases>
26. <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/dra-intro-install.html>
27. <https://docs.siderolabs.com/talos/v1.13/reference/configuration/v1alpha1/config>
28. <https://docs.cloud.google.com/kubernetes-engine/docs/reference/rest/v1beta1/NodeKubeletConfig>
29. <https://docs.nvidia.com/deploy/nvidia-smi>
30. https://cdrdv2-public.intel.com/637878/637878-E810%20DPDK%2021.02%20ConfigGuide_Rev1.0.pdf
31. <https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-bus-pci>
32. <https://openucx.readthedocs.io/en/master/faq.html>
33. <https://man7.org/linux/man-pages/man1/taskset.1.html>
34. <https://kernel.org/doc/html/v5.12/admin-guide/cputopology.html>
35. <https://man7.org/linux/man-pages/man7/numa.7.html>
36. https://rocm.docs.amd.com/projects/rocm_smi_lib/en/docs-7.2.1/doxygen/html/group__HWTopo.html
37. <https://docs.vllm.ai/en/v0.31.0/cli/bench/serve>
38. <https://man7.org/linux/man-pages/man8/numactl.8.html>
39. <https://kubernetes.io/docs/tasks/administer-cluster/topology-manager>
40. https://prometheus.io/docs/concepts/metric_types
41. <https://kubernetes.io/blog/2026/05/01/kubernetes-v1-36-feature-pod-level-resource-managers-alpha>
42. <https://prometheus.io/docs/practices/histograms>
43. <https://github.com/NVIDIA/nccl-tests>

- 44. <https://docs.mlcommons.org/inference/submission>
- 45. <https://endpoints.mlcommons.org>
- 46. <https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-versions.html>

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.