GPU SMITH
COMPUTE ENGINEERING

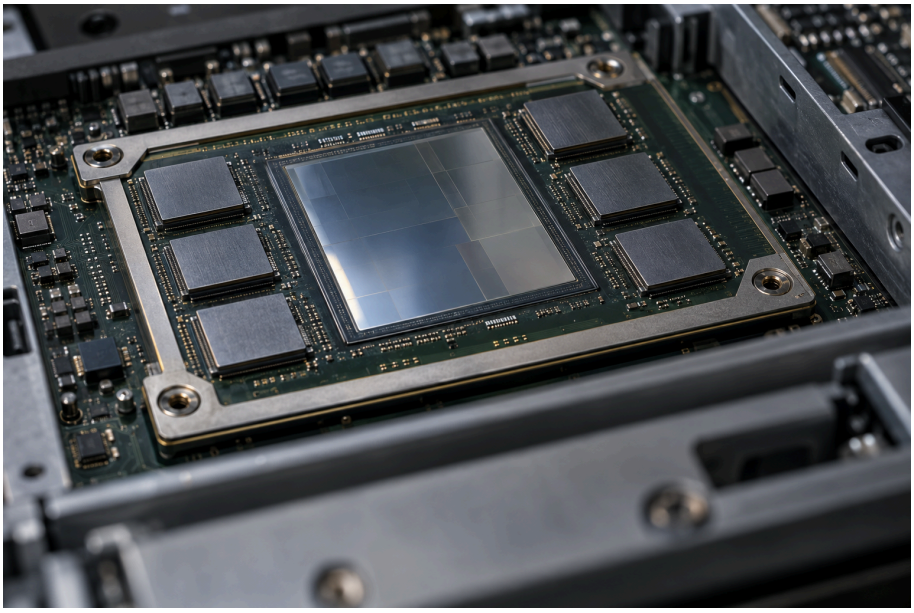
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

MIG vs MPS vs Time-Slicing for Kubernetes GPU Sharing

> 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 · 2026-10-01 · kubernetes gpu sharing · mig vs mps · gpu time-slicing · nvidia mig · nvidia mps · gpu operator · private ai infrastructure · inference · gpu scheduling

Original article: https://gpusmith.com/articles/en/kubernetes-gpu-sharing-mig-mps-time-slicing



In this article

1. Executive Summary
2. Introduction and Background
3. Whole GPU Allocation
4. Multi-Instance GPU
5. Multi-Process Service
6. Time-Slicing
7. Feature Comparison
8. Performance and Benchmarks
9. Data Analysis and Evidence
10. Implications and Future Directions
11. Frequently Asked Questions (FAQs)
12. Conclusion

Executive Summary

Private Kubernetes clusters should choose a graphics processing unit (GPU) allocation method from the workload contract: required tenant separation, maximum device memory, latency variation, acceptable fault blast radius, and the cost of changing a partition. **Whole GPU allocation** is the reference point. **Multi-Instance GPU (MIG)** creates hardware instances with dedicated compute and memory paths on supported products; **Multi-Process Service (MPS)** lets cooperative Compute Unified Device Architecture (CUDA) processes execute concurrently with configurable limits; **time-slicing** advertises multiple schedulable slots while processes take turns on the same device. The latter has no memory or fault isolation between replicas. (docs.aws.amazon.com) (docs.cloud.google.com) (docs.redhat.com) Kubernetes extended resources are integer quantities, so a sharing layer changes what the device plugin advertises; a fractional request alone does not divide a GPU. (kubernetes.io)

The practical gate is isolation. Use a whole GPU where a model needs the full memory footprint, a multi-GPU communication pattern needs an unpartitioned device, or the service cannot tolerate a neighbor. Prefer MIG when independently managed tenants need bounded memory and more predictable interference, after checking the exact GPU and profile: NVIDIA's current supported list includes **A100, A30, H100, H200, B200, and GB200**, but instance counts and memory sizes vary by product. (docs.nvidia.com) Consider MPS for cooperative jobs with moderate memory and concurrent small kernels, while treating the NVIDIA Kubernetes device plugin's MPS mode as **experimental** and accounting for its documented fault-sharing behavior. (github.com) (docs.nvidia.com) Time-slicing fits trusted, bursty notebooks or services when interference and shared memory are acceptable. (docs.cloud.google.com) (docs.aws.amazon.com)

There is no universal density multiplier. A 2026 co-execution study on an **A30 24 GiB** found favorable MPS combinations improved performance by **up to 30%**, while memory contention worsened performance by **around 30%**. These are observations under that study's workloads, not an infrastructure planning factor. (arxiv.org) (arxiv.org) (arxiv.org) Measure each candidate against the same input mix and offered load,

recording throughput, tail latency, queue depth, peak device memory, out-of-memory events, and the impact of a deliberately failing neighbor. vLLM exposes queue and latency metrics, while PyTorch can report peak tensor allocation; neither replaces whole-device memory telemetry. (docs.vllm.ai) (docs.pytorch.org)

Roll out on a labeled canary node, then compare against a whole-GPU baseline at the same service-level objective (SLO). Check the advertised resource name and count, test a second tenant's fault, and rehearse drain, reconfiguration, and uncordon before expanding the pool. Kubernetes drain prevents new pods from arriving and waits for graceful termination, subject to the application's disruption budget. (kubernetes.io) The decision record should contain the exact GPU SKU, driver and plugin versions, partition layout, observed metrics, and a rollback trigger. As of **October 1, 2026**, published mechanisms establish capabilities; only a local canary establishes the economics and latency for a private cluster.

Introduction and Background

A platform owner is often asked whether several inference servers, development notebooks, and batch jobs can use one expensive accelerator. That question hides distinct requirements. A replica count describes scheduling admission; a memory partition describes capacity ownership; concurrent kernel execution describes execution behavior. Treating those as the same promise creates a misleading capacity plan. The Kubernetes device-plugin API advertises discrete extended resources through kubelet, and those resources cannot be overcommitted directly. (kubernetes.io) The sharing strategy and its configuration determine how many resources a node appears to have.

This report addresses private clusters operated for one organization or for multiple internal tenants. The intake should start with trust boundaries and memory peaks, then evaluate supported hardware, framework behavior, service latency, observability, and change operations. Inference, notebooks, and batch work have different duty cycles; a GPU-utilization percentage alone is an unreliable autoscaling signal for inference, where queue depth and request latency can be more informative. (docs.aws.amazon.com) The choice is therefore a workload placement and operations decision, not a promise that sharing always saves money.

The comparison uses **NVIDIA's current MIG and MPS guides**, GPU Operator and device-plugin documentation, Kubernetes scheduling rules, and provider implementation notes, all accessed for an October 2026 decision. Features can vary across driver, CUDA, plugin, and GPU generations. This is especially relevant to MIG profile names and to MPS runtime features: an MPS capability in the standalone runtime does not imply the Kubernetes plugin exposes it. (github.com)

GPU Smith's first-party description places its work in workload modeling, cluster sizing, telemetry, and validation against written criteria. That is the relevant advisor perspective here: a mode should be accepted only against a workload-specific test record, not inserted as another technology option in the matrix. (gpusmith.com) (gpusmith.com)

Before opening a node pool to sharing, record five workload conditions:

- **Tenant boundary:** Identify who can submit code and who shares the device.
- **Memory envelope:** Capture a peak for each model and traffic shape.
- **Latency contract:** State the tail-latency objective and tested load.

- **Concurrency:** Record expected overlap among jobs and requests.
- **Recovery:** Define acceptable impact from a neighboring process failure.

Whole GPU Allocation

Capabilities

A pod requesting the ordinary `nvidia.com/gpu` resource can receive an entire device when the node's plugin is configured without sharing or MIG partitioning. Kubernetes permits a GPU limit without a separate request and defaults the request to the limit; if both are present, they must match. (kubernetes.io) A whole device leaves its memory and compute available to that allocation. It is the simplest baseline for identifying how much headroom the model, runtime, batch size, and context length actually need.

Adoption

Start the canary with one workload per GPU and record hardware identifiers, driver and CUDA versions, model version, traffic shape, and peak memory. For a model server, record offered requests per second, completed requests, queue time, time to first token, inter-token latency, and tail latency. vLLM exposes waiting requests and time-to-first-token histograms; client-side benchmarks also reveal network and queuing effects that a GPU counter misses. (docs.vllm.ai) (docs.vllm.ai) Keep the input set and cache state constant when testing the sharing alternatives. (docs.vllm.ai)

A reusable whole-GPU baseline should preserve:

- **Device identity:** Save SKU, memory capacity, and node identifier.
- **Software identity:** Pin driver, CUDA, plugin, and serving versions.
- **Traffic shape:** Save inputs, offered rate, and cache condition.
- **Service result:** Record completions, errors, queueing, and tail latency.
- **Memory result:** Record peak device usage and allocator observations.

Strengths and Limitations

Whole allocation is the conservative choice for large model weights or key-value cache peaks that nearly fill a device. It also avoids the allocation ambiguity of several processes competing for the same physical memory. It does not maximize admission count for small intermittent jobs, but a higher admitted-pod count is only valuable if completed work and the SLO improve. A full device is also the rollback destination when a sharing canary fails its memory, latency, or fault-isolation gate. The operational cost is capacity stranded while a low-duty-cycle pod holds the exclusive resource; measure that idle time before assigning an economic value to sharing.

The scheduler still sees a resource count, not the application-level useful work. A device plugin can mark a device unhealthy and reduce allocatable resources for new pods, but that does not prove an already running service met its SLO. (kubernetes.io) Whole allocation therefore remains a measured control group, not an exemption from telemetry.

Multi-Instance GPU

Capabilities

MIG subdivides a supported GPU into GPU instances with dedicated compute and memory resources. NVIDIA describes separate memory-system paths, including cache banks and memory controllers, and fault isolation between clients. (docs.aws.amazon.com) This is the strongest documented isolation among the three sharing choices, although application and host security still require their own controls. A GPU instance can also contain compute instances that share its parent memory and engines; operators should not describe such compute instances as separate memory partitions. (docs.nvidia.com)

The exact profile is a product decision. The current guide lists **H100 80 GB** profiles such as `1g.10gb`, `2g.20gb`, `3g.40gb`, and `7g.80gb`, while **B200 180 GB** has different profile sizes. (docs.nvidia.com) (docs.nvidia.com) The supported-products table gives a maximum of seven instances for several data-center products but fewer for A30 and selected RTX PRO Blackwell models. (docs.nvidia.com) Those maxima are geometry constraints, not promises that seven copies of a service will meet a latency target.

Adoption

Check the physical SKU against the supported-GPU table, choose a profile that exceeds measured peak device memory plus a locally chosen safety margin, and validate the application on that profile. Kubernetes device plugin `migStrategy=mixed` exposes profile-specific extended resources, allowing pods to request the named profile rather than an undifferentiated GPU. (github.com) AKS documentation likewise describes mixed MIG profiles as distinct scheduler resources, which is useful confirmation of the operational model, though a private cluster must use its own plugin configuration. (learn.microsoft.com) Label and taint a canary pool for the profile; use selectors or affinity so an ordinary `nvidia.com/gpu` request cannot accidentally land on the wrong class of node. Kubernetes node selectors require every specified label to match. (kubernetes.io)

A MIG acceptance check should confirm:

- **Product:** The exact SKU appears on the supported list.
- **Profile:** The chosen slice fits the measured memory envelope.
- **Resource:** Kubernetes advertises the expected profile name and count.
- **Placement:** Only approved workloads can select the MIG pool.
- **Recovery:** The intended layout returns after node maintenance.

Strengths and Limitations

Dedicated memory capacity and bandwidth reduce the contention uncertainty that matters in multi-tenant inference. MIG is a strong candidate when tenants require independent memory ceilings or when one neighbor's memory pressure should not change another instance's allocation. It is less flexible when a model needs more memory than one available profile or when workload sizes vary enough to fragment the available partition layout. NVIDIA explicitly notes that creating and destroying GPU instances can leave placement fragmentation. (docs.nvidia.com)

Reconfiguration needs a maintenance plan. On Ampere, enabling MIG mode can involve a GPU reset; on Hopper and later, the mode change need not reset the GPU, but MIG devices themselves do not persist across reboot. (docs.nvidia.com) (docs.nvidia.com) Configuration automation must recreate the intended instances, verify resource counts, and then admit workloads. NVIDIA's guide says MIG and MPS can coexist at the runtime level, but the current Kubernetes device plugin does **not** support MPS sharing on MIG-enabled devices. (docs.nvidia.com) (github.com) That distinction prevents a hardware possibility from being mistaken for a supported private-cluster configuration.

Multi-Process Service

Capabilities

MPS is a CUDA runtime for cooperative processes that can submit work to a shared GPU. It is an execution-concurrency mechanism, not a hardware partition. NVIDIA says MPS clients have isolated GPU address spaces and may be limited in device-memory allocation, but an active-thread percentage is a ceiling rather than a reservation of dedicated execution resources. (docs.nvidia.com) (docs.cloud.google.com) (docs.nvidia.com) The distinction matters for latency: a configured percentage does not establish a fixed latency slice when neighbors change their request mix.

The NVIDIA device plugin's MPS mode uses a control daemon and divides physical GPU memory equally among configured clients. Its documentation calls MPS support **experimental**, says MPS and time-slicing are mutually exclusive within that plugin, (github.com) and currently supports MPS sharing on full `nvidia.com/gpu` devices rather than MIG resources. (github.com) (github.com) (github.com) A separate MPS runtime guide describes newer MPS v3 memory partitioning, with Linux cgroup v2 and CUDA 13.4 or newer on a non-MIG device. That runtime feature should be evaluated separately from the plugin's documented integration. (docs.nvidia.com)

Adoption

Consider MPS after a whole-GPU baseline shows many small kernels or jobs that do not individually keep the GPU busy. NVIDIA identifies such processes as candidates for cooperative execution. (docs.cloud.google.com) Use one trusted workload family per pool until canaries show predictable coexistence. Record client count, peak memory for every client, combined memory, offered load, and per-client latency. If jobs use different CUDA frameworks or process lifecycles, test them together; successful startup alone does not establish fair scheduling or recovery behavior.

A node should use one MPS control daemon, and `nvidia-smi` may show the MPS server as the active CUDA process rather than individual clients. (docs.nvidia.com) (docs.nvidia.com) That affects incident triage and cost attribution. Pair device telemetry with application metrics and identify clients through the serving layer. Before enabling MPS in a private cluster, pin the device-plugin release and verify its stated experimental support for the particular GPU, driver, and node configuration. (github.com)

Strengths and Limitations

MPS can improve overlap among cooperative CUDA processes without imposing MIG's fixed profile geometry. That flexibility is attractive for small batch jobs and trusted inference services whose combined peaks fit memory. GKE's sharing guide presents MPS as a choice for small, cooperative batch jobs and distinguishes its software resource limits from MIG hardware isolation. (docs.cloud.google.com) The limit is fault containment: NVIDIA documents that a fatal fault in one MPS client can affect another sharing the GPU, and cautions about reduced isolation in multiuser mode. (docs.nvidia.com) MPS is therefore a poor substitute for a hardware boundary when independent tenants require a narrow blast radius. Use an explicit fault canary before accepting it for a shared service.

Time-Slicing

Capabilities

Time-slicing makes one physical GPU appear as several schedulable replicas through the plugin. The device still has one shared memory pool; NVIDIA's Operator guide says replicas do not gain memory or fault isolation. (docs.aws.amazon.com) (docs.redhat.com) AWS describes the same mechanism as advertising multiple slots per physical GPU, followed by GPU time multiplexing. (docs.aws.amazon.com) A replica is an admission token, not a guaranteed fraction of memory, bandwidth, compute, or latency. Requesting more than one shared resource does not promise proportionally more compute. (docs.nvidia.com)

Adoption

In the current GPU Operator pattern, a ConfigMap defines `sharing.timeSlicing.resources` with a resource name and `replicas`; a ClusterPolicy points the device plugin to that configuration. (docs.nvidia.com) Set `renameByDefault=true` when an explicit `.shared` resource name will help separate deployments. With the default false setting, the ordinary resource name can remain while the GPU product label gains a `-SHARED` suffix. (docs.nvidia.com) (docs.nvidia.com) `failRequestsGreaterThanOne=true` can reject requests that mistakenly treat several replicas as more guaranteed compute. (docs.nvidia.com) Use a node-specific configuration and a canary label to limit the initial blast radius. (github.com)

Always inspect the resulting node `capacity` and `allocatable` values and submit test pods requesting the intended resource. For four replicas on one physical GPU, the plugin can advertise four schedulable shared resources; it has not created four memory partitions. (docs.nvidia.com) Kubernetes still accounts for integer extended resources, so the plugin must be the source of that changed count. (kubernetes.io) Keep plain whole-GPU and shared-GPU node pools distinguishable by labels, selectors, and preferably dedicated workload classes.

Strengths and Limitations

Time-slicing has the lowest hardware barrier among these modes and suits trusted, intermittent notebooks, interactive development, and lightly loaded services whose peaks rarely coincide. GKE recommends it for bursty interactive workloads with idle periods. (docs.cloud.google.com) The tradeoff is unbounded contention

relative to a simple replica count. Red Hat warns that more replicas can raise out-of-memory risk if aggregate demand exceeds device memory. (docs.redhat.com) AWS similarly notes no memory or compute isolation between pods sharing a time-sliced device. (docs.aws.amazon.com)

Observability is also weaker: NVIDIA says DCGM Exporter cannot associate metrics with containers under device-plugin time-slicing. (docs.nvidia.com) Treat per-pod GPU utilization dashboards with caution, and retain service-level metrics for every replica. The Operator does not monitor edits to an existing time-slicing ConfigMap; applying a modified map requires a device-plugin pod restart. (docs.nvidia.com) A rollback must therefore restore the prior map or node label, restart the relevant plugin if needed, and verify advertised resources before traffic returns.

Feature Comparison

Table 1 maps mechanisms to the requirements that usually decide a private-cluster deployment. The entries describe the documented integration as of **October 1, 2026**, not a benchmark result. (docs.aws.amazon.com) (github.com)

Requirement	Whole GPU	MIG	MPS	Time-slicing
Hardware gate	Any GPU supported by the installed device plugin.	Only supported MIG products and profiles; check the exact SKU. (docs.aws.amazon.com)	CUDA MPS support and a compatible plugin; current plugin integration is experimental. (github.com)	Compatible NVIDIA GPU and sharing configuration; no MIG-capable card required. (docs.aws.amazon.com)
Memory boundary	Entire device assigned to one allocation.	Dedicated instance capacity and memory paths. (docs.aws.amazon.com)	Per-client memory limits in documented MPS modes; no separate physical memory path. (docs.cloud.google.com)	Shared pool with no per-replica isolation. (docs.redhat.com)
Fault boundary	No colocated GPU tenant in this allocation.	Hardware instance fault isolation. (docs.redhat.com)	A fatal client fault can affect another client. (docs.nvidia.com)	No fault isolation between replicas. (docs.redhat.com)
Execution	One assigned workload controls the device, subject to its own processes.	Concurrent isolated hardware instances.	Cooperative CUDA execution; active-thread limit is not a reservation. (docs.cloud.google.com)	CUDA time multiplexing; replica count does not reserve compute. (docs.aws.amazon.com)
Scheduler resource	Usually `nvidia.com/gpu`. (kubernetes.io)	Mixed strategy exposes `nvidia.com/mig-*` profile resources. (github.com)	Plugin currently exposes shared full-GPU resources. (github.com)	Configurable shared name, often `.shared`, or ordinary name plus shared label.
Change and telemetry	Drain and replace pods; whole-device DCGM baseline.	Recreate layout after reset; MIG instance labels require telemetry checks.	Control daemon and client attribution require validation.	ConfigMap changes can require plugin restart; container GPU attribution is limited. (docs.cloud.google.com)

The table should be applied as a series of gates. A hard memory or fault boundary eliminates time-slicing; it usually eliminates MPS as a cross-tenant solution. An unsupported MIG product eliminates MIG. The remaining candidates must then pass a workload SLO and an operational rollback test. Cloud implementations can differ: GKE documents sharing within individual MIG partitions, whereas the current NVIDIA Kubernetes device plugin does not support its MPS mode on MIG devices. (docs.cloud.google.com) (github.com) Confirm the integration actually being deployed rather than combining capabilities from separate products.

A workload-to-mode decision can be stated compactly:

- **Independent tenant with memory and fault boundary:** test MIG profiles, or allocate a whole GPU if no profile fits.
- **Large model with peak memory near device capacity:** start with a whole GPU; consider model-level optimization before adding sharing.
- **Trusted, small cooperative CUDA jobs:** test MPS against the whole-GPU baseline, including a fault canary.
- **Trusted bursty notebooks or low-duty-cycle inference:** test time-slicing with a small replica count and explicit admission control.
- **Multi-GPU training or tightly coupled communication:** validate whole GPUs first, since partitioned-device communication support and topology can constrain the job.

These are placement hypotheses. The following measurements determine whether any candidate should move from canary to production.

Performance and Benchmarks

A benchmark of GPU-sharing mechanisms is only comparable when the hardware, model, precision, batch shape, prompt or input distribution, runtime, and offered load are held constant. Changing several at once confuses the effect of sharing with the effect of batching or cache reuse. A client-side load tool should report completed throughput and latency, while server metrics expose queueing and execution. vLLM's serving benchmark measures latency at the client and supports a controlled target request rate. (docs.vllm.ai) (docs.vllm.ai) Triton's Perf Analyzer offers concurrency and request-rate load modes and reports latency and throughput from the client perspective.

Hold these controls constant between trials:

- **Hardware:** Compare on the same physical GPU product.
- **Software:** Pin the framework, driver, model, and precision.
- **Inputs:** Reuse the same request distribution and sequence lengths.
- **Load:** Match offered demand and the test duration.
- **Cache:** Reset or document reusable state between runs.

Published studies show why a universal ranking is unsafe. In a **2026 A30 24 GiB** co-execution study, MPS improved performance by **up to 30%** in favorable workload combinations, but memory contention worsened performance by **around 30%**. The study also reported energy reduction of about **20%** in favorable cases. (arxiv.org) Those figures are from specific applications and comparison baselines, not estimates for

Kubernetes inference. (arxiv.org) (arxiv.org) (arxiv.org) Another study used an **A100 40 GB** and image-inference batches of **one** to examine isolation, a different setting from token-generating inference with long contexts. (arxiv.org) (arxiv.org) Neither paper supplies a portable replica count for a private production pool.

The benchmark plan should include three load shapes. First, a steady state at the current production mix establishes baseline SLO compliance. Second, a coincident peak makes memory, queue, and execution interference visible. Third, a neighbor-fault case tests whether the surviving service stays available and within its latency objective. Use the same device SKU and software stack for all modes, then vary only the sharing configuration and number of colocated workloads. Flush or account for caches between trials; vLLM warns repeated runs can reuse prefix cache and inflate measured throughput. (docs.vllm.ai)

Report p50, p95, and p99 latency as measured results rather than a single average, plus completed work per GPU-hour. Prometheus histogram buckets support calculating quantiles across workers, whereas precomputed summary quantiles cannot be aggregated meaningfully. (prometheus.io) Trace the request path as well as the GPU: OpenTelemetry specifies an HTTP server request-duration histogram, which can help separate application-side delay from accelerator-side delay. (opentelemetry.io) The inference server's own queue and time-to-first-token histograms should be retained during each trial. (docs.vllm.ai)

A candidate wins only if it meets every non-negotiable isolation and support requirement, meets the existing SLO at the tested load, and improves the locally defined objective, such as completed requests per installed GPU or cost per successful request. A lower utilization percentage can coexist with better latency; a higher advertised replica count can coexist with worse completions. For autoscaling, AWS recommends queue depth and request latency rather than GPU utilization alone. (docs.aws.amazon.com)

Data Analysis and Evidence

Do not multiply a physical GPU's memory by the number of time-slicing replicas. The replicas represent admission slots; GKE says time-sharing does not enforce memory limits between shared jobs. (docs.cloud.google.com) For MIG, compare peak memory to the exact profile capacity; for MPS, distinguish a memory limit from a bandwidth guarantee. PyTorch's peak tensor-allocation API can measure one component of demand, while its memory documentation notes allocations outside PyTorch may require separate inspection. (docs.pytorch.org) (docs.pytorch.org)

Table 2 is a workload intake worksheet. Each row should be filled for the existing whole-GPU baseline before a mode is chosen. A blank cell is a measurement task, not permission to assume headroom.

Input	Record locally	Decision use
Hardware and stack	GPU SKU and memory, MIG support/profile, driver, CUDA, Operator, plugin, serving runtime versions. (docs.aws.amazon.com)	Eliminate unsupported combinations before benchmarking.
Tenant contract	Trust boundary, data classification, required memory and fault separation, acceptable neighboring workloads.	Select the minimum isolation mechanism before considering density.

Input	Record locally	Decision use
Memory	Model weights, maximum observed device use, framework peak, cache growth, OOM count, and a documented safety margin. (docs.pytorch.org) (docs.pytorch.org)	Reject a profile or client limit that cannot contain peaks.
Traffic and service	Request mix, offered rate, queue depth, completed rate, p50/p95/p99 latency, time to first token, error rate. (docs.vllm.ai) (docs.vllm.ai)	Compare modes at equal demand and the same SLO.
Operations	Drain time, restart time, label/resource verification, reconfiguration time, recovery from neighbor fault. (kubernetes.io)	Require a rollback window the service can tolerate.
Economics	User-supplied GPU acquisition or lease cost, amortization period, power, support labor, and measured successful work.	Calculate cost per completed request or job only after SLO gates pass.

Exclude a mode that fails isolation or p99 latency at peak demand. Then divide the owner's period cost by SLO-compliant completed requests or jobs. Advertised slots are not completed work.

Table 3 is a canary scorecard for a labeled node pool. Thresholds are deliberately blank until the owner inserts the existing SLO and locally approved recovery objective. The observed and pass/fail fields should be retained with the configuration snapshot.

Canary test	Observe	Pass rule to set before launch
Scheduling	Node labels, `capacity`, `allocatable`, requested resource name, placement, and pod admission. (kubernetes.io)	Only intended workloads reach the pool; count matches configured devices or replicas.
Steady and peak load	Completed throughput, queue depth, p95/p99 latency, GPU memory, per-service errors. (docs.vllm.ai) (prometheus.io)	Existing SLO holds at both local traffic shapes.
Memory pressure	Peak memory, OOM events, survivor memory and latency when a neighbor grows. (docs.pytorch.org)	A required tenant boundary remains effective; shared modes stay within approved risk.
Fault blast radius	One intentionally failing test client, surviving client status, restart and recovery time.	Surviving service meets the owner-defined availability objective.
Telemetry	Device/instance identity, service latency, label mapping, and gaps in per-container GPU attribution. (docs.cloud.google.com)	Operators can explain each measured result and detect a failed canary.
Rollback	Drain, restore prior config or node label, plugin restart if required, resource recount, uncordon. (kubernetes.io)	Previous whole-GPU or partition layout returns inside the approved window.

DCGM Exporter can label MIG samples by profile and instance, but verify metric support and mapping on the host. (docs.nvidia.com) Kubernetes pod-resource mapping is conditional, and adding labels cannot create unsupported DCGM fields. For time-slicing, application metrics are essential because shared-GPU monitoring can be node-level. (docs.cloud.google.com)

Implications and Future Directions

For platform engineering, the first implementation artifact should be a **mode policy**: which namespaces or workloads may use whole GPUs, MIG profiles, MPS, or shared replicas, and why. Labels should reflect the approved policy, not only the installed hardware. Kubernetes node selectors can enforce the selected pool, while the device plugin's advertised resources remain the scheduler's source of GPU capacity. (kubernetes.io) (kubernetes.io) A future move to another device-allocation API should preserve the same isolation and SLO tests; changing the API does not change the physical sharing mechanism.

A staged rollout is safer than a cluster-wide switch:

- **Baseline:** save the existing deployment manifests, device-plugin configuration, node labels, and whole-GPU scorecard.
- **Canary:** reserve a small node pool, configure one mode and one profile or replica count, and verify the node's reported resources before scheduling production-like traffic.
- **Load and fault tests:** replay normal and coincident peaks, then inject a test-client OOM or fatal workload fault in an isolated test environment.
- **Decision:** record SLO compliance, successful work per GPU, memory peaks, fault impact, and the cost input used. Keep a rejected candidate's evidence rather than silently increasing replica count.
- **Rollback:** stop new placement, drain the canary, restore prior configuration and labels, restart the device plugin when required, verify the original resource count, then uncordon. (kubernetes.io)

Plan for voluntary disruption limits. A Kubernetes PodDisruptionBudget can restrict how many replicated pods are voluntarily disrupted during drain, though it cannot prevent involuntary disruption. (kubernetes.io) MIG reconfiguration may also alter the set of resource names and their availability; verify application manifests before returning the node to service. In managed environments, operational constraints can be stricter: AKS documents static node-pool MIG configurations that require node reprovisioning. (learn.microsoft.com) That cloud-specific behavior should not be generalized to every private MIG host, but it illustrates why the rollback record must identify the deployment path.

GPU Smith describes its own method as deriving topology and serving stack from workload models and throughput targets, with validation against written criteria. For an independent engineering advisor, the useful outcome is a reproducible acceptance record containing the specific device and software version, not a blanket recommendation for one sharing mode. (gpusmith.com)

Frequently Asked Questions (FAQs)

Is MIG always better than MPS for AI inference?

No universal performance order follows from the mechanism. MIG offers dedicated memory and fault isolation, which can be decisive for independent tenants. MPS can overlap cooperative processes but its active-thread cap is not a dedicated reservation, and shared-client faults require testing. (docs.aws.amazon.com) (docs.cloud.google.com) Compare both on the same model, load shape, and latency objective; the A30 study's favorable and unfavorable MPS results show how workload interaction changes the outcome. (arxiv.org)

Can time-slicing be combined with MIG?

The NVIDIA GPU Operator documents time-slicing of resource types exposed by mixed MIG configuration. That shares an individual MIG resource among multiple pods and gives up isolation **between those replicas**, even though separate MIG instances retain their hardware boundaries. (docs.cloud.google.com) (docs.redhat.com) Test the exact placement and telemetry rather than assuming each replica is another independent partition.

Can the NVIDIA device plugin combine MPS and MIG?

Its current documentation says MPS sharing is not supported on MIG-enabled devices, although NVIDIA's standalone MIG guide says MPS can run within MIG instances. (github.com) A separate platform or custom integration may support a different combination, but it must be evaluated on its own support and isolation terms.

Which mode is best for private-cluster notebooks?

Trusted, intermittent notebooks often justify a time-slicing canary because they may idle between bursts. If one user can consume another's memory or a shared fault is unacceptable, use MIG or a whole GPU. GKE identifies bursty interactive jobs as a time-sharing candidate; the Operator documents its lack of memory and fault isolation. (docs.cloud.google.com) (docs.redhat.com)

Conclusion

Choose Kubernetes GPU sharing by eliminating modes that fail the workload's hard requirements, then measure the survivors. Whole GPUs supply the baseline and remain appropriate when a model needs the device's full memory or when a neighbor cannot be tolerated. MIG is the hardware-partition choice on supported products when memory and fault isolation matter. MPS is an execution-concurrency choice for cooperative, trusted CUDA workloads, subject to the installed integration's support limits. Time-slicing is a scheduling-admission choice for trusted bursty work and keeps memory and faults shared among replicas.

No mechanism supplies a portable density ratio. The accepted configuration must state the exact GPU product, profile or replica count, driver and plugin versions, and the service's own latency and availability objectives. It should show peak memory under coincident demand, completed work rather than merely scheduled pods, and the result of a controlled neighboring failure. A rollback rehearsal should restore the previous resource count and placement labels before the canary returns to service.

That record makes the decision reversible. If a candidate violates an isolation requirement, exceeds its memory envelope, misses the SLO, or cannot be observed well enough to diagnose contention, return the workload to the whole-GPU baseline or a different MIG profile. If several candidates pass, compare their measured successful work against the owner's actual period cost. The result is a capacity policy tied to evidence from this cluster, not a universal ranking of MIG, MPS, and time-slicing.

Source links

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

1. https://docs.aws.amazon.com/eks/latest/userguide/device-management-nvidia.html
2. https://docs.cloud.google.com/kubernetes-engine/docs/concepts/timesharing-gpus
3. https://docs.redhat.com/en/documentation/red_hat_openshift_ai_self-managed/3.4/html/working_with_accelerators/about-gpu-time-slicing_accelerators
4. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
5. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-gpus.html
6. https://github.com/NVIDIA/k8s-device-plugin
7. https://docs.nvidia.com/deploy/mps/architecture.html
8. https://arxiv.org/html/2604.22430
9. https://docs.vllm.ai/en/latest/usage/metrics/
10. https://docs.pytorch.org/docs/2.14/generated/torch.cuda.memory.max_memory_allocated.html
11. https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/
12. https://gpusmith.com/articles/en/good-gpu-utilization-percentage
13. https://docs.aws.amazon.com/eks/latest/userguide/ml-inference-autoscaling.html
14. https://gpusmith.com/
15. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
16. https://docs.vllm.ai/en/stable/benchmarking/cli/
17. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/latest/concepts.html
18. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-mig-profiles.html
19. https://learn.microsoft.com/en-us/azure/aks/gpu-multi-instance
20. https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/
21. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/getting-started-with-mig.html
22. https://docs.nvidia.com/deploy/mps/latest/index.html
23. https://docs.nvidia.com/deploy/mps/when-to-use-mps.html
24. https://docs.nvidia.com/deploy/mps/mpsv3-memory-partitioning.html
25. https://docs.nvidia.com/deploy/mps/common-tasks.html
26. https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html
27. https://gpusmith.com/articles/en/dcgm-exporter-gpu-monitoring-metrics-alerts-triage
28. https://arxiv.org/html/2601.07600
29. https://prometheus.io/docs/practices/histograms/
30. https://opentelemetry.io/docs/specs/semconv/http/http-metrics/
31. https://docs.pytorch.org/docs/2.14/torch_cuda_memory.html
32. https://docs.nvidia.com/datacenter/dcgm/latest/reference/dcgm-exporter-metrics.html
33. https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
34. https://learn.microsoft.com/en-us/azure/aks/concepts-gpu-partitioning
35. https://gpusmith.com/articles/en/mig-vs-mps-vs-time-slicing

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.