



Kubernetes GPU Scale to Zero: Cold Start vs Idle Cost 2026

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 · 2026-10-20 · kubernetes gpu scale to zero · kubernetes hpa scale to zero · kubernetes 1.37 · gpu cold start · gpu idle cost · keda scale to zero · gpu autoscaling · scale to zero break-even

Original article: <https://gpusmith.com/articles/en/kubernetes-gpu-scale-to-zero-cold-start>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Key Changes in Kubernetes 1.37 HPA Scale-to-Zero
 4. Eligibility: Which GPU Workloads Can Use Scale-to-Zero
 5. Cold-Start Decomposition and the Signal Path
 6. Pod Zero Versus Node Zero
 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

Kubernetes **v1.37** (announced 26 August 2026) graduated HorizontalPodAutoscaler (HPA) scale-to-zero to **Beta** and enabled it by default (kubernetes.io). An HPA that uses an object or external metric may now hold a workload at zero replicas and bring it back when the metric changes (kubernetes.io). The release notes name queue consumers, batch jobs and GPU workloads as the intended beneficiaries (kubernetes.io). Three constraints define the decision: CPU and memory metrics cannot drive scale from zero (kubernetes.io), Kubernetes Services do not buffer requests while no Pods are ready (kubernetes.io), and a workload sitting at zero without the controller's own ScaledToZero condition is treated as manually paused (kubernetes.io).

The central finding of this report is that **pod-zero is not node-zero**. Removing the last Pod does not power down a GPU server; on an owned node the saving is the difference between a GPU holding a CUDA context and a bare-idle GPU, which one preprint measured at **26 to 66 W** per GPU (arxiv.org). For an eight-GPU node, a hypothetical calculation in this report puts that at roughly **\$0.05 to \$0.12 per hour** at US commercial electricity prices (www.eia.gov). Powering the whole node off is worth more, since one measured eight-GPU H100 node idled at a median **1.86 kW** (arxiv.org), but that requires a hardware power path that Kubernetes does not provide. In a cloud, deleting a node can avoid an instance rate that is two orders of magnitude higher, but only when the instance is terminable. The **\$47.757 per hour** figure used here is the AWS Capacity Block effective rate for an eight-H100 instance (aws.amazon.com). That is prepaid reserved capacity, and AWS charges the price of a Capacity Block up front at the time the reservation is made (docs.aws.amazon.com), so deleting the node saves nothing once the block is bought and the cloud rows are illustrative only. The same feature has a completely different payoff when the node is billed by the hour and can actually be released.

Cold start is the cost side. Published component measurements range from a **20.32 s** total start for a 3B-parameter model with cached weights (arxiv.org) to **more than 40 s** to first token for a cold-start instance in the authors' production serverless environment (arxiv.org), with a 10 GB GPU container image alone taking **1m52.876s** to pull under default settings (aws.amazon.com). One vendor puts a full cold start from zero at **8 to 15 minutes** for large models (cast.ai).

The proposed break-even rule is $T^* = (\text{wake cost} + \text{SLO cost}) / \text{idle cost rate}$. When wake power roughly equals the power avoided, the energy-only T^* is about the wake duration; more precisely, energy-only T^* equals the wake duration times the ratio of wake power to the power saved, which is why the pod-zero rows (wake draw of about 1.6 kW against 0.4 kW saved) sit at 4.0 minutes for a 60 s wake while the node-off and cloud rows sit at 10.0 minutes. Either way the service-level objective (SLO) term dominates. In the worked example, a \$0.50 penalty per late job moves the owned-hardware pod-zero threshold from about 4 minutes to about 5.6 hours, while a cloud node stays near 10 minutes. The practical conclusion: use native HPA scale-to-zero for durable-queue workers whose deadlines exceed the measured cold start, keep a warm replica for interactive endpoints, and measure power, load time and arrival gaps on the actual stack before claiming any saving.

Introduction and Background

Operators of private Kubernetes inference and queue-worker fleets have long faced a blunt choice. Either a GPU Pod stays resident and burns idle capacity, or an add-on such as KEDA (Kubernetes Event-driven Autoscaling) removes it and the next request pays the start-up penalty. One vendor guide published in July 2026 summarized the native position as "HPA alone cannot scale GPU pods to zero" ([cast.ai](#)). That description was accurate for default clusters at the time. Kubernetes 1.37 changed it.

The change is smaller than the headline suggests. Alpha support dates to Kubernetes v1.16, and the work needed to distinguish an autoscaler-owned zero from a manual pause arrived later. The Kubernetes Enhancement Proposal (KEP) 2021 describes the motivation as workloads with high resource demands, such as those requiring GPUs ([github.com](#)). Yet the same KEP lists request buffering at the Service level as an explicit non-goal ([github.com](#)). In other words, the platform now decides when replicas exist, and it leaves the question of what happens to work that arrives in the meantime to the operator.

This report treats the feature as a decision problem for owned clusters. It separates pod scale-to-zero from GPU-node power and cost savings, decomposes cold start into measurable stages, defines a break-even model with explicit assumptions, and proposes a reproducible test plan. It is published by GPU Smith, an independent AI infrastructure consultancy. GPU Smith is an adjacent advisor here: it does not sell autoscalers, cloud capacity or hardware, and it has no benchmark results of its own in this report. All measured figures come from the cited third-party sources, dated where the source dates them, and all worked calculations are labeled as hypothetical examples.

Evidence was retrieved on 20 October 2026. The Kubernetes feature itself is weeks old, so no public body of production experience exists yet. The project states that its next step is to gather operational feedback before considering graduation to GA ([kubernetes.io](#)). Readers should treat the thresholds below as hypotheses to be tested, not as findings.

Key Changes in Kubernetes 1.37 HPA Scale-to-Zero

Beta Status and Default-Enabled Gates

The `HPAScaleToZero` feature gate is enabled by default on both `kube-apiserver` and `kube-controller-manager` (kubernetes.io). The API server accepts `minReplicas: 0`, and the controller manager performs the condition-based scaling. A KEP graduation criterion for GA is that the feature has been enabled by default in beta for at least one release without blocking bugs (github.com), so the earliest GA signal is at least one release cycle away.

Control-plane version skew deserves attention in any rolling upgrade. The release post advises waiting until both components support the feature and have it enabled before creating HPAs with `minReplicas: 0` (kubernetes.io). A controller manager with the feature disabled treats zero replicas as a manual pause and may leave a workload at zero (kubernetes.io). The KEP gives the underlying reason: the old controller cannot distinguish an HPA-driven zero from a manual one (github.com).

Object and External Metrics Only

Scale from zero requires a signal that exists when no Pods do. Once the replica count reaches zero there are no Pods left to measure, so documentation states that scaling to zero is supported only for object and external metrics (kubernetes.io), and the API server rejects an HPA that contains only resource metrics such as CPU or memory (kubernetes.io).

For private clusters, this means a metrics adapter is mandatory. The Prometheus Adapter, used in the official example, needs special `externalRules` entries to publish external metrics (raw.githubusercontent.com). A queue depth, consumer lag or pending-job count is the natural signal, because it exists independently of the workers that consume it. GPU utilization is not a candidate: it disappears with the Pod, and one inference-routing project notes that GPU utilization is often pegged near 100% during active batching regardless of actual load (llm-d.ai).

The ScaledToZero Condition and Manual Pause

Scaling from zero creates an ambiguity between "the autoscaler scaled this down" and "an operator paused it". The controller resolves it with the `ScaledToZero` condition, which it uses to distinguish a workload that it scaled to zero from one deactivated by hand (kubernetes.io). The practical consequence is a deployment trap: manually setting a Deployment to zero has always paused autoscaling, and the HPA preserves that behavior (kubernetes.io). A new Deployment should therefore start with at least one replica, so that the controller itself performs the first scale-down.

Normal HPA timing still applies. The default downscale stabilization window is five minutes (kubernetes.io). The control loop runs every 15 seconds by default (kubernetes.io).

What `minReplicas 0` Does Not Do

Four things are not provided by the feature, and each maps to a cost in the break-even model:

- **No request buffering.** Kubernetes Services do not buffer requests while no Pods are ready, so HTTP and other request-driven workloads need a separate buffering layer (kubernetes.io).
- **No node action.** The HPA changes a replica count. Whether a GPU node is drained, powered off or deleted is a separate system (see the section on pod zero versus node zero).
- **No warm path.** The post names the trade-off directly: the HPA must observe the metric, schedule a Pod, and start the application (kubernetes.io).
- **No metric-outage tolerance.** An HPA cannot scale from zero when its metric is unavailable (kubernetes.io), and recovery may require restoring the metric or manually scaling the workload (kubernetes.io).

Eligibility: Which GPU Workloads Can Use Scale-to-Zero

The release post says the pattern works well when work can wait in a durable queue (kubernetes.io). That sentence is the eligibility test. *Table 1* below applies it to five workload classes common in private GPU fleets. The verdicts are this report's analytical judgments from the cited documented behavior, not vendor certifications.

Workload class	Where does waiting work live?	Cold-start tolerance	Verdict for native HPA minReplicas: 0	Documented considerations
Durable queue or batch worker	In a broker or job store that outlives the workers	Minutes, bounded by the job deadline	Eligible	Queue depth is a valid external metric; Kafka keeps log data for a configurable period, 168 hours by default in its broker docs (kafka.apache.org); Amazon SQS retains messages 4 days by default (docs.aws.amazon.com)
Buffered asynchronous inference	In a queue in front of the model server, with a callback or polling result	Tens of seconds to minutes, bounded by the client timeout	Eligible with a measured ceiling	Retry and visibility timeouts must exceed cold start; the SQS default visibility timeout is 30 seconds (docs.aws.amazon.com), shorter than many measured GPU cold starts
Interactive HTTP endpoint	Nowhere: the client connection is the buffer	Seconds	Not eligible without a buffer or warm replica	Services do not buffer; KEDA's HTTP add-on holds requests until the backend is ready (keda.sh); Knative's activator forwards queued requests after scale-up (knative.dev)
Stateful or cache-heavy endpoint	In the Pod (KV cache, compiled graphs, loaded adapters)	Depends on how costly the state is to rebuild	Usually keep one warm replica	vLLM documents that compile artifacts can be cached and shared among instances (blog.vllm.ai); rebuilding runtime state is part of cold start
Air-gapped model store	In a local registry and storage the cluster owns	Governed by local pull and read speed	Eligible if images and weights are pre-staged	The GPU Operator needs all images in a local registry when offline (docs.nvidia.com); pre-pulling and a local mirror such as Harbor's proxy cache (goharbor.io) keep pulls off the critical path

The matrix shows why the feature is narrower than a generic "cost saver". Only the first two rows have a place for waiting work to sit. Queue durability must be checked against the broker's own documentation: RabbitMQ describes quorum queues as durable, replicated queues (www.rabbitmq.com), and the retention defaults in Kafka and SQS cited above set an upper bound on how long a fleet can stay at zero before messages expire.

Alternatives That Already Buffer or Gate

Native scale-to-zero is not the only route, and the alternatives differ in who owns the 0-to-1 step. KEDA's ScaledObject checks each trigger source every 30 seconds by default (keda.sh) and waits a cooldown period of 300 seconds by default after the last active trigger before scaling to zero (keda.sh). Once at one replica, the HPA controller makes the 1-to-N decisions (keda.sh). A CNCF-hosted post by an individual author shows a KEDA external scaler that reads GPU memory metrics and scales a vLLM deployment, including to zero when idle (www.cncf.io).

For request-driven services, Knative keeps the last Pod active for a configurable retention period before removal (knative.dev), and KServe documents setting minimum replicas to zero for use cases like serving on GPUs (kserve.github.io). Ray Serve notes that the first requests after a scale to zero see a cold start while replicas start (docs.ray.io). The native HPA path adds a core option; it does not remove the need for a buffer on synchronous traffic.

Cold-Start Decomposition and the Signal Path

Cold start is a chain of stages, and they add. *Table 2* below assembles published measurements for each stage. These are not one benchmark. They come from different models, hardware and storage, so the table is a map of where time goes and which levers exist, not a prediction for any particular cluster.

Stage	What happens	Published measurement (source stack)	Lever
1. Signal and detection	Metric scraped, adapter serves it, HPA reads it	Prometheus default scrape interval is 1 minute (prometheus.io); HPA loop defaults to 15 s; KEDA polls every 30 s (keda.sh)	Shorten scrape interval; use a metric computed at query time
2. Scheduling and node readiness	Pod placed; if no GPU node is free, a node is added	Cluster Autoscaler waits up to 15 minutes for requested nodes (github.com)	Keep the node; pre-bake images; owned fleets skip this stage
3. Image pull	Runtime and server image fetched	A roughly 10 GB vLLM image took 1m52.876s with default containerd and 45.121s with parallel pull on EKS (aws.amazon.com); image streaming cut a 5.4 GB Triton start from 191 s to 30 s on GKE (cloud.google.com)	Pre-pull; local registry mirror; lazy or parallel pull; kubelet pulls serially by default
4. GPU runtime initialization	Driver and CUDA context created	NVIDIA states that without persistence the GPU is initialized per CUDA job, adding load times on the order of seconds (docs.nvidia.com)	Enable persistence mode or daemon

Stage	What happens	Published measurement (source stack)	Lever
5. Weight read	Model parameters moved from storage to GPU memory	OPT-30B into 4 GPUs from local NVMe took 34 s with a PyTorch loader (www.usenix.org); fetching a 130 GB checkpoint from S3 or blob storage takes at least 26 s at 5 GB/s (www.usenix.org); a 7B load measured 29.7 s on an H100 (arxiv.org)	Concurrent streaming loaders (developer.nvidia.com); local NVMe; faster safetensors loading, reported at 4.8x to 7.5x (arxiv.org)
6. Runtime warmup	Graph compilation, CUDA graph capture, KV-cache profiling	Loading cached compiled graphs took up to 5.66 s for one 9B model (arxiv.org); a 3B model's total start was 20.32 s with weights in page cache (arxiv.org)	Share compile cache (blog.vllm.ai); pre-built engines (docs.nvidia.com)
7. Readiness	Probes pass; endpoint becomes routable	Liveness and readiness probes do not run until a configured startup probe succeeds (kubernetes.io)	Size the startup probe to worst-case, not typical, start

Table 2 shows two things. First, no single stage dominates across stacks: on fast local storage, overall start improved by only 1.04x when weights were read from SSD rather than page cache (arxiv.org), which means CPU-bound initialization can matter as much as bytes moved. Second, the total is workload-specific. NVIDIA describes cold-starting inference workloads on Kubernetes as something that can take several minutes (developer.nvidia.com), while the same company's snapshot approach reports a 21x start-time reduction for a 120B-class model in a proof of concept (developer.nvidia.com). A published 130 GB-class read time of at least 26 s (www.usenix.org) is already a large share of a minute, and the vendor claim in the summary (8 to 15 minutes including node provisioning) comes without a stated measurement method.

Failure Modes on the Signal Path

The signal path has more links than a CPU-based HPA. The failure modes worth testing are these:

- **Adapter staleness.** The Prometheus Adapter refreshes its cache of available metrics every 10 minutes by default (raw.githubusercontent.com). The README ties that interval to metric discovery, so whether a newly appearing series is served promptly must be tested, not assumed.
- **Slow detection.** If the queue gauge is scraped at the 1-minute default (prometheus.io) and the HPA loop adds up to 15 seconds, a Pod may not be requested until roughly 75 seconds after work arrives, assuming the adapter queries Prometheus at request time (an assumption to verify).
- **Metric outage.** An HPA cannot scale from zero when its metric is unavailable. A workload stuck at zero with queued work is the highest-severity case, and the documented recovery is restoring the metric or scaling manually.
- **Version skew and downgrade.** Mixed control-plane versions can leave a workload paused at zero, and a downgrade requires raising minReplicas and scaling up first (kubernetes.io).

Pod Zero Versus Node Zero

Scaling a Pod to zero frees a GPU resource request. It does not, by itself, change what the server draws from the wall. What happens next depends on three separate cases.

Case 1: The Node Stays, the GPU Idles

The only guaranteed energy saving is the difference between a GPU holding a CUDA context and a bare-idle GPU. A preprint measured 14 production H100s and found a bimodal distribution, with bare-idle and CUDA-active states differing by about 71 W ([arxiv.org](#)). Across H100, A100 and L40S in its lab measurements, a CUDA context added 26 to 66 W over bare idle, with 26 to 50 W on the HBM parts (H100, A100) and 66 W on the GDDR6 L40S ([arxiv.org](#)). Nameplate power is not the right basis: NVIDIA lists the H100 SXM at up to 700 W ([www.nvidia.com](#)), but that is a maximum design power, and the same preprint recorded a 1 Hz H100 model-load trace peaking at 124.1 W, far below TDP ([arxiv.org](#)).

Case 2: The Node Is Powered Off

A server draws far more than its GPUs at idle. One eight-GPU H100 HGX node measured a median 1.86 kW with the node idle, against 7.92 kW for a Llama2-13b training run ([arxiv.org](#)), and a maximum observed draw of 8.4 kW against a manufacturer rating of 10.2 kW ([arxiv.org](#)). A classic result for servers generally is that an energy-efficient machine still consumes about half its full power when doing almost no work ([www.cs.princeton.edu](#)). Turning a node off saves the whole baseline, but Kubernetes only drains and cordons; powering hardware requires another layer. Bare-metal tooling provides it: Metal3 keeps a host powered on only while its online flag is set ([github.com](#)), Tinkerbell's PBnJ service handles BMC power on, off and reset ([github.com](#)), and the Redfish schema defines a graceful shutdown that powers the machine off ([redfish.dmtf.org](#)). No primary-source figure for the boot-to-ready time of a GPU server was found in this research, so the wake duration in the model below is an explicit planning assumption.

Case 3: The Node Serves Other Work

On shared owned fleets, the better use of freed capacity may be other workloads, not lower power. NVIDIA's GPU Operator supports time-slicing, with no memory or fault isolation between replicas ([docs.nvidia.com](#)), and priority-based preemption lets a scheduler evict lower-priority Pods to place higher-priority ones ([kubernetes.io](#)). A low-priority batch worker is a candidate for this reuse, and the value is capacity rather than energy.

Node Scale-Down in Clouds and Autoscalers

Cloud node deletion is a different calculation, because the avoided cost is the instance rate. Cluster Autoscaler can scale a node group to zero and back ([github.com](#)), treats accelerator nodes by GPU utilization only ([github.com](#)), and by default waits for a node to be unneeded for a configured time before removal ([github.com](#)). The Cluster API path is opt-in for infrastructure providers ([cluster-api.sigs.k8s.io](#)). Karpenter's consolidation timer resets whenever a pod is added to or removed from the node ([karpenter.sh](#)). Provider documentation is explicit about limits: on GKE Standard the autoscaler never scales a whole cluster to zero because system Pods need a node ([cloud.google.com](#)), and AKS allows, but does not force, autoscaling to zero on user node pools ([learn.microsoft.com](#)). GKE notes that GPU toleration lets GPU nodes scale down quickly when no Pods request GPUs ([docs.cloud.google.com](#)). Google also reports that a Cluster Autoscaler change removed a previous 30-second delay that affected GPU autoscaling ([cloud.google.com](#)), a reminder that node-level latency includes vendor-specific components that change between releases.

A combined timeline follows from these defaults. After the queue empties, the HPA needs at least the five-minute stabilization window before reaching zero, and a node autoscaler then waits for its own unneeded interval. With the Cluster Autoscaler's 10-minute scale-down delay after a scale-up (github.com) also in play, node removal can plausibly lag the last job by about 15 minutes or more. That is a derived estimate, not a documented figure. Idle windows shorter than that never reach node-zero.

Implementation Considerations and Process Changes

Decision Thresholds

The thresholds below translate the model into go or no-go checks. Each uses a measured input from the test plan that follows.

- **Deadline test.** Job deadline or client timeout must exceed detection latency plus p95 cold start plus margin; the p95 and not the mean, because averages hide tail latency (sre.google).
- **Gap test.** Mean or replayed idle gap must exceed T^* computed with measured power, boot time and SLO cost.
- **Durability test.** Queue retention must exceed the longest expected time at zero; defaults such as 4 days in SQS (docs.aws.amazon.com) or 168 hours in Kafka (kafka.apache.org) are far longer than typical idle windows, but must still be confirmed per queue.
- **Acknowledgement test.** Visibility timeouts, ack deadlines and retry policies must exceed cold start, so that a message is not redelivered while the first worker is still loading weights.
- **Signal test.** The external metric must be queryable at zero, and an alert must fire when it is not.
- **Probe test.** Startup probe budget must cover worst-case start.
- **Warm-replica test.** Keep `minReplicas: 1` where the latency SLO is shorter than the p95 cold start.
- **Node-zero test.** Only power nodes off where the power path, boot time and idle draw have all been measured on that hardware.

A Reproducible Test Plan

The test plan below is designed to be run on a staging cluster that mirrors production versions. Each test records timestamps, so every cold-start stage can be separated.

- **Baseline the idle power.** With persistence mode on, record per-GPU power with NVML, whose power-usage call reports milliwatts averaged over 1 second on recent GPUs (nvidia.github.io), in three states: no CUDA context, a loaded idle model, and active inference. Also record whole-node power at the power distribution unit or BMC with all GPUs bare-idle.
- **Run the cold-start waterfall twice.** Once with images and weights cached on the node, once after clearing them. Image garbage collection on the node can evict cached images, so a cache that works in testing may not survive production disk pressure.
- **Replay queue bursts.** Use recorded or synthetic arrival gaps, including long idle gaps, bursts and a gap just longer than the five-minute stabilization window.
- **Inject a metric outage.** Stop the adapter while the workload is at zero with queued work, then confirm detection by alert and recovery by restoring the metric.

- **Exercise version skew.** In a test cluster, enable the gate on the API server only, then on the controller manager only, and record behavior. Confirm the downgrade steps from the release post.
- **Test rollback.** Switch affected HPAs to `minReplicas: 1`, verify that workloads at zero scale to one, then disable the gate.
- **Verify buffering.** For any synchronous path, confirm what a client sees during the cold start. A Service gives no buffering.

Safe Rollout

A conservative sequence is: start with one durable-queue worker, set its HPA to `minReplicas: 1`, confirm the external metric through the External Metrics API, and only then lower the floor to zero. Keep the first rollout out of any path where a client waits. Alert on `ScalingActive=False` and on queue age rather than only queue depth, and size startup probes to the worst measured start. Caching and snapshot options exist, such as NIM Operator model caching (docs.nvidia.com) and vLLM sleep mode, which offloads weights to CPU RAM and discards the KV cache (docs.vllm.ai). These are mitigations to evaluate, not substitutes for measurement.

Air-Gapped Considerations

In an air-gapped cluster the cold-start stages move onto local infrastructure. The GPU Operator documents that all images must be hosted in a local image registry without internet access (docs.nvidia.com), and a pull-through registry such as Harbor's proxy cache can proxy and cache images (goharbor.io). In each case the local registry and storage become part of the scale-from-zero critical path and must be sized for simultaneous pulls.

Data Analysis and Evidence

The Break-Even Model

Define the net benefit of an idle interval of length g (hours) as:

net benefit = avoided idle cost – wake cost – SLO cost

where avoided idle cost is the idle cost rate r (dollars per hour) multiplied by the time actually spent in the lower state, wake cost W is the energy or billed time spent returning to service, and SLO cost S is the cost of deferred or lost work. The minimum idle interval at which the decision breaks even is:

$$T^* = (W + S) / r$$

For energy-only owned-hardware analysis, $r = \text{idle power (kW)} \times \text{PUE} \times \text{electricity price}$. PUE (power usage effectiveness) is total facility power divided by IT power; Uptime Institute's 2025 survey reported a weighted average of 1.54 (datacenter.uptimeinstitute.com). EIA data gives a US average commercial electricity price of 14.53 cents per kilowatt-hour for July 2026 (www.eia.gov). When wake power is approximately idle power, W / r is simply the wake duration, so the energy-only T^* equals the wake duration, consistent with a published model in which the energy break-even between keeping warm and cold-starting depends on arrival rate and loading latency (arxiv.org). That preprint reports break-even intervals of 1 to 5 minutes for keeping a model resident versus reloading it (arxiv.org).

Two refinements matter. First, the HPA's stabilization window means a cold start is only triggered by gaps longer than about five minutes. If gaps are exponentially distributed (Poisson arrivals) with mean G , the expected idle time saved per triggered cold start is exactly G by the memoryless property, so the sign of the net benefit depends on whether G exceeds T^* , independent of the window. Heavy-tailed or bursty arrivals break that shortcut, and the replay test above should compute the sum over observed gaps instead. Second, SLO penalties should be set on tail behavior: an average can obscure tail latencies ([sre.google](#)), and a quarterly error budget gives a way to price them ([sre.google](#)).

Inputs and Their Sources

Table 3 below applies the model to a **hypothetical** eight-GPU H100 node. Every input is either cited or marked as an assumption, and the results should be replaced with measurements.

(Hypothetical Example) Assumptions: PUE 1.54 ([datacenter.uptimeinstitute.com](#)); electricity \$0.1453 per kWh ([www.eia.gov](#)); pod-zero GPU saving of 50 W per GPU (inside the measured 26 to 66 W range ([arxiv.org](#))); node idle 1.86 kW ([arxiv.org](#)); pod wake energy of 8 GPUs at 200 W for 60 s (an assumption in the 100 to 300 W loading range the preprint validates, rounded); node boot of 10 minutes at idle power (a planning assumption, not a measurement); a cloud instance rate of \$47.757 per hour (AWS Capacity Blocks effective rate per eight-H100 instance, a reserved-capacity price and not on-demand ([aws.amazon.com](#))); SLO cost of \$0.50 per late job with one late job per cold start (an assumption).

Scenario (hypothetical, 8x H100)	Idle cost rate r (dollars per hour)	Wake cost W	T^* with $S = \$0$	T^* with $S = \$0.50$ per cold start	Reading
Pod-zero, node retained (50 W per GPU saved)	\$0.090	\$0.006	4.0 min	5.65 h	Energy saving is small; any SLO cost dominates
Pod-zero, node retained (26 W per GPU)	\$0.047	\$0.006	7.7 min	10.9 h	Lower bound of the measured range
Pod-zero, node retained (71 W per GPU)	\$0.127	\$0.006	2.8 min	4.0 h	Single production-telemetry observation, not a per-GPU maximum; the lab range is 26-50 W on HBM parts (H100, A100) and 66 W on the L40S
Owned node powered off (1.86 kW idle)	\$0.416	\$0.069 (10 min boot)	10.0 min	1.37 h	Better rate, but boot time and SLO cost set the threshold
Cloud node deleted , illustrative terminable rate set equal to the AWS Capacity Block effective rate	\$47.757	\$7.96 (10 min billed boot)	10.0 min	10.6 min	Rate swamps the SLO term; scale-down pays off quickly only if the instance is terminable (inside a Capacity Block the cost is sunk and there is no saving)

Table 3 yields three observations. First, the owned-hardware energy saving per hour from pod-zero alone (\$0.090) is about **530 times smaller** than the illustrative terminable cloud rate (\$47.757, taken from the AWS Capacity Block effective rate), so intuitions from cloud cost-cutting do not transfer to owned fleets. Second, a modest \$0.50 SLO penalty moves the pod-zero threshold from minutes to hours, meaning that for owned nodes the question is mostly about deferred work and not electricity. Third, powering an owned node off improves the rate roughly 4.6x relative to the 50 W case (\$0.416 versus \$0.090 per hour), but still needs a hardware power path and an unmeasured boot time. Annualized, an always-idle node's facility energy at these assumptions is about 25,092 kWh, or roughly \$3,646 per year, an upper bound on what node-off can save per node in energy alone.

Utilization Context

Idle GPU capacity is commonly large. Cast AI's 2026 report, a vendor-sponsored dataset, found GPU utilization averaged just 5% across the clusters analyzed (cast.ai). Datadog's 2024 State of Cloud Costs (data from May 2023 to April 2024) found that 83 percent of container costs are associated with idle resources, though not GPU-specific (datadoghq.com). These figures show where the idle is likely to be; they do not measure a saving available from scale-to-zero on any specific fleet.

Implications and Future Directions

For private inference operators, the immediate implication is classification before configuration. Queue-backed workers with deadlines longer than the measured cold start are straightforward candidates. Synchronous endpoints need either a warm replica or a buffering layer, and the choice between KEDA's HTTP add-on (keda.sh), Knative's activator (knative.dev) or a custom queue is an architecture decision, not an HPA setting.

The second implication is that pod-zero value on owned hardware is mostly a capacity story. Unless nodes are powered off, the measured energy delta is on the order of tens of watts per GPU (arxiv.org). The larger prize is putting freed GPUs to work for lower-priority jobs through admission and preemption (kueue.sigs.k8s.io), or reducing the number of nodes purchased. That argument depends on demand data the operator must collect.

Third, cold-start engineering has an outsized effect on eligibility. Parallel image pulls (aws.amazon.com), shared compile caches (blog.vllm.ai), streaming weight loaders (developer.nvidia.com) and snapshot restore (developer.nvidia.com) each shrink a stage. When they are combined with an honest tail-latency view of cold starts (sre.google), they widen the set of workloads that can safely sit at zero. Sleep-style approaches are also relevant: one vLLM benchmark reported five model switches taking 85 s with sleep mode versus 226 s with cold starts (blog.vllm.ai), although that test was on a small GPU and not a fleet.

On the platform side, the Beta label signals that field data is still being collected. Teams should watch for changes to the condition semantics, and for provider support in managed offerings. This research did not find a vendor-neutral measurement of native HPA scale-to-zero on GPU workloads; it should be treated as an open question.

The practical levers for reducing idle GPU cost, in rough order of implementation effort, are:

- **Classify and right-size first.** Identify workloads with durable queues and long idle gaps.

- **Keep one warm replica** for latency-sensitive services, as vendor guidance also recommends reserving scale-to-zero for endpoints where a 10-minute startup is acceptable ([cast.ai](#)).
- **Share GPUs** through MIG or time-slicing ([docs.nvidia.com](#)).
- **Reuse freed capacity** for preemptible batch work.
- **Cut cold start** by pre-pulling images and sharing compile caches ([blog.vllm.ai](#)).
- **Power nodes off** only after measuring the hardware path.
- **Track the result.** Re-run the break-even calculation quarterly, since images, models and arrival patterns change.

Frequently Asked Questions (FAQs)

Does Kubernetes 1.37 scale GPU pods to zero by default?

The capability is Beta and enabled by default ([kubernetes.io](#)), but an HPA must still set `minReplicas: 0` and use an object or external metric. Existing HPAs are not changed automatically.

What is the cold-start time for GPU inference after scaling from zero?

It depends on the stack. Published component measurements include 20.32 s for a 3B model with cached weights ([arxiv.org](#)), more than 40 s to first token for a cold-start instance in the authors' production serverless environment ([arxiv.org](#)), and a vendor claim of 8 to 15 minutes for large models with node provisioning ([cast.ai](#)). Measure your own.

How does KEDA compare with native HPA scale-to-zero?

KEDA owns the 0-to-1 activation and hands 1-to-N to the HPA ([keda.sh](#)), with its own polling and cooldown defaults ([keda.sh](#)). Native HPA now covers the same replica range using the External Metrics API. KEDA's HTTP add-on additionally buffers requests ([keda.sh](#)).

How can operators reduce idle GPU cost on Kubernetes without scale-to-zero?

Options include sharing a GPU with time-slicing or MIG ([docs.nvidia.com](#)), running lower-priority work in the freed capacity via preemption, and keeping a single warm replica for latency-sensitive services.

Is there a break-even calculator?

The formula is $T^* = (W + S) / r$. A spreadsheet with the inputs in Table 3 is enough; replace the assumptions with measured values.

Conclusion

Kubernetes 1.37 puts scale-to-zero into the core HPA, and for a durable-queue GPU worker that is a real simplification: one fewer add-on, with a clear status condition that separates an autoscaler-owned zero from a manual pause. It does not buffer requests, does not touch nodes, and does not make a cold start shorter.

The decision for an owned cluster therefore comes down to measured quantities. Idle energy avoided by pod-zero is tens of watts per GPU unless the node is powered off; the cold-start chain spans seconds to minutes depending on image, weights and runtime; and the SLO term can move the break-even interval from minutes to hours. A queue-backed workload with generous deadlines clears that bar easily. An interactive endpoint does not, unless a buffer or warm replica is added.

The recommended path is to classify workloads with the eligibility matrix, run the test plan on staging, compute T^* from measured inputs, and then lower `minReplicas` one workload at a time, with `minReplicas: 1` as the rollback position. No saving should be claimed until power, load-time and demand data from the actual fleet are in hand.

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/2026/08/26/kubernetes-v1-37-release>
2. <https://kubernetes.io/blog/2026/09/02/kubernetes-v1-37-hpa-scale-to-zero-beta>
3. <https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale>
4. <https://arxiv.org/abs/2605.23918>
5. https://www.eia.gov/electricity/monthly/epm_table_grapher.php?t=table_5_03
6. <https://arxiv.org/html/2412.08602>
7. <https://aws.amazon.com/ec2/capacityblocks/pricing>
8. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/capacity-blocks-pricing-billing.html>
9. <https://arxiv.org/pdf/2606.07362>
10. <https://arxiv.org/pdf/2502.15524>
11. <https://aws.amazon.com/blogs/containers/introducing-seekable-oci-parallel-pull-mode-for-amazon-eks>
12. <https://cast.ai/blog/kubernetes-gpu-autoscaling>
13. <https://github.com/kubernetes/enhancements/blob/master/keps/sig-autoscaling/2021-scale-from-zero/README.md>
14. <https://raw.githubusercontent.com/kubernetes-sigs/prometheus-adapter/master/docs/externalmetrics.md>
15. <https://llm-d.ai/docs/0.7/well-lit-paths/workload-autoscaling>
16. <https://kafka.apache.org/42/configuration/broker-configs>
17. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-message-lifecycle.html>
18. <https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html>
19. <https://keda.sh/http-add-on/0.16/user-guide/configure-cold-start>
20. <https://knative.dev/docs/serving/architecture>
21. <https://blog.vllm.ai/2025/08/20/torch-compile.html>
22. <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/install-gpu-operator-air-gapped.html>
23. <https://goharbor.io/docs/main/administration/configure-proxy-cache>
24. <https://www.rabbitmq.com/docs/quorum-queues>
25. <https://keda.sh/docs/2.17/reference/scaledobject-spec>

26. <https://keda.sh/docs/2.17/concepts/scaling-deployments>
27. <https://www.cncf.io/blog/2026/05/27/gpu-autoscaling-on-kubernetes-with-keda-building-an-external-scaler>
28. <https://knative.dev/docs/serving/autoscaling/scale-to-zero>
29. <https://kserve.github.io/website/docs/model-serving/predictive-inference/autoscaling/kpa-autoscaler>
30. <https://docs.ray.io/en/latest/serve/advanced-guides/advanced-autoscaling.html>
31. <https://prometheus.io/docs/prometheus/latest/configuration/configuration>
32. <https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md>
33. <https://cloud.google.com/blog/products/containers-kubernetes/tips-and-tricks-to-reduce-cold-start-latency-on-gke>
34. <https://docs.nvidia.com/deploy/driver-persistence/background.html>
35. <https://www.usenix.org/system/files/osdi24-fu.pdf>
36. <https://arxiv.org/html/2605.23918v1>
37. <https://developer.nvidia.com/blog/reducing-cold-start-latency-for-llm-inference-with-nvidia-runai-model-streamer>
38. <https://arxiv.org/abs/2505.23072>
39. <https://docs.nvidia.com/nim-operator/latest/nim-build.html>
40. <https://kubernetes.io/docs/concepts/workloads/pods/probes>
41. <https://developer.nvidia.com/blog/nvidia-dynamo-snapshot-fast-startup-for-inference-workloads-on-kubernetes>
42. <https://raw.githubusercontent.com/kubernetes-sigs/prometheus-adapter/master/README.md>
43. <https://www.nvidia.com/en-us/data-center/h100>
44. https://www.cs.princeton.edu/courses/archive/spring13/cos598C/Barroso07_EnergyProp-clean.pdf
45. <https://github.com/metal3-io/baremetal-operator/blob/main/docs/api.md>
46. <https://github.com/tinkerbell/pbnj>
47. <https://redfish.dmtf.org/schemas/v1/Resource.json>
48. <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html>
49. <https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption>
50. <https://cluster-api.sigs.k8s.io/tasks/automated-machine-management/autoscaling>
51. <https://karpenter.sh/docs/concepts/disruption>
52. <https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler>
53. <https://learn.microsoft.com/en-us/azure/aks/scale-cluster>
54. <https://docs.cloud.google.com/kubernetes-engine/docs/how-to/gpus>
55. <https://cloud.google.com/blog/products/containers-kubernetes/whats-new-with-gke-cluster-autoscaler>
56. <https://sre.google/sre-book/service-level-objectives>
57. https://nvidia.github.io/cuda-python/cuda-bindings/latest/module/generated/cuda.bindings.nvml.device_get_power_usage.html
58. <https://docs.nvidia.com/nim-operator/latest/cache.html>
59. https://docs.vllm.ai/en/latest/features/sleep_mode.html
60. <https://datacenter.uptimeinstitute.com/rs/711-RIA-145/images/2025.Annual.Survey.Report.pdf?version=0>
61. <https://sre.google/sre-book/embracing-risk>

- 62. <https://cast.ai/press-release/2026-state-of-kubernetes-optimization-report>
- 63. <https://datadoghq.com/state-of-cloud-costs>
- 64. <https://kueue.sigs.k8s.io/docs/overview>
- 65. <https://blog.vllm.ai/2025/10/26/sleep-mode.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.