

KUBERNETES GPU OBSERVABILITY

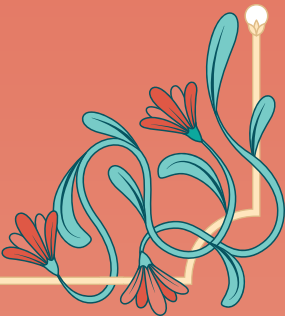
Vendor-Neutral GPU Observability

From the HAMi scheduler

Reza Jelveh

Solution Architect, Dynamia AI - Makers of HAMi

🐙 github.com/fishman 🌐 linkedin.com/in/rezajelveh



One question, many dashboards

The question

Are our accelerator workloads using the capacity we reserve efficiently?

The fragmentation

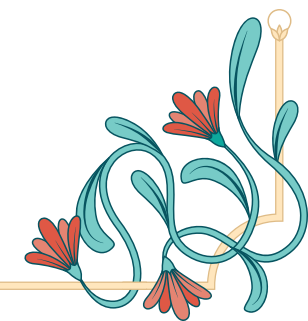
NVIDIA DCGM/NVML, AMD ROCm SMI, Intel XPU Manager and Ascend tooling expose different views.

The missing connection

Who requested this capacity? What was allocated? Which application benefits?

The architecture

Keep vendor sensors. Add a common workload identity and scheduling-intent layer.



GPU terms in one minute

CUDA context

A process's private workspace on the GPU: memory allocations, loaded kernels and work queues (streams).

SM (Streaming Multiprocessor)

The GPU's compute block. A GPU is many SMs; each kernel runs as thread blocks spread across them.

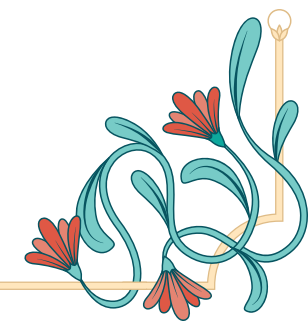
NVML

NVIDIA's library behind nvidia-smi: device state plus per-process samples keyed by host PID. It knows no Pods.

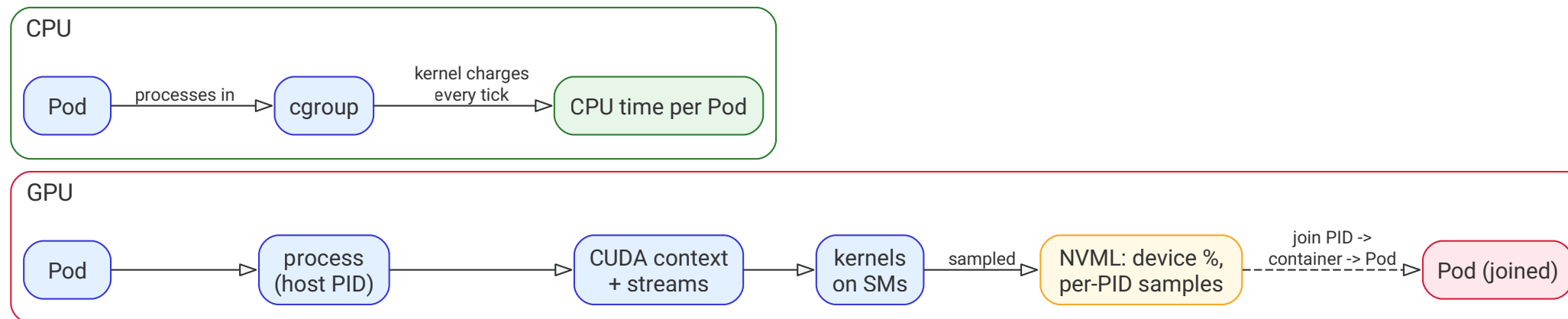
MIG (Multi-Instance GPU)

Hardware partitioning, Ampere and newer: up to seven instances, each with its own SMs and memory.

Source: *CUDA contexts*; *NVML API*; *MIG introduction*



Why GPU usage is hard to attribute



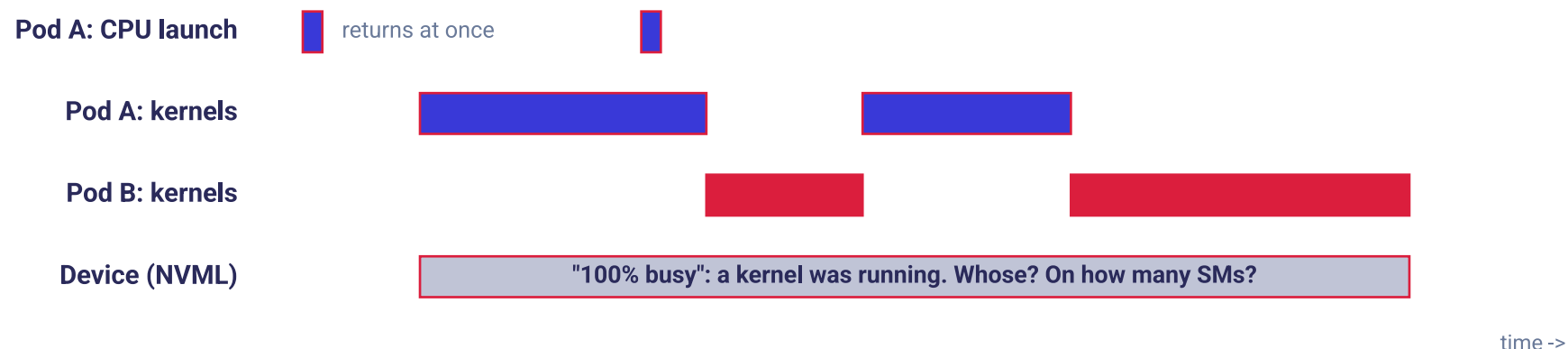
- ▶ CPU: the kernel charges time to the cgroup, so Pod usage is a direct reading.
- ▶ GPU: the device plugin grants access but reports no executed work.
- ▶ NVML sees devices and host PIDs; attribution to a Pod must be joined back.

Source: *Kubernetes device-plugin contract; NVML process queries*



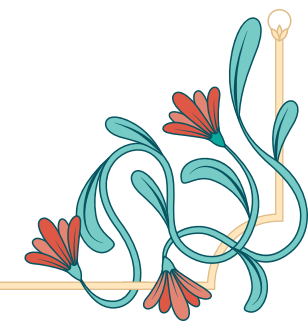
Queued GPU work blurs who used it

ILLUSTRATIVE TIMELINE, NOT A MEASUREMENT



- ▶ Starting GPU work only queues it. Timing that call on the CPU shows queuing time, not how long the GPU worked.
- ▶ Contexts from different Pods take turns; streams overlap inside one context.
- ▶ NVML "GPU utilization" means a kernel ran, not how many SMs were busy.

Source: *NVML utilization definition; CUDA streams and concurrency*



Same GPU, three different scopes

CUDA context

Memory and execution live in driver/runtime contexts. A Pod-to-device mapping does not reveal every operation.

MIG instance

Use instance/profile identity and partition capacity. Whole-card utilization can hide an idle or saturated slice.

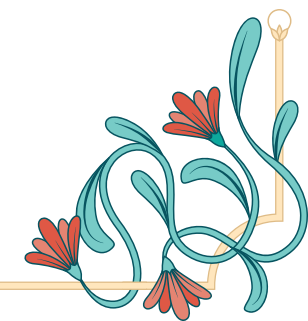
Soft-shared vGPU

Limits and accounting depend on the runtime interposition and sharing mode.

Comparison rule

Compare matching physical, partition or tenant scopes with documented units.

Source: *CUDA runtime/context interaction; MIG guide; MIG and accounting identities*



The common view operators need

GPU layer

Admission

Who requested what, what was reserved, and what was rejected or failed to fit?

Usage

How much of its reservation does each tenant use on a shared device?

Device

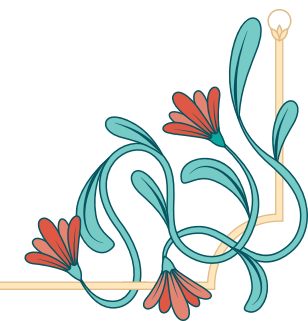
Memory, compute, power, temperature and errors: the hardware view.

Application layer

Associate, don't re-measure

Latency, throughput, queue depth and errors, joined to the GPU layer through workload identity.

Source: *scheduler capacity; workload allocations; outcome counters; tenant runtime*



What DCGM alone cannot answer

What it measures well

Whole GPU or MIG instance: memory, utilization, power, thermals, XID errors and profiling counters.

No per-tenant split

DEV_ fields describe the whole GPU or MIG instance; co-tenants on one card share one number. Time-sharing attribution is opt-in.

No budget or intent

No per-container memory limit, reserved share, sharing count or quota. Utilization has no reservation to compare against.

No decisions, one vendor

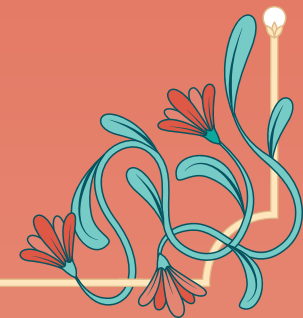
Pending, no-fit and rolled-back Pods never reach a GPU. AMD, Ascend and Hygon need separate stacks.

Source: *time-slicing limitation; dcgm-exporter flags; container limit and use; reservation and sharing; outcomes*



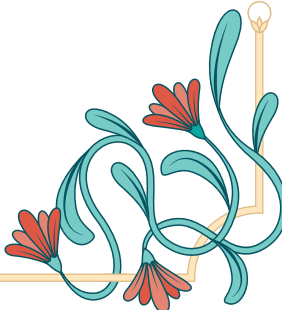
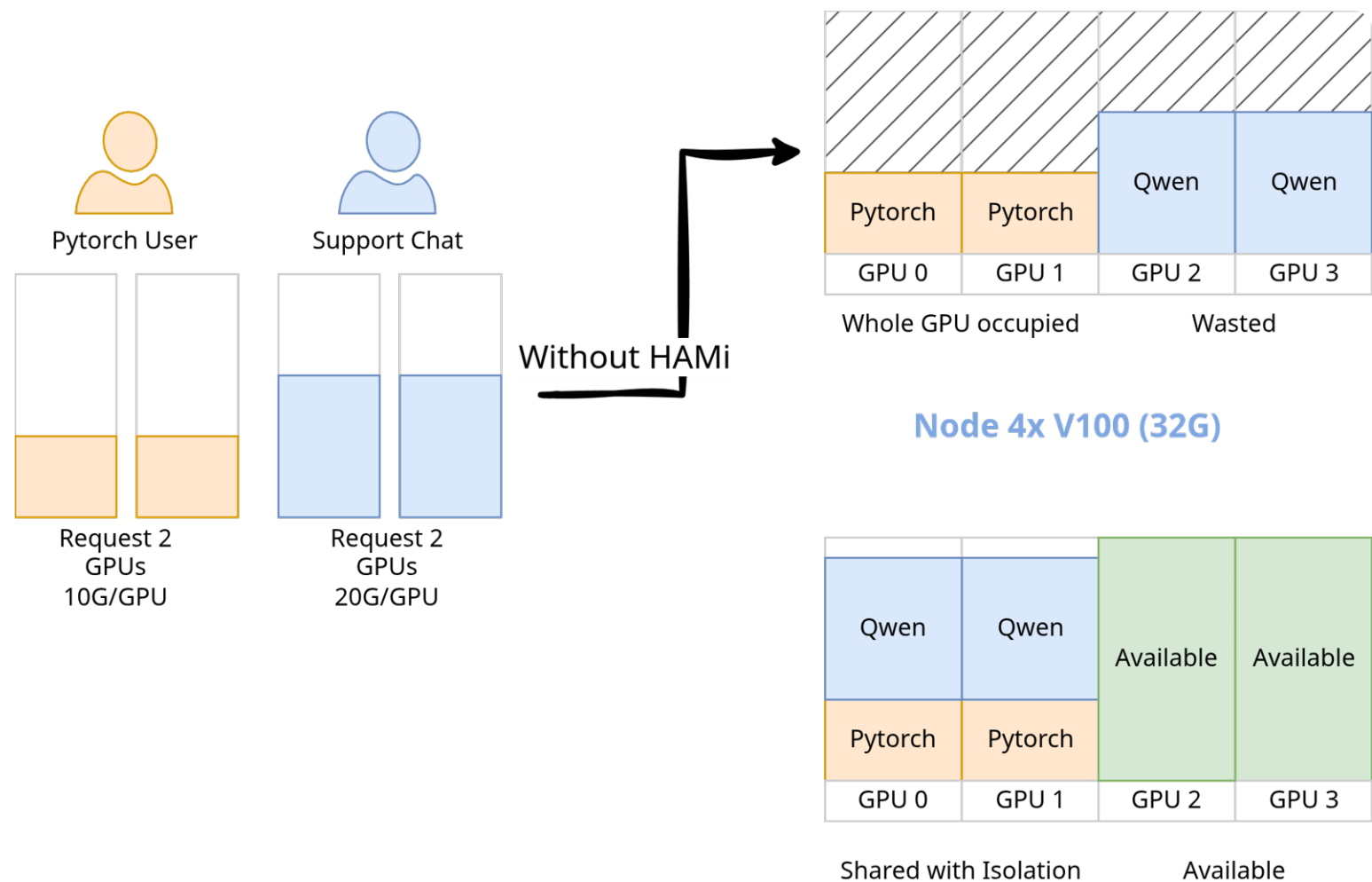
How does HAMi help

One scheduling plane across heterogeneous accelerators



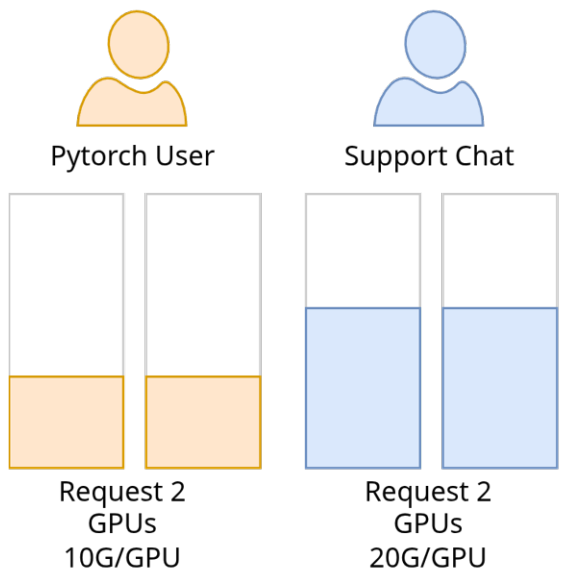
What is HAMi

Before: one device, one task

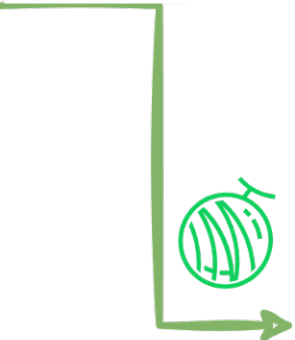


What is HAMi

After: one device, many agents



With HAMi



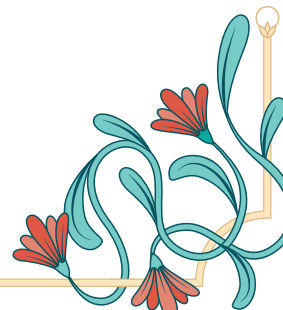
		Qwen	Qwen
Pytorch	Pytorch		
GPU 0	GPU 1	GPU 2	GPU 3

Whole GPU occupied Wasted

Node 4x V100 (32G)

Qwen	Qwen	Available	Available
Pytorch	Pytorch		
GPU 0	GPU 1	GPU 2	GPU 3

Shared with Isolation Available



HAMi Capabilities

Six things HAMi brings to GPU scheduling

Heterogeneous Management

Manage GPU, NPU, MLU, and other accelerators in one workflow.

Hard Isolation

Slice memory and compute with hard isolation at runtime.

Advanced Scheduling

Binpack, spread, and topology-aware placement policies.

Kubernetes Native

Kubernetes-native APIs, DRA, and CDI support.

Resource Isolation & QoS

Memory and core quotas for fair, stable sharing.

Unified Monitoring

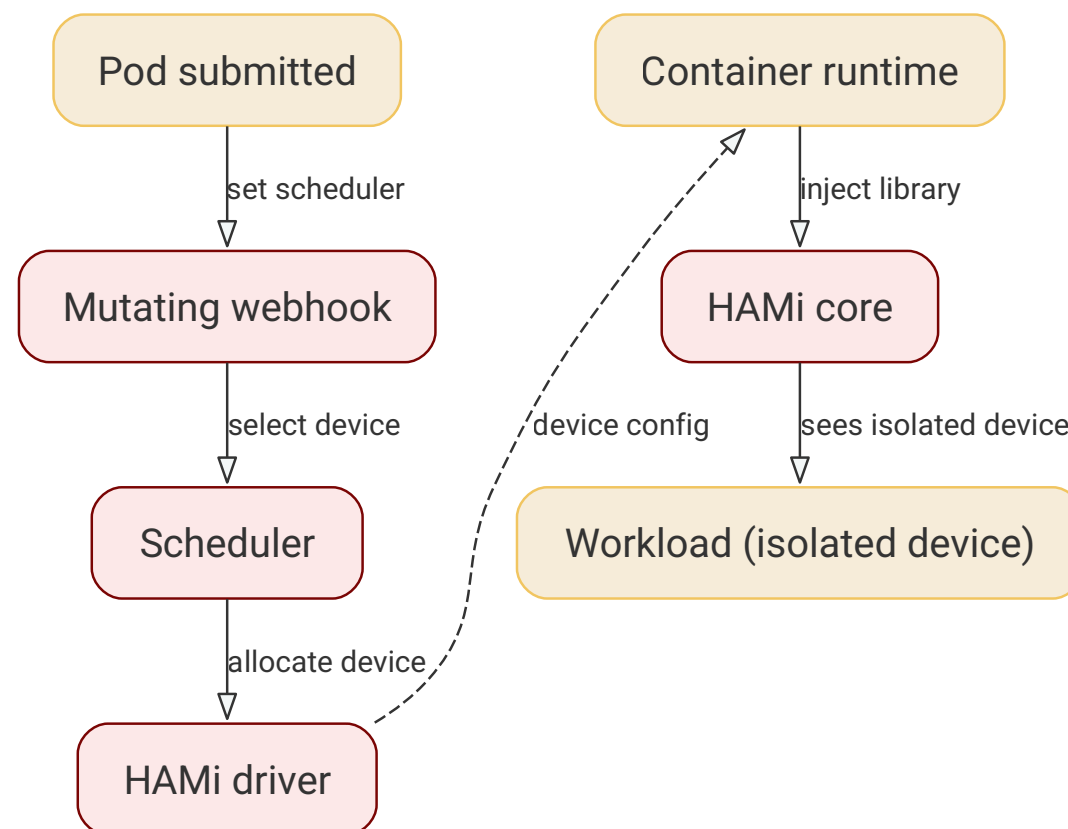
Consistent metrics and visibility across vendors.



How HAMi Works

From pod submission to isolated device

- ▶ **Mutating webhook:** sees accelerator requests, routes pod to HAMi scheduler
- ▶ **Scheduler:** selects device and node
- ▶ **HAMi driver:** generates device config
- ▶ **Container runtime:** reads config, injects HAMi-Core library
- ▶ **HAMi core:** enforces isolation in-process

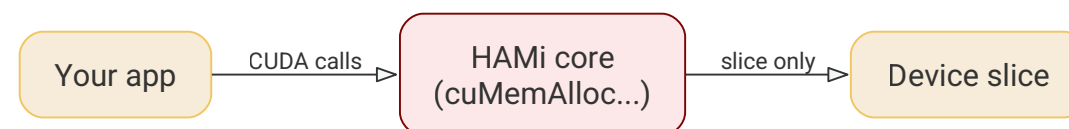


The Magic: Runtime Hijacking

Your app does not change

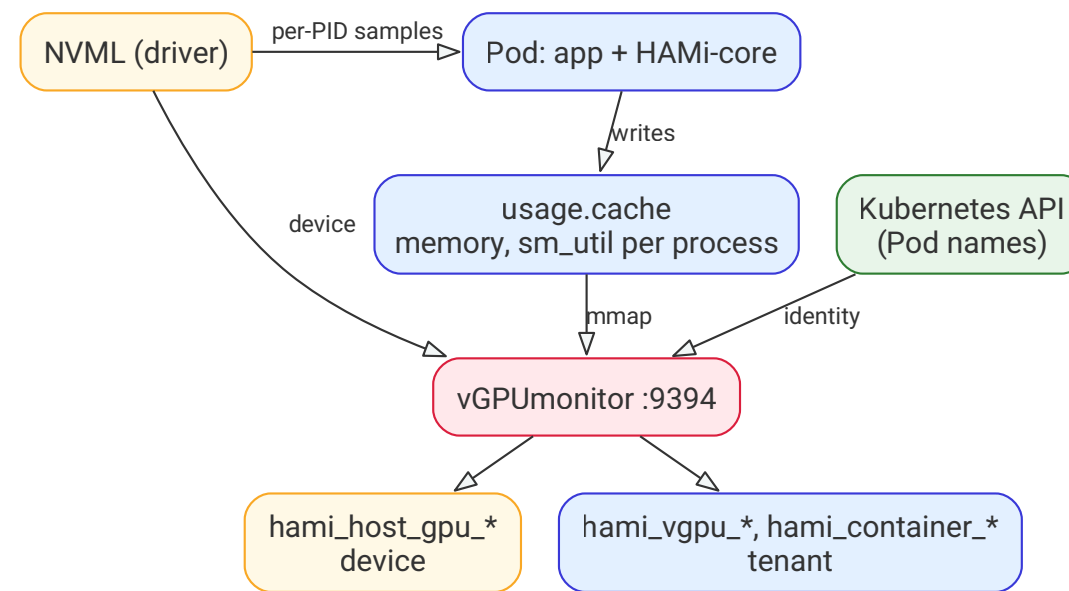
Your app calls CUDA. HAMi answers with a slice of the device.

- ▶ Small library loaded before your app (LD_PRELOAD)
- ▶ Intercepts CUDA calls, no app code changes
- ▶ No kernel modules, no driver changes
- ▶ Same pattern per vendor: ACL on Ascend, CNRT on Cambricon



How vGPUmonitor works on NVIDIA

- ▶ **Device view** from NVML: memory, utilization, power, temperature, ECC.
- ▶ **Tenant view** from HAMi-core's per-container cache: memory used vs limit, SM utilization.
- ▶ Joins both to Pods through the Kubernetes API and serves :9394/metrics.
- ▶ Tenant SM utilization is a share of the whole GPU, not of the reservation.



Source: *metric families; cache discovery; HAMi-core NVML samples*



NVIDIA has the fullest usage data

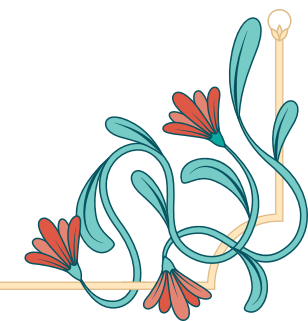
Physical device

NVML supplies used memory, GPU activity, memory-controller activity, temperature, power and ECC.

Shared workload

HAMi-core accounting supplies container memory and activity. Metrics carry namespace, pod, container and device UUID.

Source: *runtime metric contract*; *NVML collection*



What HAMi covers for each vendor

NVIDIA

Scheduler accounting plus NVML/shared-runtime collector code. Sharing mode still matters.

Ascend

Mode-gated tenant memory collector. vNPU tenant activity repeats card activity; WebUI task gap remains.

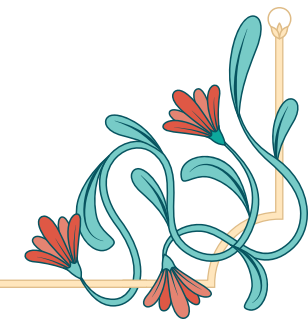
AMD

Allocation and plugin-health integration; first-party tenant runtime parity is not established.

Other accelerators

Require a compatible backend and telemetry adapter. A device plugin alone is insufficient.

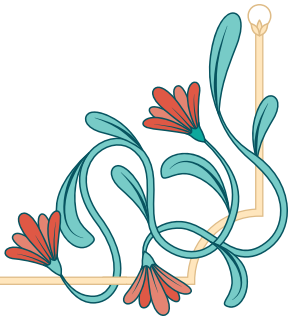
Source: *scheduler capacity; provider adapters; Ascend tenant semantics*



What the HAMi scheduler reports

- ▶ **Per GPU:** how much memory and compute HAMi can hand out, how much is already reserved, and how many containers share it.
- ▶ **Per Pod and namespace:** what each Pod reserved on which GPU, and how much of each namespace's GPU quota is used.
- ▶ **Scheduler health:** is this replica in charge, has it loaded cluster state, and why did placements fail (no GPU fits, lock, bind error)?
- ▶ **AMD:** compute units are converted to a percentage, so they compare with other vendors.
- ▶ These describe **placement and reservation**. Measured GPU work comes from runtime collectors (vGPUmonitor on NVIDIA); Prometheus keeps the history.

Source: *AMD normalization; allocation families; pod reservations; namespace quotas; leader and cache sync; outcome counters; measured work*



Reserved is not the same as used

Reserved memory

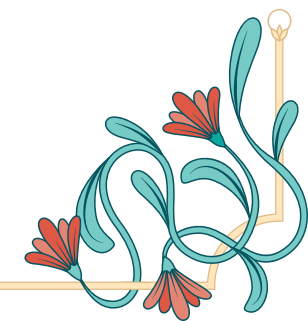
`hami_vgpu_memory_allocated_bytes`
 Scheduler reservation at Pod/device scope.

Consumed memory

`hami_vgpu_memory_used_bytes`
 Runtime accounting at container/vdevice scope.



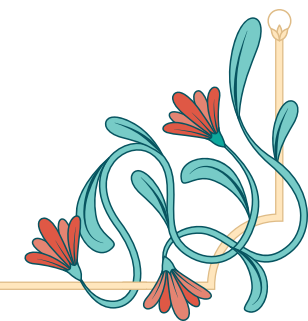
Source: *workload allocations; tenant runtime metrics*. Screenshot: Grafana on a real cluster.



Pattern 1: find unused reservations

- ▶ Compare reserved memory with observed memory over a representative window.
- ▶ Require fresh, supported runtime data and a positive limit.
- ▶ Separate idle periods, model loading and steady-state demand.
- ▶ Rank sustained headroom alongside pending or rejected workloads.
- ▶ Treat low compute activity as a separate signal, not proof of reclaimable memory.

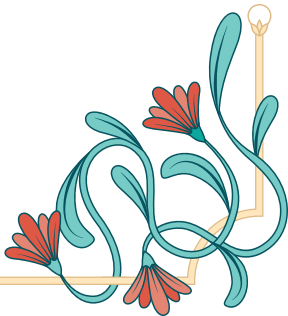
Source: *tenant runtime metrics; workload allocations*



Pattern 2: track down noisy neighbors

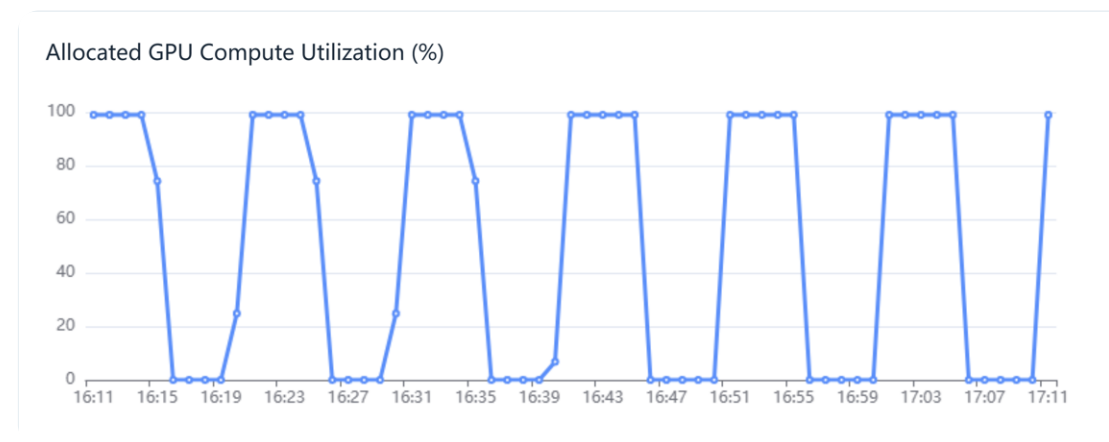
- ▶ Identify co-located tenants and the physical or slice identity they share.
- ▶ Correlate rising application latency with sibling activity and memory pressure.
- ▶ Check CPU, network, queueing and thermal throttling as alternatives.
- ▶ Repeat with controlled sibling load or isolated placement.
- ▶ Card-wide activity cannot identify which tenant caused contention.

Source: *scheduler capacity; tenant runtime metrics*



Pattern 3: right-size GPU requests

- ▶ Measure startup peaks and steady-state demand across the workload lifecycle.
- ▶ Segment by model, batch size, request load and sharing mode.
- ▶ Choose memory headroom from peaks and failure risk.
- ▶ Tune compute limits only after normalizing quota and activity semantics.
- ▶ Canary the new request; compare latency, failures and placement outcomes.

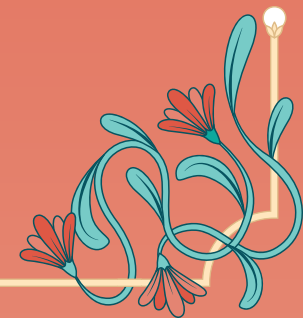


Source: *tenant runtime metrics; scheduling outcome counters*. Screenshot: HAMi-WebUI 1.3.0 on a single-node A30 cluster.



Advice for platform builders

Cross-vendor GPU observability, with or without HAMi



What an operator dashboard needs

Allocation

Reserved/total memory, sharing count, placement failures and pending demand.

Runtime

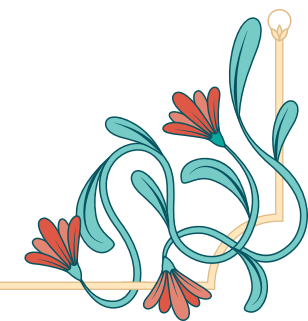
Consumption/limit, scoped activity, memory errors and freshness.

Application

Latency, throughput, queue depth and error rate, by stable workload identity.

Confidence

Measured or derived? Supported on this vendor/mode? Current or stale?



OpenTelemetry can carry all of it

Existing inputs

HAMi Prometheus allocation metrics plus compatible runtime metrics.

Collector pipeline

A Prometheus receiver can scrape, enrich and export metrics through OTLP.

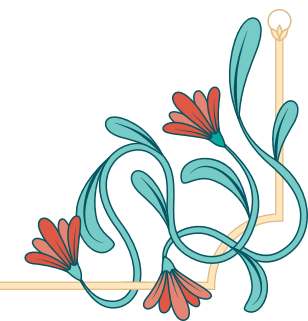
Admission and scheduler events

Proposed: request, policy decision and rejection; then reservation, no-fit and bind rollback.

Common attributes

Workload identity, device/slice identity, scope, unit, source and freshness.

Source: *OpenTelemetry Collector; Prometheus receiver; scheduling outcome counters*



One metric contract for every vendor

Stable identity

Cluster, node, vendor, physical device, slice, workload UID and container.

Explicit semantics

Reservation vs consumption; bytes vs MiB; ratio vs percent; physical vs slice vs tenant.

Capability and freshness

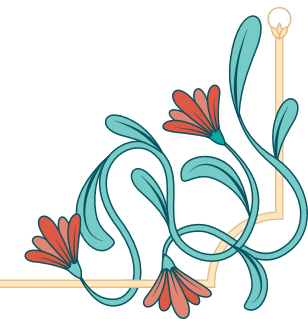
Measured, derived, unsupported or stale. Export the collection timestamp and error state.

Low cardinality

Report per container, not per process. Keep changing values like PIDs, sizes and limits out of labels.

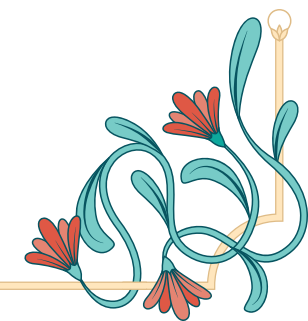
Today: `_ratio` means 0-100 at runtime but 0-1 in the scheduler; scheduler memory is bytes, WebUI memory is MiB.

Source: *0-100 contract*; *MiB conversion*; *0-1 contract*; *WebUI MiB*



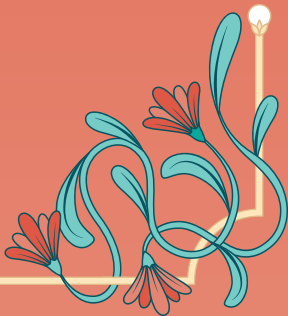
Takeaway: start from the workload

- ▶ Scheduling supplies ownership, intent and allocation decisions.
- ▶ Runtime sensors supply consumption and device behavior.
- ▶ Their correlation enables overprovisioning and contention investigations.
- ▶ OpenTelemetry can transport the model and decision events.
- ▶ The pattern applies to other schedulers and device integrations.



Appendix: what each vendor provides

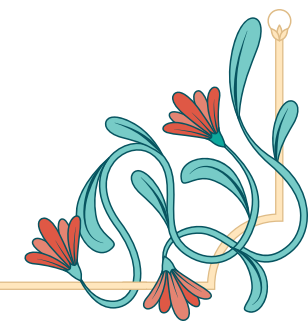
Exporters, gaps and setup, vendor by vendor



Ascend: metrics only in some modes

- ▶ HAMivNPUCore and ENPU modes expose :9395/metrics.
- ▶ Tenant memory comes from shared accounting or DCMI process attribution.
- ▶ vNPU tenant utilization repeats **physical AICore activity**.
- ▶ Context/module/buffer memory fields are zero placeholders.
- ▶ WebUI physical NPU adapters exist; task adapters remain unsupported.

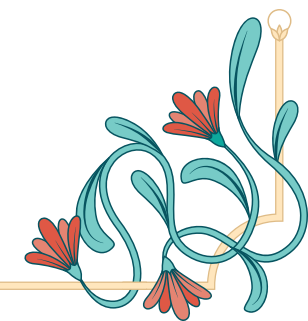
Source: *mode gate*; *tenant semantics*; *WebUI gap*



Hygon: exporter and WebUI disagree

- ▶ DCU exporter serves /metrics on default port **16080**.
- ▶ Physical and virtual gauges poll DCGM every **10 seconds**.
- ▶ Exporter: vdcu_utilizationrate, vdcu_usedmemory_bytes.
- ▶ WebUI expects vdcu_percent, vdcu_usage_memory_size and different labels.
- ▶ Validate or bridge the workload contract before demoing WebUI trends.

Source: *exported virtual families; polling and endpoint; compute query; memory query*



AMD and Biren: no per-Pod usage data

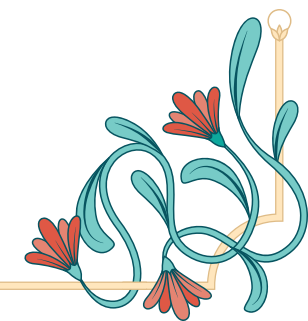
AMD

ROCm/AMD plugin health integration exists. Scheduler reservations are visible. No AMD-specific WebUI telemetry adapter in the reviewed switch.

Biren

Device registration, allocation and health logic need a separate exporter contract. No Biren WebUI telemetry branch in the reviewed switch.

Source: *AMD accounting; implemented provider branches*



Cambricon and MetaX: vendor exporters

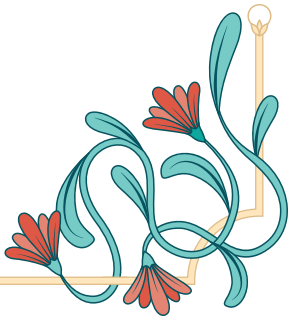
Cambricon

WebUI queries `mlu_*` series and joins container mapping on UUID. Workload attribution inherits exporter semantics.

MetaX

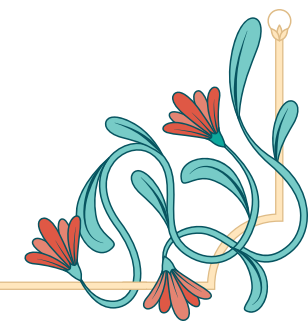
WebUI queries `mx_*` physical and workload series. GPU and sGPU paths have different identity labels and assumptions.

Source: *provider queries; conversion and fallback; workload memory*



Other vendors: metrics not yet checked

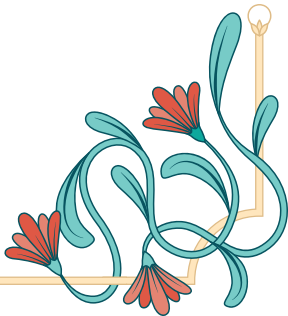
- ▶ HAMi documents more hardware than WebUI has telemetry adapters.
- ▶ Iluvatar, Enflame, Kunlunxin, Mthreads and other backends need exporter-by-exporter review.
- ▶ Check physical health, runtime usage and workload attribution independently.
- ▶ Label an unverified capability **unknown**, not zero or supported.



WebUI: one format, not one meaning

- ▶ Uses Kubernetes allocation state plus Prometheus vendor queries.
- ▶ Translates each vendor's units and label names into one set of hami_* metrics, so one query works for every vendor.
- ▶ If a Pod's compute share can't be determined (some Ascend Pods today), it reports "unknown" instead of guessing.
- ▶ Exposes refresh health and last-success timestamp.
- ▶ Common names do not guarantee common measurement semantics.

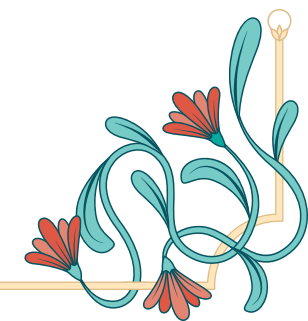
Source: *unit conversion; refresh health; known/unknown and usage; semantics*



DRA has its own allocation metrics

- ▶ HAMi-DRA serves allocation metrics at `:8080/metrics`.
- ▶ Capacity comes from ResourceSlices; reservations from ResourceClaims.
- ▶ Current families use `hami_dra_*` names and core ratios **0-1**.
- ▶ Kubelet-driver health and Prepare/Unprepare are a separate surface.
- ▶ Runtime device and tenant telemetry still require their own collectors.

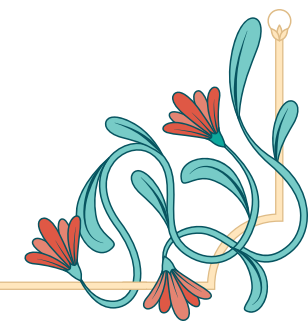
Source: *DRA descriptors; normalization; endpoint setup*



Make sure Prometheus scrapes HAMi

- ▶ HAMi provides separate scheduler and runtime ServiceMonitors.
- ▶ Chart rendering requires the ServiceMonitor CRD and `prometheus.enabled`.
- ▶ Runtime monitor also requires the device plugin to be enabled.
- ▶ Both templates set `honorLabels: true`.
- ▶ Verify target selection, labels and actual series after installation.

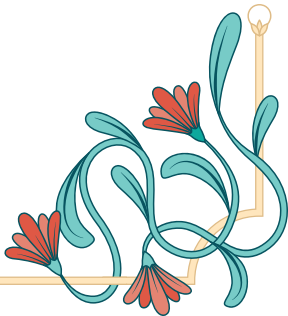
Source: *scheduler scrape*; *runtime scrape*



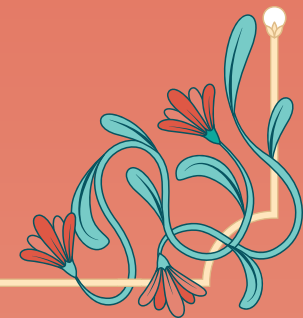
Mismatched versions break dashboards

- ▶ Current hami_* names coexist with opt-in legacy descriptors.
- ▶ Names and labels changed together: deviceuuid versus device_uuid.
- ▶ Some older guides expect legacy families; current Grafana JSON uses hami_*.
- ▶ Pin exporter, scheduler, WebUI and dashboard versions together.
- ▶ Test missing series as well as successful queries.

Source: *legacy runtime descriptors; current NVIDIA selector*



Where we are today



Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Backed, Production Ready

5.2k

Github Stars

325k

Docker Pulls

500+

Contributors

27

Contributor Countries



Ecosystem & Device Support



Adopters



THANK YOU

Questions? Try HAMi

github.com/Project-HAMi/HAMi

Got devices we do not support yet? We would love to play with them.

Reza Jelveh

Solution Architect, Dynamia AI - Makers of HAMi

🐙 github.com/fishman 🌐 linkedin.com/in/rezajelveh

Thanh Loi Hoang

Research Engineer, CNLab.ai | LFX Mentee, HAMi

🐙 github.com/loiht2 🌐 linkedin.com/in/loiht2

