GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

DCGM Exporter GPU Monitoring: Metrics, Alerts, Triage

> 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 · 2026-09-27 · dcgm exporter · gpu monitoring · prometheus metrics · nvidia dcgm · gpu alerts · grafana dashboard · mig monitoring · gpu fleet observability

Original article: https://gpusmith.com/articles/en/dcgm-exporter-gpu-monitoring-metrics-alerts-triage

In this article

1. Executive Summary
2. Introduction and Background
3. Monitoring Architecture and Field Taxonomy
4. DCGM Fields Mapped to Operator Decisions
5. Alert Semantics and Routing
6. Kubernetes, MIG, and Dashboard Design
7. Triage Runbook and Validation
8. Dashboard and Alert Coverage Checklist
9. Data Analysis and Evidence
10. Implications and Future Directions
11. Frequently Asked Questions (FAQs)
12. Conclusion

Executive Summary

DCGM Exporter turns selected NVIDIA Data Center GPU Manager (DCGM) fields into Prometheus metrics. A useful GPU monitoring design starts with the operator decision each field can support: detect missing telemetry, identify a vendor-defined health condition, investigate a change in cumulative errors, or explain workload saturation. NVIDIA documents **one exporter per monitored GPU node**; it supports a service, container, or Kubernetes DaemonSet, and a working endpoint exposes `DCGM_` metrics. (docs.nvidia.com) For Kubernetes, GPU-to-pod mapping comes from the kubelet pod-resources socket, so a missing pod label does not necessarily mean a missing GPU sample. (kubernetes.io)

The field's **meaning**, the collector's **Prometheus type**, and its **reset scope** must be checked separately. The shipped collector configuration can mark a field `counter` or `gauge`, but the exporter does not validate that choice against DCGM semantics. A temperature or framebuffer reading is a state that can rise or fall; an energy total is cumulative but resets on driver reload; an XID field carries a specific error code, rather than an accumulating event count. (prometheus.io) (docs.cloud.google.com) Optional exporter-owned XID totals count observed records and reset at startup or collector rebuild. They should not be read as a lifetime incident tally.

This report proposes a conservative **page, ticket, dashboard** classification. Page when telemetry is missing from an expected production GPU, or when a vendor-defined health result or corroborated hardware condition needs immediate operational action. Ticket a sustained change in error counters, persistent remap state, or workload-capacity trend after local calibration. Keep utilization, temperature, and power on dashboards unless an application impact or device-specific limit confirms urgency. A DCGM `Healthy` result covers only enabled watches and retained data, while profiling activity is an interval average rather than a kernel trace. (docs.nvidia.com) Vendor integration examples also describe configurable rather than universal thresholds. (docs.datadoghq.com) (docs.datadoghq.com) (cloud.ibm.com)

The implementation requires three independent checks: local `/metrics` output, Prometheus target and series presence, and alert delivery. Prometheus `up` distinguishes a failed scrape from a healthy one, while a separate absent-series rule detects a dropped DCGM family on a reachable endpoint. (prometheus.io) (prometheus.io)

A Grafana panel can explore a variable namespace or GPU, but Grafana alert queries do not accept dashboard template variables; production rules need explicit label matching and routing. (grafana.com) (grafana.com) The catalog, example queries, and checklist below give a starting design. **All numeric alarm values remain local placeholders** because hardware, workload, sample cadence, and business impact determine sensible thresholds.

Introduction and Background

Graphics processing unit (GPU) fleets expose many measurements, but an operator needs a small set of decisions: whether to wake someone, create a ticket, inspect a dashboard, or collect evidence for a later capacity review. DCGM supplies typed fields and retained samples; **DCGM Exporter** selects fields and exposes their latest values for Prometheus. A field read normally returns a cached sample rather than initiating a fresh hardware query, so scrape frequency and source sampling frequency are different clocks. (docs.nvidia.com) A fast HTTP scrape does not make an older DCGM observation fresh.

The scope here is **production telemetry and incident triage** on bare-metal and Kubernetes private AI fleets. For GPU fleet monitoring metrics, it covers DCGM exporter Prometheus metrics, NVIDIA GPU monitoring, Grafana dashboards, alert rules, health fields, memory and utilization, and a practical runbook. It does not prescribe a universal temperature, utilization, or ECC threshold. A chart of high temperature alone does not establish a fault; a low utilization chart alone does not establish wasted capacity. Profiling fields are averages over intervals and need workload context, including throughput and latency, to explain a bottleneck. AWS recommends observing GPU metrics alongside serving and application metrics for that reason. (aws.amazon.com)

GPU Smith is an **adjacent engineering advisor**, not a DCGM vendor. Its stated operational scope includes capacity planning, telemetry, and procedures for facilities, which makes a reproducible field-to-action record relevant to its audience. (gpusmith.com) Its air-gapped infrastructure scope also makes local collection and evidence retention operationally relevant; the monitoring guidance below is independent of that service description. (gpusmith.com)

Version-sensitive details in this report are checked **as of 27 September 2026**. The shipped collector file is a release artifact: an active CSV row is enabled, whereas a commented row is only an optional example. A configured row still may not appear if the DCGM field, entity, device, driver, or permissions do not support it. This is why every installation should inspect actual `/metrics` output before importing a dashboard or enabling an alert.

Monitoring Architecture and Field Taxonomy

Collection boundaries and failure modes

On a bare-metal host, the exporter can run as a system service or container. Oracle's deployment example places a container on every monitored GPU node, while IBM recommends a restart mechanism for a standalone container after host reboot. (docs.oracle.com) (cloud.ibm.com) In Kubernetes, a DaemonSet or

GPU Operator installation can place it on selected GPU nodes. Azure documents automatic scraping of new GPU node pools after its monitoring configuration is applied. (learn.microsoft.com) These are examples of the same node-local collection boundary, not a requirement to copy either platform's complete stack.

A healthy path has several links: GPU and driver, DCGM watch, exporter process, HTTP endpoint, scrape target, time-series ingestion, rule evaluation, and notification routing. The NVIDIA chart disables Go, process, and HTTP instrumentation families by default, so their absence alone is not proof that the GPU collector stopped. (github.com) An `up == 0` alert points to the scrape boundary; a successful `up == 1` with no expected DCGM series points toward field selection, entity support, or collection. Prometheus documents `up` as the reachability signal, while `absent_over_time()` detects missing series in a chosen range. Grafana-managed alerts have a separate **No Data** state when a query succeeds but returns nothing, which should receive an explicit policy rather than silently appearing healthy. (grafana.com)

- **Node scope:** confirm exporter placement on every expected GPU node and monitor the target itself.
- **Watch scope:** distinguish DCGM cache refresh from Prometheus scrape cadence; record both values in the runbook.
- **Field scope:** compare collector configuration with the live metric inventory on each GPU model.
- **Attribution scope:** distinguish hardware identity from the pod or container currently assigned to the device.
- **Delivery scope:** inspect Prometheus evaluation and Alertmanager grouping, silencing, and routing. (prometheus.io)

Counter, gauge, and event-like field

A **gauge** represents a present value that may rise or fall, such as board power, GPU temperature, or used framebuffer. A **counter** is cumulative within a defined reset scope; Prometheus `rate()` and `increase()` compensate for detected resets. (prometheus.io) (prometheus.io) An **event-like field** may encode the last observed condition, as XID does. Treating an XID value as a counter, or treating a pending boolean as a cumulative count, creates misleading alert logic. The NVIDIA field catalog calls XID the specific error value and describes pending page retirement as a yes/no state.

There are two further caveats. First, a collector CSV type is an export declaration, not a semantic proof. Second, reset behavior is field-specific: NVIDIA says the energy total is since the last driver reload, while its exporter-owned XID totals are process state and reset on startup or reload. The correct rule for any customized field should be written only after its field definition and live `# TYPE` line agree with the intended PromQL function.

DCGM Fields Mapped to Operator Decisions

Table 1 is a **dated field-to-action catalog, checked 27 September 2026**. “Local window” means a value to set after observing this hardware and workload; it is not a vendor threshold. Field IDs are NVIDIA DCGM numeric identifiers where verified. “Scope” distinguishes a physical device from a potentially mapped GPU instance. A command is a first read-only triage step, not a diagnosis.

Signal and metric name (exporter alias where applicable; DCGM field ID where shown)	Unit, type, reset or state	Scope and proposed evaluation	Local action and first triage step	False-positive or availability caveat
Exporter target `up`	Binary scrape status; Prometheus-generated gauge (prometheus.io)	One discovered target; local failure window	**Page** if scrapes of an expected production target keep failing; inspect target and exporter logs	A drained node or maintenance window must be excluded.
GPU temperature, field **150**, `DCGM_FI_DEV_GPU_TEMP`	Degrees Celsius; gauge (docs.nvidia.com)	GPU; trend over local workload window	**Dashboard** by default; inspect `nvidia-smi -q` and cooling context	Temperature alone does not prove throttling; compare thermal margin, field **153**. (docs.nvidia.com)
Board power, field **155**, `DCGM_FI_DEV_POWER_USAGE`	Watts; gauge (docs.nvidia.com)	GPU; workload-aligned window	**Dashboard**, or ticket on a sustained unexplained shift; inspect `nvidia-smi -q`	A lower reading may simply reflect a lighter job.
Used framebuffer, field **252**, `DCGM_FI_DEV_FB_USED`	Megabytes; gauge (docs.nvidia.com)	GPU or supported instance; local headroom window	**Ticket** before expected allocation pressure; compare workload request and free memory	Used memory is not identical to application allocation or imminent out-of-memory.
Total energy, field **156**, `DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION`	Millijoules; cumulative; resets on driver reload (docs.nvidia.com)	GPU; use a rate over a local window	**Dashboard** for energy accounting; inspect driver restart history	Do not interpret a reset as negative consumption.
PCIe replay, field **202**, `DCGM_FI_DEV_PCIE_REPLAY_COUNTER`	Retries; cumulative counter (docs.nvidia.com)	GPU; rate or increase over local window	**Ticket** when growth persists with workload impact; inspect `nvidia-smi -q` and host PCIe logs	A single historical total is not a current incident.
XID, field **230**, `DCGM_FI_DEV_XID_ERROR`	Specific code; event-like value (docs.nvidia.com)	GPU; new observation plus code context	**Page or ticket by code and impact**; inspect host driver logs	Last-value gauges can repeat a prior event; do not use `rate()` on the code.
Persistent double-bit ECC, field **313**	Aggregate error count; monotonic per field definition (docs.nvidia.com)	GPU; increase over local window	**Page** if vendor health and workload impact corroborate; inspect DCGM health and logs	Hardware support and reset behavior must be checked on the exact device.
Pending retirement, field **392**	Boolean, **1** means pending (docs.nvidia.com)	GPU; state confirmation window	**Ticket**, escalate with vendor guidance; inspect `nvidia-smi -q`	A collector may declare an unsuitable Prometheus type, so inspect live `# TYPE`. (docs.nvidia.com)

Signal and metric name (exporter alias where applicable; DCGM field ID where shown)	Unit, type, reset or state	Scope and proposed evaluation	Local action and first triage step	False-positive or availability caveat
Row-remap failure, field **395**	Failure state; not an event rate (docs.nvidia.com)	GPU; confirm across samples	**Page** if vendor health marks the device unsuitable; inspect DCGM health	A missing series can mean unsupported telemetry, not a zero value. (docs.nvidia.com)

The table intentionally avoids a fixed “GPU above X degrees” alarm. NVIDIA provides a **thermal margin** field that is closer to a device-specific slowdown boundary than raw temperature, while Red Hat's documented dashboard labels a displayed maximum temperature as empirical rather than a hardware limit. (docs.redhat.com) Likewise, the AWS example exports GPU utilization as a gauge, PCIe replay as retries, and XID as the last code; those three field families demand different queries despite sharing a DCGM prefix. (aws.amazon.com) (aws.amazon.com) (aws.amazon.com)

Health, interconnect, and saturation beyond the first table

- **DCGM health:** the opt-in `DCGM_EXP_GPU_HEALTH_STATUS` family is per watched subsystem and GPU. A `Healthy` result means no enabled rule found an issue in retained data; it does not prove all watches were enabled. Some watches need about **60 seconds** of observations before an initial check is meaningful. (docs.nvidia.com)
- **NVLink and PCIe:** use link error counters and throughput with application communication patterns. A throughput dip may reflect a different collective or a quieter workload; replay growth requires trend and host context. AWS exposes PCIe retries in a production telemetry example.
- **Workload saturation:** use GPU utilization, framebuffer occupancy, power, and optional profiling activity together with request rate, token throughput, or training progress. NVIDIA says profiling metrics are interval averages, and Red Hat's dashboard shows graphics-engine active time as a distinct view. (docs.redhat.com)
- **Unsupported fields:** treat absence explicitly. Enabling a CSV row or Kubernetes labels cannot make a GPU unsupported field appear.

Alert Semantics and Routing

PromQL templates with local placeholders

The templates below show **shape**, not production values. For each expected GPU node, record the actual exporter `instance` label in an inventory independent of scrape discovery and instantiate both availability checks for that endpoint. Replace the metric names with those observed at the endpoint, set windows longer than the effective DCGM watch and Prometheus scrape cadence, and record a hardware-specific rationale for every placeholder. `increase()` is for a genuine cumulative series and adjusts for resets; `rate()` should be applied before summing across devices. (prometheus.io) (prometheus.io)

```
# Failed scrape while the expected exporter target remains discovered
up{job="dcgm-exporter",instance="EXPECTED_EXPORTER_HOST:9400"} == 0

# Expected exporter target absent from Prometheus for the local window
absent_over_time(up{job="dcgm-exporter",instance="EXPECTED_EXPORTER_HOST:9400"}[LOCAL_WINDOW])

# A true cumulative PCIe replay counter, after confirming live # TYPE
increase(DCGM_FI_DEV_PCIE_REPLAY_COUNTER[LOCAL_WINDOW]) > LOCAL_REPLAY_DELTA

# Current framebuffer pressure, only after local capacity calibration
DCGM_FI_DEV_FB_USED{job="dcgm-exporter"} > LOCAL_MB_LIMIT
```

A removed target eventually has no current `up` series, so `up == 0` alone cannot detect its disappearance; evaluate the absence check against the independent expected-node inventory. Prometheus alert rules can add a `for` duration so a condition remains pending until it persists; use a separate `keep_firing_for` only if the notification policy needs it. (prometheus.io) Route and deduplicate by stable hardware identity, then include node, GPU UUID, owner, time window, and a runbook URL in alert annotations. Prometheus supports runbook links in annotations, and Alertmanager handles grouping and routing. (prometheus.io) (prometheus.io) Do not copy a dashboard query containing Grafana template variables directly into a Grafana-managed alert: alert queries do not support those variables. (grafana.com)

Table 2 gives a **proposed routing policy**, rather than NVIDIA's vendor-defined severity classification. The local page rule must include workload criticality and a contactable owner. Where there is no confirmed production ownership, the same signal may belong in a ticket queue.

Condition	Page	Ticket	Dashboard only
Discovered exporter scrape failure	Sustained scrape failure for an active GPU node	Intermittent scrape or maintenance mismatch	Coverage and scrape duration trend
Expected exporter target absent from inventory check	Sustained absence for an active GPU node	Maintenance or inventory mismatch	Expected versus discovered target coverage
DCGM health status	Vendor result and active workload impact	Degraded subsystem without immediate impact	Watch coverage and initial warm-up state
XID event-like code	Code-specific runbook plus observed service impact	New code requiring review without impact	Historical code distribution, no event-rate fiction
ECC or remap state	Confirmed vendor condition affecting service	Persistent pending state or rising cumulative errors	Baseline and device inventory
PCIe or NVLink trend	Only when accompanied by confirmed service degradation	Sustained new errors above local baseline	Link activity and throughput by workload
Temperature, power, utilization	Only when paired with device limit or application impact	Persistent, unexplained capacity or cooling trend	Normal operating distribution (docs.datadoghq.com)

The policy is deliberately asymmetric. A **missing telemetry** alert protects the ability to detect later conditions, while a high utilization gauge usually describes work being done. Conversely, low utilization can be a scheduling, batching, or demand question, so it belongs in a capacity review until tied to a service objective. IBM's published alert examples use configured thresholds for memory, temperature, and underutilization, and Datadog asks operators to check whether default values suit their needs. (cloud.ibm.com) (docs.datadoghq.com)

Kubernetes, MIG, and Dashboard Design

Multi-Instance GPU (MIG) changes the unit of attribution. NVIDIA's default selector covers full GPUs without MIG and GPU instances when MIG is enabled; instance samples add `GPU_I_PROFILE` and `GPU_I_ID`. (docs.nvidia.com) Hardware UUID and instance identity should be kept in the diagnostic view, while pod, namespace, and container labels explain current ownership. Kubernetes' pod-resources API is the source of the device-to-container mapping. (kubernetes.io) A GPU with an unmapped pod label is therefore a different condition from a missing physical GPU metric.

For Kubernetes, verify the kubelet socket mount and access first. Kubernetes recommends mounting the **pod-resources directory** rather than only the socket file so clients can reconnect after kubelet restarts. (kubernetes.io) Compare exporter DaemonSet readiness with Prometheus targets and endpoint output; then compare a GPU UUID against the scheduled pod. GKE's managed DCGM path attaches GPU and Kubernetes labels, while AWS demonstrates joining exporter data with EKS metadata. (docs.cloud.google.com) (aws.amazon.com) GKE cautions against running its managed and a self-managed DCGM collector together because duplicate metrics can result. (docs.cloud.google.com) These examples show attribution methods, not a guarantee that a particular field is available on every GPU.

Cardinality deserves a budget. Every unique label set creates another Prometheus series; adding pod UID or arbitrary pod labels multiplies history as workloads churn. (prometheus.io) The NVIDIA chart's empty pod-label allowlist includes all labels if that enrichment is enabled, while GPU Operator offers an allowlist regex. (github.com) Kubernetes warns about unbounded dimensions, and OpenTelemetry marks potentially high-cardinality metric attributes as opt-in. (kubernetes.io) (opentelemetry.io) Keep `UUID`, node, model, and the minimum workload labels needed for routing. Put rich incident context in a ticket or log link instead of a metric label. Prometheus Operator exposes both metric relabeling and per-scrape sample limits, but an exceeded Prometheus sample limit can fail the entire scrape; limits require a measured baseline. (prometheus-operator.dev) (prometheus-operator.dev) (prometheus.io) A pilot should compare series counts before and after any new workload label, consistent with OpenTelemetry's opt-in treatment of high-cardinality dimensions. (opentelemetry.io)

A useful Grafana DCGM exporter dashboard has four rows: **coverage and freshness**, **health and error change**, **thermals and power**, and **workload saturation**. Variables can select cluster, node, namespace, and GPU for exploration. (grafana.com) IBM's dashboard scopes by cluster, namespace, workload, and device; Azure documents a managed Grafana DCGM exporter dashboard; Oracle documents importing a prebuilt NVIDIA dashboard. (cloud.ibm.com) (learn.microsoft.com) (docs.oracle.com) New Relic also documents a

prebuilt DCGM dashboard. (docs.newrelic.com) An imported dashboard is a starting visualization, not evidence that every panel has a live field. Use the exporter endpoint and query inspector to find missing or differently typed metrics before relying on a panel.

Triage Runbook and Validation

A DCGM exporter incident should preserve **what changed, when, on which entity, and under which workload**. Start with low-impact reads. Active DCGM diagnostics can use substantial GPU, CPU, memory, power, and fabric resources, so schedule them with workload owners rather than running them reflexively on a busy production node. (docs.nvidia.com) Profiling fields can require GPU support and `SYS_ADMIN` capability, so collection gaps may be configuration or permission issues.

1. **Identify the target.** Record alert name, firing time, Prometheus labels, GPU UUID, node, model, MIG instance, and mapped pod. Preserve the original expression and a snapshot of recent samples. (prometheus.io)
2. **Check collection.** Compare DaemonSet or service state, `up`, and the local `http://localhost:9400/metrics` response. NVIDIA expects `DCGM_` names in a successful response.
3. **Check field availability.** Read the live `# TYPE` line and compare the collector file with the device's field support. A selected metric may be absent at runtime; `customMetrics` in the Helm chart replaces the shipped set rather than extending it.
4. **Confirm the signal.** For a counter, inspect `increase()` and resets over a window that spans several valid samples. For a gauge, compare recent history with the node's normal workload phase. For XID, inspect driver logs and code context rather than applying a counter function.
5. **Correlate impact.** Compare application latency, token throughput, job progress, scheduler placement, and host power or cooling telemetry before changing service state. AWS's EKS observability guidance explicitly combines GPU, serving, and application layers.
6. **Choose the least disruptive next step.** Collect `nvidia-smi -q`, exporter and host-engine logs, relevant host journal entries, and scheduler events. Escalate to active diagnostics only with a safe maintenance window.

Validation before paging should include a safe synthetic path: stop a test exporter while retaining its scrape target to prove failed-scrape routing; separately remove a test target from discovery to prove missing-target routing; temporarily remove a noncritical collector field in a test environment to prove absent-series handling; inject a synthetic Prometheus recording-rule series to exercise ticket routing; and run rule syntax and unit tests. Prometheus documents rule-file syntax checking with `promtool`, and Grafana separates No Data from evaluation error. (prometheus.io) (grafana.com) Never simulate a hardware error by stressing a production GPU solely to prove an alert. Check that silence and maintenance selectors are limited to the intended node and period.

Dashboard and Alert Coverage Checklist

The checklist below turns the architecture into an acceptance record for each fleet class. It should be completed against the **actual exporter endpoint**, not merely against a collector file. NVIDIA cautions that selecting a field does not guarantee emission, and Prometheus exposes the sample count remaining after relabeling for a real scrape. (prometheus.io) Review the checklist after a driver, exporter, GPU model, or MIG profile change.

- **Inventory:** list every production GPU node, its owner, model, and expected exporter target.
- **Placement:** verify one exporter process or pod on every selected node, including tainted Kubernetes nodes.
- **Reachability:** check `up` and the endpoint response independently; record planned maintenance exclusions.
- **Field presence:** compare each required `DCGM_` family with the live endpoint and document unsupported fields.
- **Type contract:** record the observed `# TYPE` for each counter, gauge, and event-like value.
- **Cadence:** record the DCGM watch interval, Prometheus scrape interval, and rule evaluation interval.
- **Hardware identity:** preserve node and UUID in alerts even when a pod label is absent.
- **MIG identity:** confirm the selected entity scope and instance labels on a representative partition.
- **Pod mapping:** check the kubelet pod-resources mount and one known GPU-to-pod assignment.
- **Cardinality:** measure series count before and after enabling pod UID or label enrichment. (prometheus-operator.dev)
- **Coverage panel:** show missing targets and absent required families, not only nonzero GPU values.
- **Trend panels:** separate temperature, power, framebuffer, error changes, and workload activity.
- **No Data policy:** decide explicitly whether a missing evaluation result pages or creates a ticket. (grafana.com)
- **Routing:** attach owner, hardware identity, severity rationale, and a runbook URL to each alert.
- **Synthetic check:** prove safe target-loss, field-loss, and notification paths in a test environment.
- **Review:** capture false positives and update local windows after a workload or cooling change.

Prometheus Operator supports per-scrape sample limits, but a limit should protect ingestion only after the normal sample count is measured. (prometheus-operator.dev) Grafana dashboard variables help operators explore an incident, while the alert expression itself must use explicit matchers. (grafana.com) (grafana.com)

Data Analysis and Evidence

The useful numbers in this topic describe **measurement mechanics**, not a universal fault threshold. NVIDIA's documented default watch interval is **30,000 milliseconds**; Prometheus has a separately configured scrape interval. Some DCGM health watches need approximately **60 seconds** of samples before a first check is meaningful. (docs.nvidia.com) (docs.nvidia.com) A rule that evaluates a 10-second transient against a 30-second watch could repeatedly examine the same retained value. This is an engineering deduction from the two cadences, not a claim that all deployments sample every 30 seconds.

Table 3 is a **worked planning example**, not measured fleet data. Assume **64 physical GPUs**, **20 enabled field families** per GPU, **30-second** scrapes, and no extra per-link or per-MIG series. The arithmetic is transparent so an operator can replace every assumption with endpoint measurements. Prometheus counts each distinct label set as a separate series and exposes scraped sample counts for checking the model. (prometheus.io) (prometheus.io)

Planning quantity	Illustrative calculation	Operational use
Base GPU series	**64 × 20 = 1,280** series	Compare with actual endpoint family counts and scrape samples.
Samples per day at 30-second scrape	**1,280 × 2,880 = 3,686,400** samples	Estimate ingestion before compression and metadata overhead.
One extra stable two-value label	Up to **2,560** series if every original series appears in both label states	Avoid multiplying labels without a query or routing need.
Eight MIG instances per physical GPU, if every family is instance-capable	Upper-bound illustration **64 × 8 × 20 = 10,240** instance series	Verify actual field support; never use this as a universal MIG multiplier.

The table shows why an apparently small metric-label change can have a larger storage effect than a new dashboard. The actual count differs with field support, entity scope, cardinality from pod churn, and exporter configuration. GKE explicitly notes that DCGM metrics consume time-series ingestion quota, which makes measuring actual series prudent even when a managed collection path hides some setup work. (docs.cloud.google.com) Prometheus Operator's sample limit can cap a scrape, but exceeding the underlying Prometheus limit fails that scrape; a cap should be paired with coverage alerts and a measured safety margin. (prometheus-operator.dev) (prometheus.io) Metric relabeling occurs before ingestion, but removing a label or metric for storage must not remove the evidence required by an alert. (prometheus-operator.dev)

The quantitative catalog also clarifies units. NVIDIA identifies GPU temperature in **degrees Celsius**, board power in **watts**, framebuffer use in **megabytes**, and total energy in **millijoules since driver reload**. (docs.cloud.google.com) These are not interchangeable. Power is a present draw; energy is an accumulating integral. A utilization percentage is a workload activity reading, not a fault probability. Red Hat's dashboard treats graphics-engine activity and power as separate panels, reinforcing that one gauge cannot substitute for another. (docs.redhat.com)

Implications and Future Directions

The first implication is **version control for observability**. Keep the collector CSV, exporter image, DCGM version, dashboard revision, and alert rules in a change record. NVIDIA recommends pairing exporter and DCGM releases, and a changed collector can alter which families exist. A new GPU generation or driver may change field support, so a successful deployment means endpoint, target, and rule checks on the actual hardware. Azure's documented scrape profile is configured separately from the dashboard, illustrating why these are distinct checks. (learn.microsoft.com) No dashboard import should be treated as proof of collection completeness.

Second, store a **field contract** per fleet class: source field identifier, unit, declared Prometheus type, reset behavior, entity scope, expected labels, and operational owner. This makes counter rules reviewable and allows the platform team to distinguish an unavailable metric from a zero value. The contract should record both the vendor definition and the observed `# TYPE` line because the exporter does not semantically validate CSV types. As MIG use grows, preserving instance identity and separating physical-device health from workload attribution becomes more valuable.

Third, keep infrastructure and application signals together. A GPU utilization panel can help identify when capacity is busy or idle, but serving latency, queue depth, batch behavior, or training progress determines whether intervention is useful. Azure's managed path, GKE's managed DCGM package, and IBM's GPU dashboard demonstrate different collection and viewing arrangements; each still requires local rules for ownership and escalation. (learn.microsoft.com) (docs.cloud.google.com) (cloud.ibm.com) New Relic documents remote write ingestion and a prebuilt DCGM dashboard, while Sysdig ties GPU cost allocation to Operator and exporter usage data. (docs.newrelic.com) (docs.sysdig.com) Sysdig's documented cost-advisor path explicitly requires the GPU Operator and exporter to provide usage data. (docs.sysdig.com) Those integrations broaden destinations, but they do not change the underlying field semantics.

Finally, **alert coverage should be tested as a service**. Track missing exporter targets, absent critical field families, evaluation errors, notification delivery, and runbook ownership. Grafana's No Data state and Prometheus's `up` and absent-series functions separate several failure modes that otherwise look like one blank panel. (grafana.com) (prometheus.io) Revisit local thresholds after workload changes, firmware updates, new MIG profiles, or cooling changes. The report's placeholders should remain placeholders until each fleet can show a baseline, false-positive review, and documented action.

Frequently Asked Questions (FAQs)

Which DCGM exporter Prometheus metrics should be collected first?

Begin with exporter availability, GPU identity, temperature, board power, framebuffer use, a workload-activity gauge, and a small set of health or error signals supported by the deployed GPU. Then add PCIe, NVLink, and profiling fields only where they answer an operator question. The shipped default CSV and actual `/metrics` output are the authoritative inventory for that installation, because a configured field is not guaranteed to appear.

Should GPU temperature or utilization page an SRE?

Usually only after a **locally calibrated** threshold is paired with a device-specific condition or application impact. Temperature and utilization are gauges; they can change with ordinary workload and cooling behavior. NVIDIA's thermal margin is closer to a device-specific slowdown reference, and Datadog explicitly recommends checking whether default values suit the environment. (docs.datadoghq.com) Red Hat labels one displayed temperature maximum in its dashboard as empirical, which is another reason not to copy it into a universal page rule. (docs.redhat.com)

Can a Grafana dashboard be used as an alert rule?

The panels and alert rules may use the same Prometheus source, but Grafana alert queries cannot use dashboard template variables. Translate the panel into an explicit expression, test missing data behavior, and attach a runbook. (grafana.com) (grafana.com) Oracle's documented dashboard import and New Relic's prebuilt template are visualization examples, not ready-made local paging policies. (docs.oracle.com) (docs.newrelic.com)

How should Kubernetes and MIG labels be handled?

Retain stable node and GPU identity, add MIG instance identity where present, and use pod, namespace, and container labels for current attribution. Verify the kubelet pod-resources socket and avoid unbounded pod labels. NVIDIA adds `GPU_I_PROFILE` and `GPU_I_ID` for MIG instance samples and offers a pod-label allowlist in GPU Operator.

Conclusion

DCGM exporter GPU monitoring works when a field can be traced from its DCGM definition through a live endpoint to a specific operator action. Start with node-level coverage, confirm which families and labels actually appear, and record source cadence separately from Prometheus scrape cadence. Use counters for changes over time, gauges for present state, and event-like XID values with code-aware context.

The proposed field catalog and routing matrix put missing data and corroborated health conditions closest to paging, sustained error trends in ticketing, and ordinary utilization, power, and temperature patterns on dashboards. The classification is a starting policy, not a vendor guarantee. A defensible production rule needs a hardware-specific baseline, an evaluation window that contains enough fresh samples, an owner, and a runbook.

For Kubernetes and MIG fleets, keep hardware identity distinct from workload labels and budget for series growth. Validate the complete path with safe missing-target and missing-field tests before enabling notifications. When a signal fires, preserve raw samples, `# TYPE`, GPU identity, workload context, and relevant logs before considering active diagnostics. This sequence gives operators a repeatable explanation for why a GPU alert paged, ticketed, or stayed on the dashboard. It leaves a reviewable record for refining routing when hardware, workload mix, or ownership changes.

Source links

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

1. https://docs.nvidia.com/datacenter/dcgm/latest/installation/install-dcgm-exporter.html
2. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
3. https://prometheus.io/docs/concepts/metric_types/
4. https://docs.cloud.google.com/kubernetes-engine/docs/how-to/dcgm-metrics
5. https://docs.nvidia.com/datacenter/dcgm/latest/learn/modules/health-monitoring.html
6. https://docs.datadoghq.com/integrations/dcgm/

7. https://cloud.ibm.com/docs/monitoring?topic=monitoring-nvidia-gpu-monitoring
8. https://prometheus.io/docs/concepts/jobs_instances/
9. https://prometheus.io/docs/prometheus/latest/querying/functions/
10. https://grafana.com/docs/grafana/latest/datasources/prometheus/template-variables/
11. https://grafana.com/docs/grafana/latest/datasources/prometheus/alerting/
12. https://aws.amazon.com/blogs/machine-learning/enable-pod-based-gpu-metrics-in-amazon-cloudwatch/
13. https://gpusmith.com/
14. https://docs.oracle.com/en/learn/oci-supercluster-gpu/
15. https://learn.microsoft.com/en-us/azure/aks/monitor-gpu-metrics
16. https://github.com/NVIDIA/dcgm-exporter/blob/main/deployment/README.md
17. https://grafana.com/docs/grafana/latest/alerting/fundamentals/alert-rule-evaluation/nodata-and-error-states/
18. https://prometheus.io/docs/alerting/latest/alertmanager/
19. https://docs.nvidia.com/datacenter/dcgm/latest/dcgm-api/dcgm-api-field-ids.html
20. https://docs.nvidia.com/datacenter/dcgm/latest/reference/dcgm-exporter-metrics.html
21. https://docs.redhat.com/en/documentation/openshift_container_platform/4.11/html/monitoring/nvidia-gpu-admin-dashboard
22. https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/
23. https://prometheus.io/docs/practices/the_zen/
24. https://github.com/NVIDIA/dcgm-exporter/blob/main/deployment/values.yaml
25. https://kubernetes.io/docs/concepts/cluster-administration/system-metrics/
26. https://opentelemetry.io/docs/specs/semconv/general/attribute-requirement-level/
27. https://prometheus-operator.dev/docs/api-reference/api/
28. https://prometheus.io/docs/prometheus/latest/configuration/configuration/
29. https://docs.newrelic.com/docs/infrastructure/host-integrations/host-integrations-list/nvidia-dcgm-integration/
30. https://docs.nvidia.com/datacenter/dcgm/latest/reference/command-line-reference/dcgmi/dcgmi-diag.html
31. https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/
32. https://gpusmith.com/articles/en/good-gpu-utilization-percentage
33. https://docs.sysdig.com/en/sysdig-monitor/cost-advisor/

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.