KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Inspektor Gadget

GPU Observability with eBPF

From Kernel Hooks to Kubernetes Metrics



Speakers

KubeCon CloudNativeCon North America 2026



Qasim Sarfraz

Senior Software Engineer @ Microsoft

@mqasimsarfraz



Michael Friese

Senior Software Engineer @ Microsoft

@flyth

Unmasking GPU Bottlenecks



Capacity and cost

Can we answer GPU capacity questions from the telemetry we collect?



Suspected memory leaks

Can we spot memory leaks before they affect workloads?



Better service throughput

Can we use profiles to fine-tune our AI/ML workloads?

See GPU workloads without changing the application

Attach on the host CPU, at the boundary between the application and CUDA.

01 / AT THE SOURCE

CUDA library calls

Attach uprobes where GPU work is requested.

02 / EXECUTION CONTEXT

Arguments + stacks

Capture call arguments, results, and call stacks.

03 / NO CODE CHANGES

External probes

Instrument without modifying the application.

Application-side evidence complements GPU-reported metrics.

How do we build and deploy that visibility?

Our focus throughout: GPU memory



Observe

Activity Counter

Shows what is allocated



Profile

Allocation call stacks

Code-level drill-down



Correlate

GPU state

Joined to the workload



Deploy

OCI gadget images

Helm install

KubeCon CloudNativeCon North America 2026

Inspektor Gadget

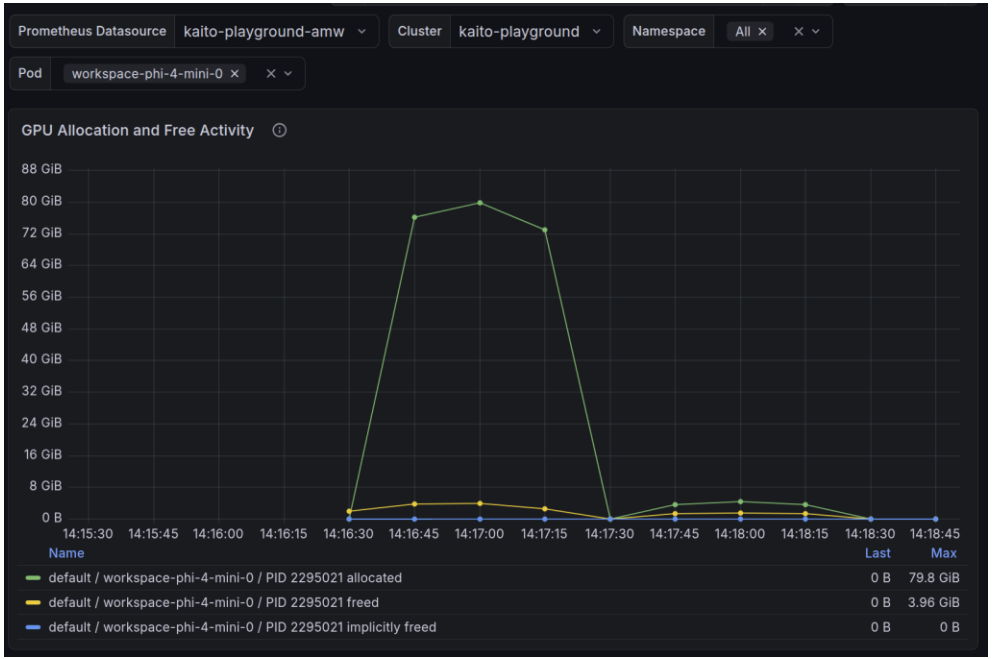
Observe – Activity



First, make allocation and free activity visible

- Observe GPU Memory Activity:
  - Allocation Calls
  - Free Calls
- Record activity by Pod / Namespace.

top_cuda_memory / gadget source



Observed vLLM device allocations

How we capture allocation activity

CUDA LIBRARY HOOKS -> BPF MAP ACCOUNTING

01 / UPROBE

Hook Cuda calls

cuMemAlloc_v2

cuMemFree_v2

Capture call arguments

02 / URETPROBE

Check the result

Observe function return

Confirm success

Capture allocation result

03 / BPF MAPS

Account for activity

Track allocations and frees

Keep per-process totals

GADGET: signal + context

Host CPU uprobes on CUDA userspace libraries — not eBPF inside the GPU.

How we export the telemetry to Prometheus



01 / READ

BPF Map Iterator

GADGET_MAPITER

Periodic BPF map drain

Data to userspace



02 / ENRICH

Workload labels

Process -> container

Namespace + pod

Selected metric keys



03 / ACCUMULATE

A counter

otel-metrics

Counter.Add(delta)

Same attribute set



04 / EXPOSE

A metric

Prometheus /metrics

Workload-labeled

allocation total

Inspektor Gadget Framework: read, enrich, process, export

What allocation accounting cannot tell us



Which code requested the memory?

Counters don't show the application call path



What happened before tracing?

Earlier allocations aren't captured



Is growing memory a leak?

Growth alone can't distinguish leaks from normal allocations

KubeCon CloudNativeCon North America 2026

Inspektor Gadget

Profile – Origin



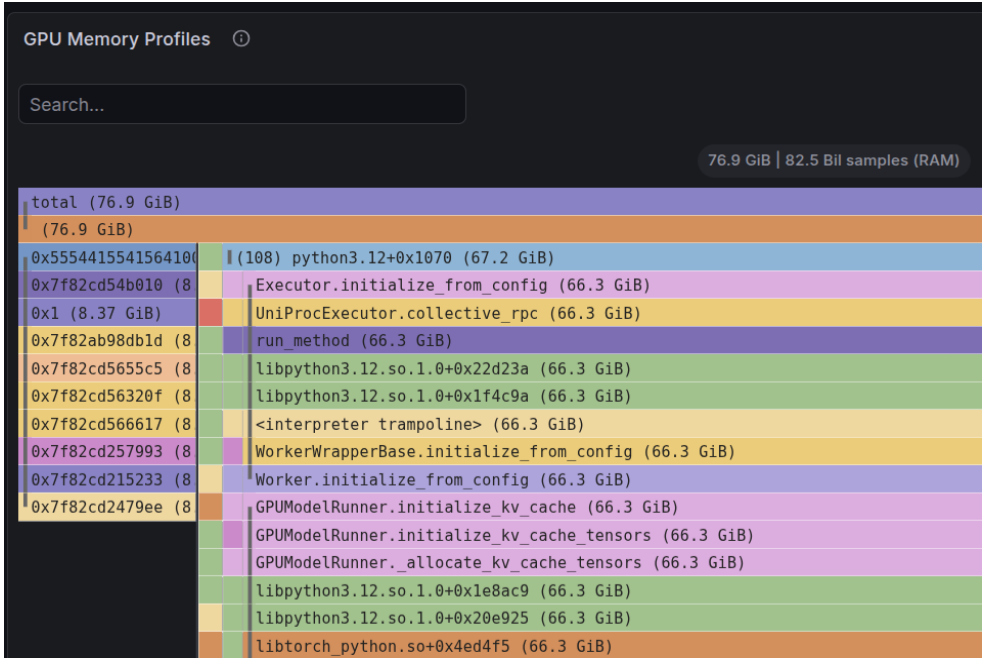
Which application paths requested those bytes?

66.3 GiB

~86% of captured allocation volume

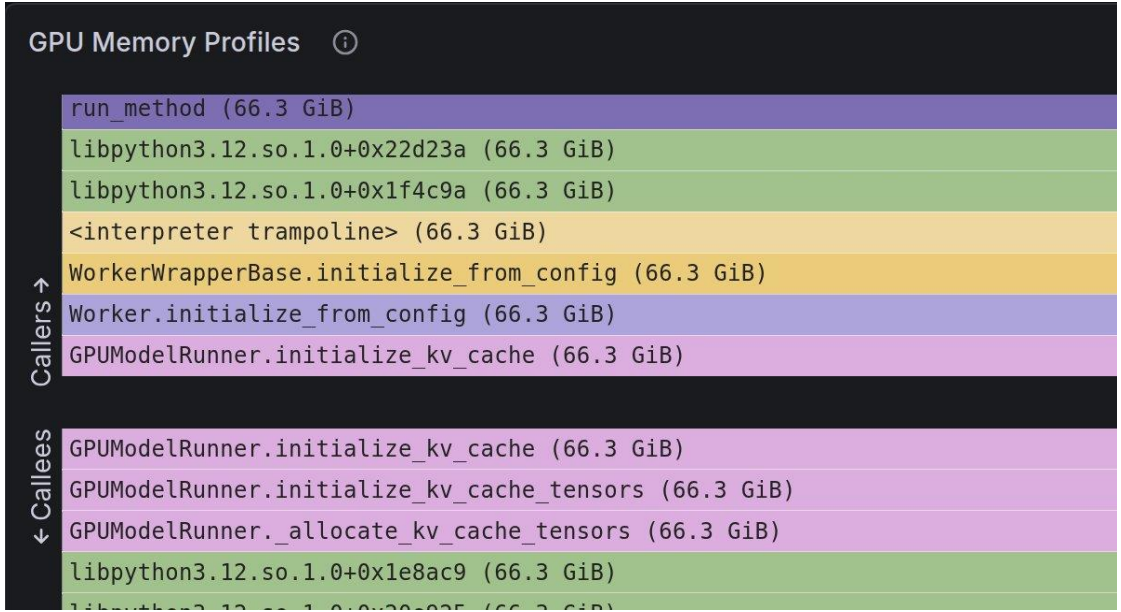
KV-cache initialization is the dominant resolved path in this vLLM capture.

profile_cuda / gadget source



Follow the bytes back to the workload

Observed caller path



How we capture allocation call paths

CUDA LIBRARY HOOKS -> STACK CAPTURE -> BPF MAPS

01 / UPROBE

Hook Cuda calls

cuMemAlloc_v2

Save requested bytes

Match entry to return

02 / URETPROBE

Check the result

Check CUDA success

gadget_get_user_stack

Keep process context

03 / BPF MAPS

Account for activity

Stack ID -> bytes

Add allocated bytes

GADGET_MAPITER

How we get allocation profiles in Pyroscope



01 / INPUT

Stack context

Captured stack

Process context



02 / RESOLVE

Readable frames

ustack + symbolizers

Function names

Pod labels



03 / SERIALIZE

Profile samples

otel-profiles

Stack + byte value

OTLP Profiles



04 / DISPLAY

Pyroscope

Configured route

Pod-filtered view

Width = bytes

Allocation profiles are not live memory

Allocate 1 GiB > free it > allocate 1 GiB

Profile: 2 GiB allocated | Live: 1 GiB

Some calls paths may be incomplete

First runs may be missing symbols

Allocated bytes do not show GPU activity

Large allocations don't tell how busy the GPU is.

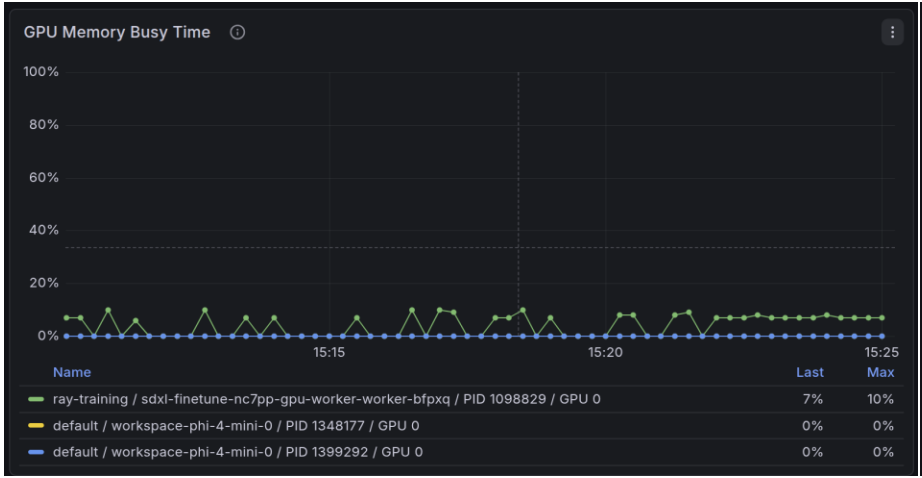
KubeCon CloudNativeCon North America 2026

Inspektor Gadget

Correlate – State



Memory held is not the same as GPU activity





Inference device

76.4 GiB used; reported GPU Memory Activity -> 0%

Two sources meet at the host PID



Put the three views together

DEMO

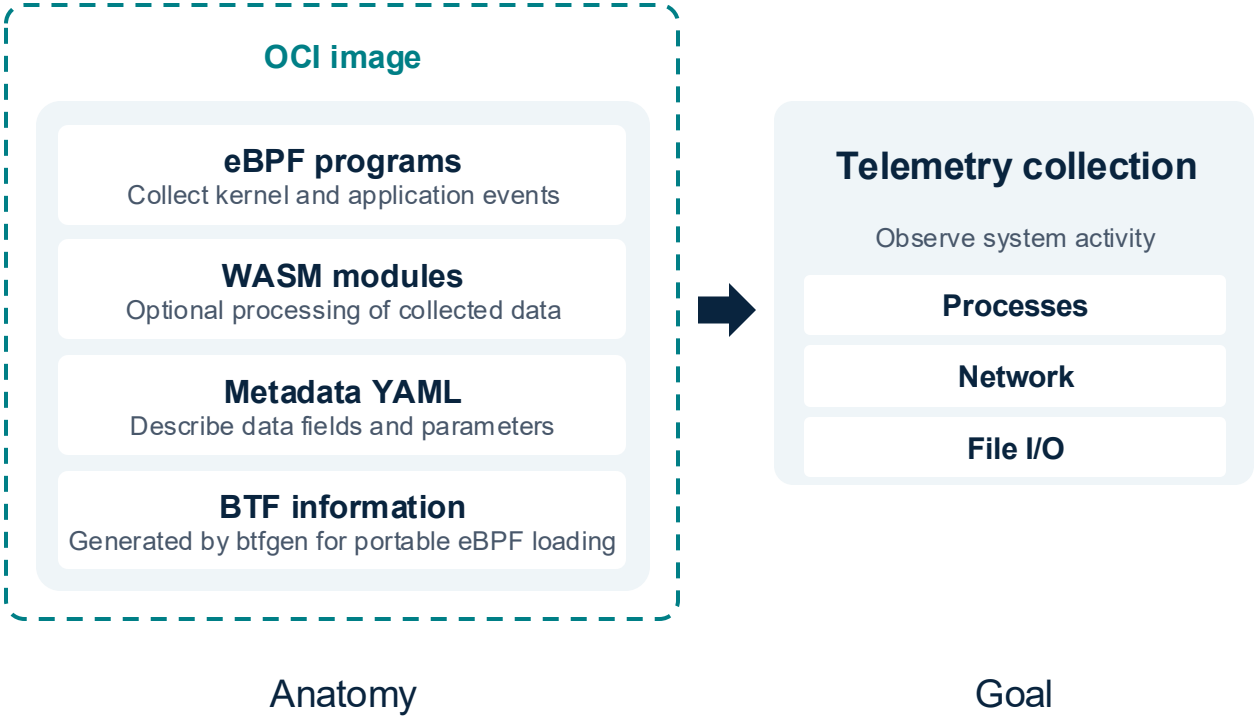
KubeCon | CloudNativeCon
North America 2026

Inspektor Gadget

Deploy



Package your eBPF program as an OCI image



Official Gadgets for GPU Observability



01 / ACTIVITY

top_cuda_memory

CUDA uprobes

Enriched counters



02 / ORIGINS

profile_cuda

Stacks + symbolization

Allocation profiles



03 / DEVICE STATE

gpu_top

NVML bridge

Device gauges



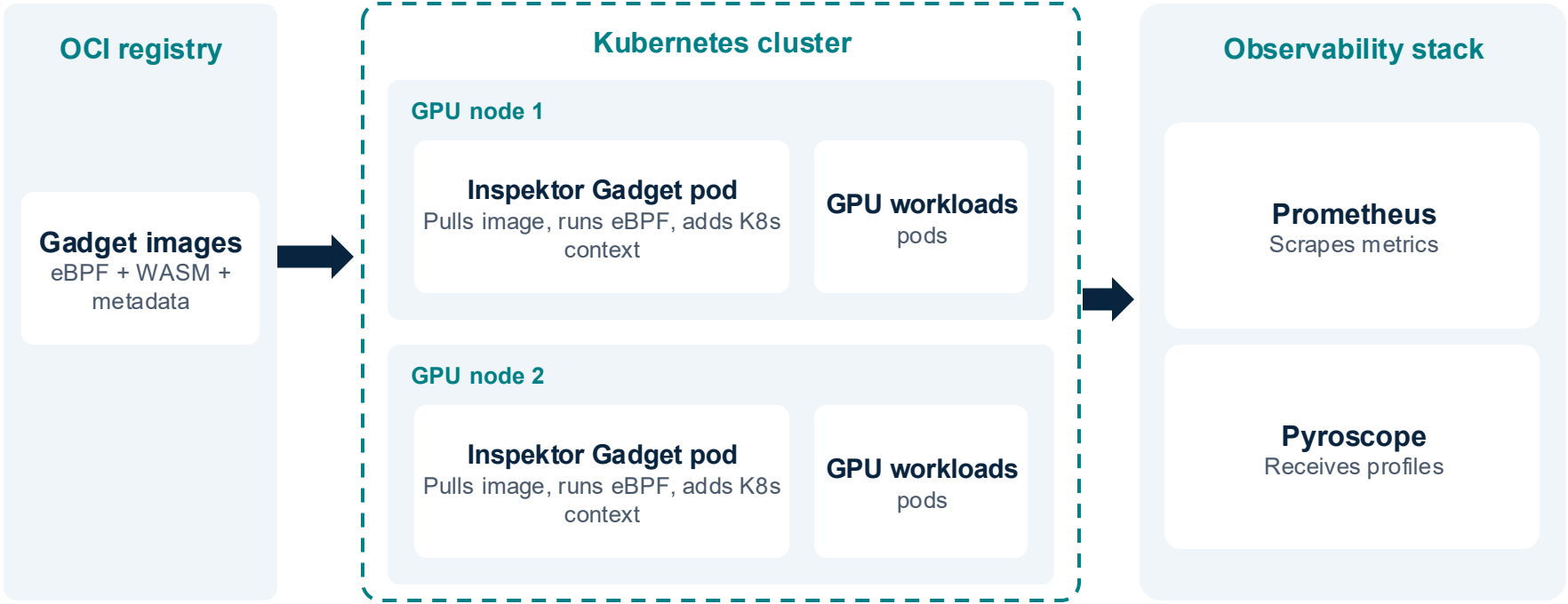
04 / POD STATE

gpu_top_per_pid

Task join

Pod gauges

One framework orchestrates gadgets on every node



Deploy with Helm

Step-by-step instructions
https://inspektor-gadget.io/use-cases/gpu-observability

DAEMONSET

Inspektor Gadget

Runs as a Kubernetes DaemonSet: one pod on every node.

GADGETS

Gadgets included

Gadget images are deployed together with the DaemonSet.

CONFIGURED

Prometheus + Pyroscope

Metrics and profiles export are set up through the same chart.

Reach out to us

SLACK

Kubernetes Slack

#inspektor-gadget

Ask questions, share feedback and talk to the maintainers.

WEBSITE

Website

inspektor-gadget.io

Docs, gadget catalog and getting started guides.

GITHUB

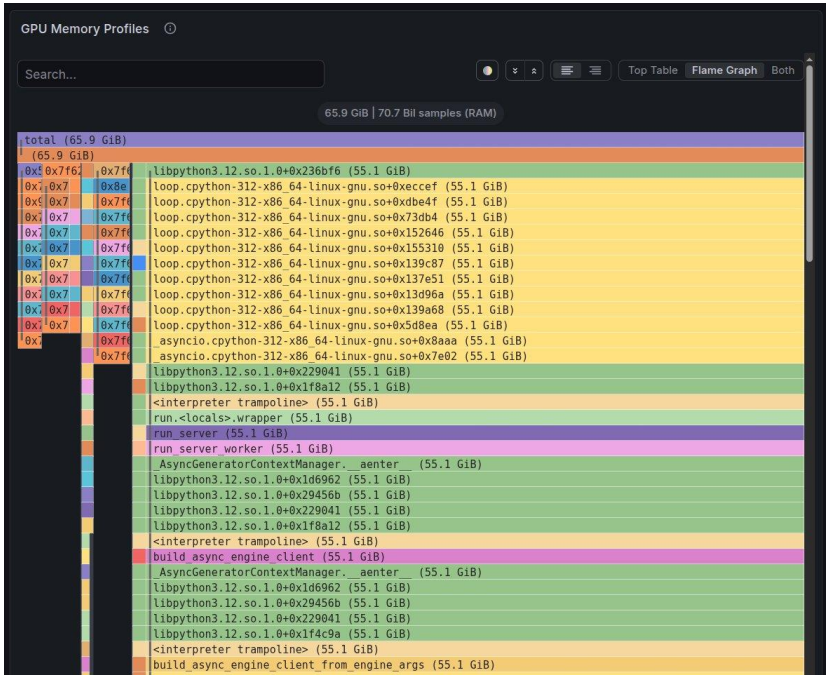
GitHub repository

inspektor-gadget/ inspektor-gadget

github.com/inspektor-gadget/inspektor-gadget

AKS GPU Profiling

KubeCon CloudNativeCon North America 2026



Current Capabilities

- Identify GPU memory allocation hotspots and their application call paths.
- Visualize GPU memory allocations with function-level flame graphs.
- Profile workloads without application code changes or sidecars.
- Optimize memory allocation and validate improvements by re-profiling.

To learn more, visit: **aka.ms/aks/gpuprofiling**

Inspektor Gadget Contribfest

KubeCon CloudNativeCon North America 2026

Wednesday 4:10 PM · 1h 20min · Salt Palace | Level 2 | 255 A

Contribfest: Zero to Published Gadget - eBPF for Everyone with Inspektor Gadget



Qasim Sarfraz
Software Engineer at Microsoft



Michael Friese
Senior Software Engineer at Microsoft

eBPF gives deep, low-overhead visibility into Kubernetes and Linux, but writing, packaging, and sharing eBPF tools is still out of reach for most contributors. Inspektor Gadget, a CNCF project, changes that: it packages eBPF programs as portable OCI images called "Gadgets" that anyone can build, share, and run with `kubectl gadget`. Join the maintainers for a hands-on session that takes you from zero to your own published Gadget. We will walk through the Gadget architecture, set up your environment together, and help you write eBPF code that traces real kernel activity, enriched with Kubernetes context. You will build and publish your Gadget so the whole community can use it, then see the payoff: paired with Agent Skills, your Gadget lets an AI agent run `kubectl gadget` to debug a live cluster. Whether you want to write eBPF, work in Go, or improve docs, we will have curated issues and mentors to help. No prior eBPF experience needed.

KubeCon CloudNativeCon North America 2026

Inspektor Gadget

Thank you!

