

# Beyond Software Estimates

Measured Per-Pod Energy Telemetry for Kubernetes with OpenTelemetry

5 October 2026 · South Hall 3B

Presenting: *Jędrzej Wasiukiewicz*  
Presenting: *Markus Lehtonen*  
Co-author: *Christopher Cantalupo*



intel

# Legal Notices & Disclaimers

This document is classified Public.

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex)

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates.

Statements regarding future products, features, timing, roadmaps and open-source upstream availability are forward-looking, reflect current plans, and are subject to change without notice.

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

# How much energy is this pod consuming?

## WHAT MOST TOOLS ANSWER TODAY

Node-level power (RAPL, IPMI, BMC)

Split across pods by CPU utilization

Energy = model(utilization) - **software estimate!**

## WHAT WE ACTUALLY WANT

Higher accuracy energy counters

Attribution that follows dynamic workloads

Low overhead telemetry

Industry standard observability stacks

**The goal isn't a better software model - it's not needing one.**

# Why per-pod attribution is hard

1. Energy counters lack granularity
  - RAPL reports the whole package
  - No per-process breakdown
2. Pods are dynamic process trees
  - Threads and children fork constantly
3. Short-lived processes slip between polls
  - Init containers and cron jobs can finish before polling sees them
4. A utilization share is not an energy share
  - Instruction mix and working conditions change the energy outputs

## TWO THINGS CHANGED

**Linux now exposes per-group energy counters (7.0)**

**Container runtimes expose lifecycle hooks (NRI)**

## Linux Resource Control filesystem — /sys/fs/resctrl

```
/sys/fs/resctrl/  
├── cpus  
├── cpus_list  
├── info ...  
├── mode  
├── mon_data  
│   ├── mon_L3_00 ...  
│   └── mon_PERF_PKG_00 ...  
├── mon_groups  
│   └── mon_group_0  
│       ├── cpus  
│       ├── cpus_list  
│       ├── mon_data ...  
│       └── tasks  
├── schemata  
├── size  
└── tasks
```

**CLOSID**

allocation

**RMID**

counting

mkdir = kernel allocates RMID  
rmdir = kernel frees it

| File        | Type              | Unit | Meaning                                         |
|-------------|-------------------|------|-------------------------------------------------|
| core_energy | cumulative, float | J    | Core energy for this group                      |
| activity    | cumulative, float | nF   | Frequency/temperature-independent work estimate |

- Available on Intel® Xeon 6+, kernel  $\geq 7.0$  (with config)
- info/PERF\_PKG\_MON advertises capability
- Counters accumulated independently per-package
- Scope: core energy only — excludes power used by L3, memory, and I/O

```
# cd /sys/fs/resctrl/mon_groups/9646ae6b-b12d-48de-8054-27d1d4a38fba/mon_data
# grep . $(find mon_PERF_PKG_00 -type f)
mon_PERF_PKG_00/core_energy:3202.729111
mon_PERF_PKG_00/activity:2498.675507
# grep . $(find mon_L3_00 -type f)
mon_L3_00/mbm_local_bytes:16344842240
mon_L3_00/mbm_total_bytes:16344842240
mon_L3_00/llc_occupancy:16343040
```



**"The resource group of a newly spawned task is the same as the group of its parent."**

## **BENEFIT**

**Place the init process in a monitoring group before it forks. Every thread and child inherits the RMID.**

## **TRAP**

**Place it after start and the children already forked are not moved. Polling never quite catches up.**

# The fork race: userspace daemon approach

**runtime:** create init —▶ exec —▶ fork —▶ fork —▶ fork —▶ ...

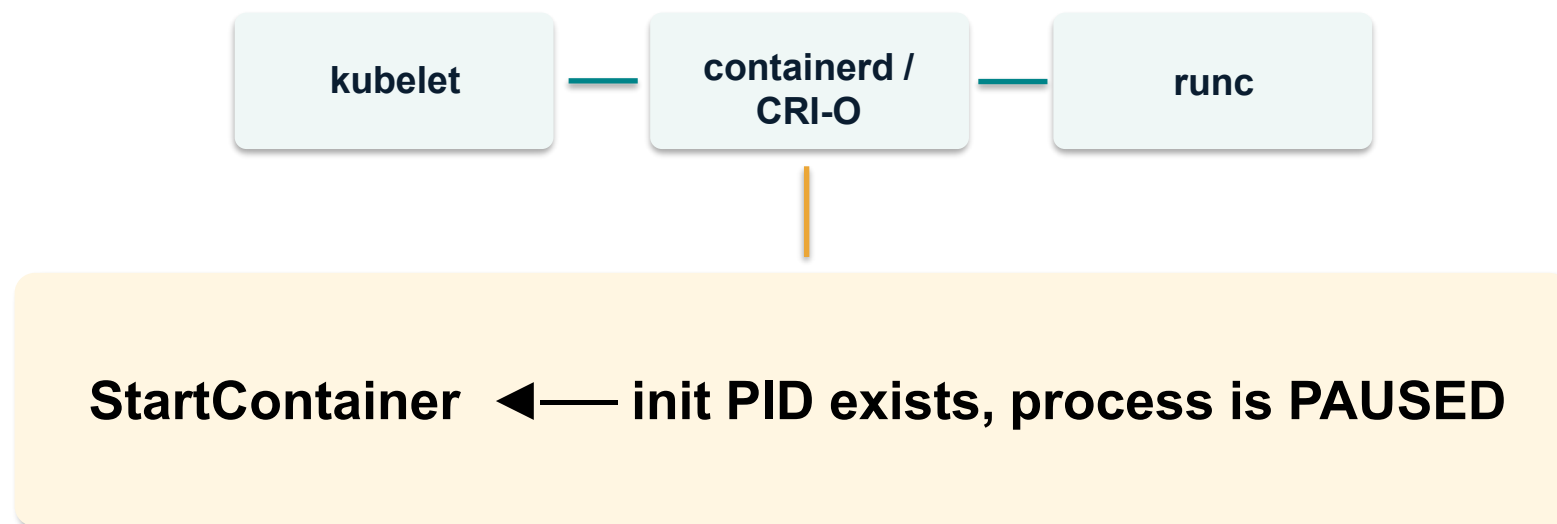
**daemon:** poll API / cgroups —┐ discovers PID

children already forked? ✗ rescan /proc → repeat...

**Attribution gap = everything between exec and first tag.  
Short-lived containers can exit inside the gap.**

**We need a hook, not a poller.**

A standardized synchronous plugin framework in containerd and CRI-O



- Plugins are synchronous with the runtime
- The process has a PID but has executed nothing and forked nothing
- New: nri-resctrl-mon — per-pod resctrl monitoring groups

# Hook mapping: nri-resctrl-mon

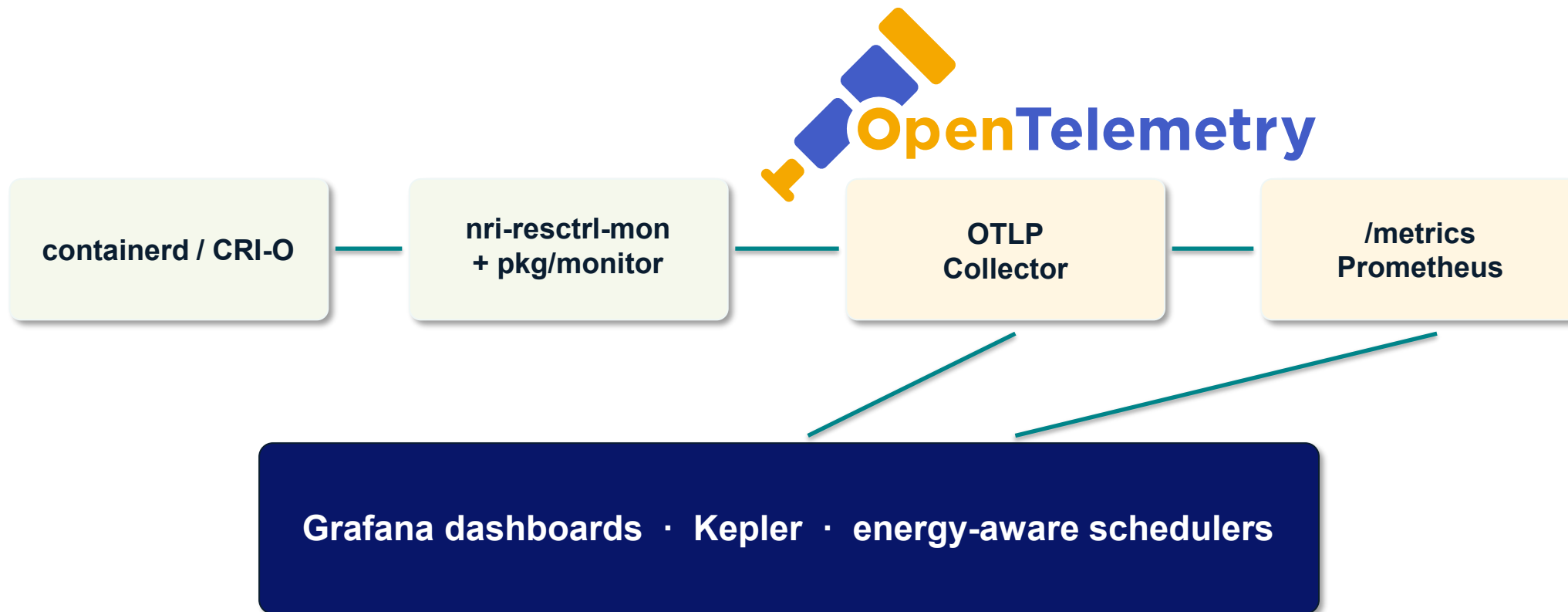
| NRI hook            | What the plugin does                                          | Why here                                                                         |
|---------------------|---------------------------------------------------------------|----------------------------------------------------------------------------------|
| RunPodSandbox       | Record sandbox ID for the pod UID                             | A pod can have several sandboxes; the group belongs to the pod                   |
| PostCreateContainer | <code>mkdir [&lt;class&gt;/]mon_groups/&lt;pod-uid&gt;</code> | Kernel assigns the RMID; PID not yet available                                   |
| StartContainer      | write init PID → tasks                                        | Process exists, is paused, has zero threads → all descendants inherit            |
| PostStartContainer  | Re-write PID (idempotent)                                     | Fallback if runtime did not deliver PID pre-start                                |
| RemovePodSandbox    | <code>rmdir</code> when pod's last sandbox is gone            | Pod UID will never be reused → safe to free RMID; failures retried by reconciler |

**One monitoring group per pod, keyed by pod UID.  
Container restarts keep the pod group and RMID stable.**

# Reusable core: goresctrl pkg/monitor

|           | pkg/rdt (existing)                  | pkg/monitor (new)                        |
|-----------|-------------------------------------|------------------------------------------|
| Purpose   | Allocation: cache / MB partitioning | Monitoring-group lifecycle               |
| Scope     | L3 CMT / MBM                        | All mon_data domains, including PERF_PKG |
| Placement | goresctrl-managed control groups    | Any existing control group               |
| Readings  | L3 integers                         | Typed Reading: kind, unit, float64       |
| Recovery  | n/a                                 | Reconcile(live)                          |

**Design rule: pkg/rdt monitors what it allocates;  
pkg/monitor manages groups for workloads it does not allocate.**



One component creates the groups and reads the counters:  
no second exporter to keep in sync



The plugin exports `k8s_pod_uid` —  
the only stable identity available at StartContainer.

## QUERY-TIME: PROMETHEUS

```
sum by (pod, namespace) (  
  rate(perf_core_energy_joules_total{  
    resctrl_group_source="pod"  
  }[$__rate_interval])  
  * on (k8s_pod_uid) group_left(pod, namespace)  
    label_replace(kube_pod_info, ...))
```

## INGEST-TIME: OTEL COLLECTOR

```
processors:  
  k8sattributes:  
    pod_association:  
      - sources: [{ from: resource_attribute,  
                    name: k8s.pod.uid }]  
    extract:  
      metadata: [k8s.namespace.name,  
                 k8s.pod.name, k8s.node.name]
```

**UIDs are not reused.**  
**Names are joined back from the source of truth.**

**Problem: plugin restarts, node reboots, or transient rmdir failures can leave orphaned groups and ghost series.**

## NRI-Plugin resolution

- Synchronize on connect: EnsureGroup for every live pod and compute the live set.
- Reconcile(live): remove groups under our root whose names are valid pod UIDs but are not live.
- Background reconciler every 30 seconds retries failed removals and reaps missed events.
- Groups with names that are not pod UIDs are left alone.

## REQUIRED PERMISSIONS

- CAP\_SYS\_ADMIN + CAP\_DAC\_OVERRIDE
- hostPID: true
- /sys/fs/resctrl mounted on the node
- Not privileged: true
- No Kubernetes API access or RBAC

## DEFENSIVE MEASURES

- Kernel owns RMID allocation
- Pod UID validated before becoming a path
- Never creates control groups
- Namespace and label filters scope monitoring

## Demo: silicon counter → Grafana panel

```
kubectl apply → mon_groups/<uid> → core_energy_total → Grafana
```

- Same CPU utilization, different power: vector vs scalar workload
- Energy counter starts incrementing on the first scrape
- New monitoring group appears within the lifecycle hook
- Pod deletion removes the group and ends the series cleanly





Search or jump to... ctrl+k

+ ? 📶 👤

Home

Welcome to Grafana

Need help? [Documentation](#) [Tutorials](#) [Community](#) [Public Slack](#)

Basic

The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL

DATA SOURCE AND DASHBOARDS

Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

Remove this panel

COMPLETE

Add your first data source

Learn how in the docs [↗](#)

COM

Cre. das

Lear

Dashboards

Starred dashboards

Recently viewed dashboards

Kubernetes Pod Energy (Intel AET / resctrl-mon) AET ☆

Kubernetes Pod Perf Counters (Intel AET / resctrl-mon) AET ☆

Latest from the blog



Sep 30

File Edit View Run Kernel Tabs Settings Help

devcloud@a4bf018d8112: ~ × +

devcloud@a4bf018d8112: ~\$

```
helm install resctrl-mon \  
oci://ghcr.io/containers/nri-plugins/helm-charts/nri-resctrl-mon \  
--devel --version v0.14-unstable \  
--namespace kube-system \  
--set nri.runtime.patchConfig=true
```

## PREREQUISITES

- Kubernetes  $\geq 1.24$
- containerd  $\geq 1.7$  or CRI-O  $\geq 1.36$
- Intel® Xeon 6+
- Linux  $\geq 7.0$  with config
- resctrl mounted
- kube-state-metrics  
(for the kube\_pod\_info join)

## HELM CHART

- DaemonSet
- Plugin ConfigMap
- Optional Prometheus Operator PodMonitor

## REPO SAMPLES

- OTel Collector agent
- RBAC manifests
- Two Grafana dashboard JSONs
- Source path: deployment/samples/resctrl-mon/

## SAME INTERFACE, MORE HARDWARE

- Architecture-independent fs/resctrl
- Arm MPAM integration
- AMD resctrl monitoring
- New domains appear as mon\_data directories

## VIRTUAL MACHINE SUPPORT

- Solve the fork race for VM creation with libvirt
- libvirt v12.4.0: support for resctrl monitoring
- libvirt v12.8.0: whole process monitoring
- Extensions for Telegraf integration under review



1. Most Kubernetes energy metrics are modeled
  - Linux 7.0 resctrl + Intel AET make per-group measurement possible
2. Attribution is a timing problem
  - NRI StartContainer tags the init PID while it is paused and thread-less
3. Separate monitoring from allocation
  - goresctrl/pkg/monitor provides one lifecycle, typed-counter, and OTel library
4. Use the stack you have
  - Prometheus pull or OTLP push turns energy into a signal for scheduling, tuning, and chargeback

```
helm install ... nri-plugins/nri-resctrl-mon
```

THANK YOU

# Questions?

## Code Links



[containers/nri-plugins](#) — resctrl-mon, Helm chart, dashboards

[intel/goresctrl](#) — pkg/monitor (v0.14.0)

[Linux Documentation/filesystems/resctrl.rst](#) — PERF\_PKG / core\_energy / activity

[Demo on Intel Cloud Services](#) — Scripts and AI assistant

|                       |         |
|-----------------------|---------|
| Jędrzej Wasiukiewicz  | · Intel |
| Markus Lehtonen       | · Intel |
| Christopher Cantalupo | · Intel |



# Intel® AET Documentation



- [Intel® AET overview slides](#) — links below on last slide
- [Data Center Business Strategy: Computex 2026](#)
- [DCG Sustainability and Energy Efficiency 30-3-30](#)
- [Intel® Xeon 6+ Processors Landing Page](#)
- [Intel® Xeon 6+ Intel AET Solution Brief](#)
- [Intel® AET Fact Sheet](#)
- [Intel Community Blog: A New Era of Energy Visibility](#)
- [Intel® AET enablement guide](#)

# Backup Slides

```
mgr, err := monitor.New(monitor.Options{
    ResctrlRoot: "/sys/fs/resctrl",
    KeyValidator: monitor.PodUIDValidator,
})

grp, err := mgr.EnsureGroup(podUID, rdtClass)
err = mgr.AssignPID(podUID, initPID)
rds, err := mgr.ReadCounters(podUID)
err = mgr.Remove(podUID)
err = mgr.Reconcile(liveKeys)
```

| Reading         | Kind       | Unit   |
|-----------------|------------|--------|
| core_energy     | Cumulative | J      |
| activity        | Cumulative | farads |
| llc_occupancy   | Gauge      | By     |
| mbm_local_bytes | Cumulative | By     |

Idempotent EnsureGroup; never creates the control group  
ErrClassMismatch prevents accidental allocation changes  
Pluggable key validation also supports VM UUIDs

```
reg, err := monitor.RegisterOTelInstruments(meter,
    monitor.WithAttributes(func(g *monitor.Group)
[]attribute.KeyValue {
    return []attribute.KeyValue{
        attribute.String("k8s.pod.uid", g.Key()),
        attribute.String("resctrl.group.source", "pod"),
    }
}),
    monitor.WithFilter(perfCounterFilter),
)
defer reg.Unregister()
```

Discovers instruments from mon\_data at registration time

One batch observer reads every tracked group per collection

Each point carries domain.id and domain.name

Works with Prometheus pull, OTLP push, or both

# Prometheus pull vs OTLP push

## PROMETHEUS PULL (DEFAULT)

```
telemetry:  
  prometheus:  
    enabled: true  
    listenAddress: ":9100"
```

Auto-discovered by annotation

Zero new infrastructure

Enrich at query time

## OTLP PUSH (OPTIONAL)

```
otlp:  
  enabled: true  
  endpoint: otel-collector:4317  
  protocol: grpc  
  interval: 15s
```

k8sattributes enriches at ingest

Vendor-neutral fan-out

Both modes can run simultaneously



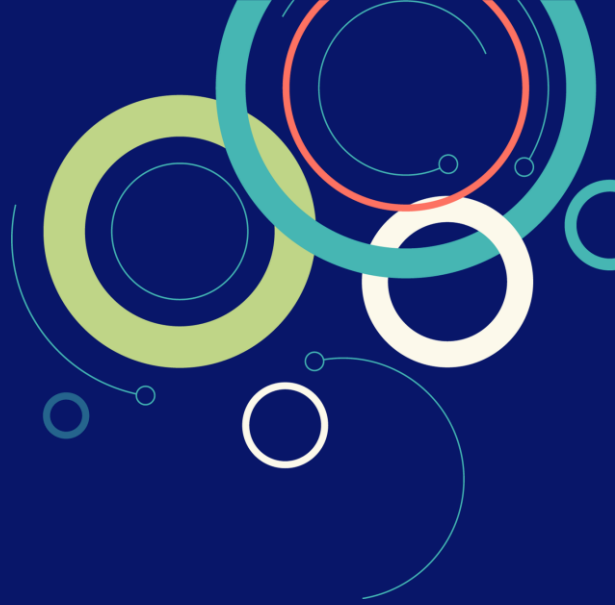
Creates groups under the container's existing RDT class, or at root if none.

Never creates or modifies control groups; allocation policy remains authoritative.

Class mismatch between a sidecar and its pod means PID is not assigned.

resctrl.control\_group on every series makes energy per RDT class available with a sum by query.

```
/sys/fs/resctrl/  
├─ mon_groups/<pod-a-uid>/      # root  
└─ gold/  
    ├─ schemata  
    └─ mon_groups/<pod-b-uid>/  # class gold
```



# intel