Why Are You Still Wasting Whole GPUs?

Hardware-level sharing, scheduled with DRA.

Engin Diri · Principal Solutions Architect, Pulumi

KubeCon + CloudNativeCon North America · Salt Lake City · November 10, 2026





Engin Diri

Principal Solutions Architect at Pulumi

@_ediri engin-diri dirien

Building platform tooling and infrastructure-as-code. Helping teams ship cloud infrastructure faster. With and without agents.

A customer called.





Actually it was more like this:





Customer

Active



Customer 10:00 AM

Hey Bro, how are you?

M

Me 10:01 AM

I am fine and you?



Customer 10:02 AM

Good good! Bro, you rmember tha twe pay you folks a shit load of money

M

Me 10:03 AM

Yes... Why you telling me this?



Customer 10:04 AM

I want you help build us someting, now!

M

Me 10:05 AM

Ok. I schedule a meeting!



Customer 10:06 AM

Nice, talk to you and bring your whole gang. This is big



A European automotive OEM asked us to help build an internal AI platform: training, fine-tuning and serving the models they own.

The pattern is industry-wide. Others in the industry have been shipping similar work:

BMW GROUP

Connected AI platform on Kubernetes (AWS EKS). **230 PB** usable storage, **240M km** of test data simulated for autonomous-driving model training.

Red Hat OpenShift · DXC

VOLKSWAGEN GROUP

Generative-AI pipeline for brand-compliant vehicle assets. SageMaker, `ml.g5.2xlarge` async inference. Live in 2025.

AWS · Amazon Nova Lite for prompt optimization

CONTINENTAL

Distributed AI infrastructure with NVIDIA DGX clusters and IBM Elastic Storage. Training time dropped **from weeks to days**.

Equinix · NVIDIA · IBM

Why run your own models? Revolut did it.



David Abitbol, Revolut · AI Engineer New York
"How Revolut Cut AI Coding Costs 90% With Self-Hosted Open Models"

A tenth of the bill. An open model (GLM 5.3) for 2,000 engineers at about $0.30 per million tokens. Flat cost.

Unlimited tokens changed the work. Lead time down 70%, coding time down 60%.

Regulated, and still fine. An EU bank on a Chinese open model: weights on US GPUs, a human in the loop.

The GPUs are now your only bill. 56× B200. Tuning, not more GPUs, cut time to first token by 88%.

Owning the model is the easy part. Using every GPU well is the hard part.


I’m limited by the technology of my time.

We trialed and evaluated two options.

OPTION 1: THE DEVICE PLUGIN

`nvidia.com/gpu: 1`. A count, not an attribute. The pod gets the whole GPU or nothing.

An 80GB A100 serving a 12GB quantized LLM: **the GPU runs at 30–35% compute utilization.**

OPTION 2: TIME-SLICING

Lie to the scheduler: advertise N replicas of one GPU.

```
# ClusterPolicy → device plugin
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 4
```

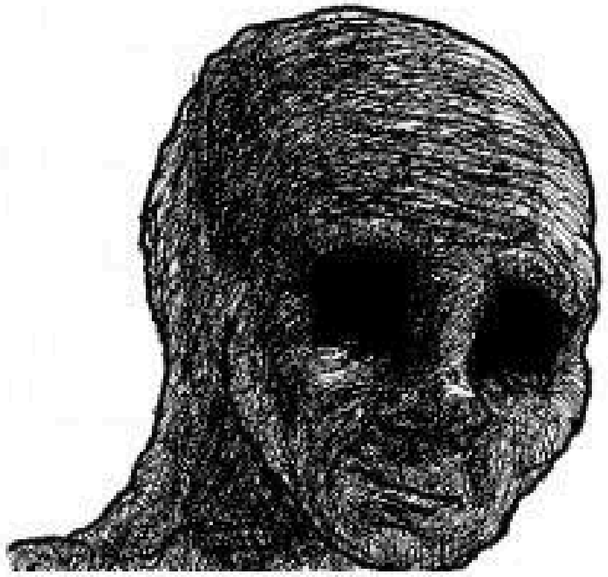
Three things break the moment you turn it on:

- No memory or fault isolation between workloads on the same GPU
- No guaranteed compute share. A noisy neighbor starves the rest
- Per-pod GPU metrics go dark in DCGM-Exporter

Whole GPUs wasted the card. Time-slicing put every neighbor at risk.



2020:
Wow GPU so expensive, I guess I'll wait until price goes down



2026:

imgflip.com

LET'S CHECK GPU PRICES





WTF

imgflip.com



ASUS TUF GAMING RTX5090 OC, Scheda Grafica 32GB GDDR7 NVIDIA GeForce, Nera

512 Bit, PCIe 5.0, HDMI 2.1,...

4,4 ★★★★½ (51)

9.330[71] €

Consegna GRATUITA **17 - 24 nov** sul tuo primo ordine idoneo

Disponibilità: solo 3



Then things changed.

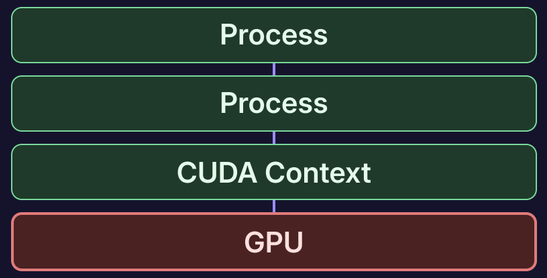


Change one: the GPU itself can be carved.

NVIDIA's Multi-Instance GPU (MIG) splits one physical card into partitions with their own streaming multiprocessors (SMs), memory, L2 cache and memory bandwidth. Hardware-enforced.

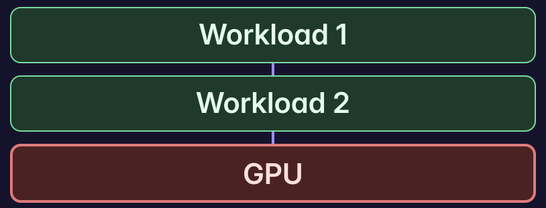
MPS

Multi-Process Service



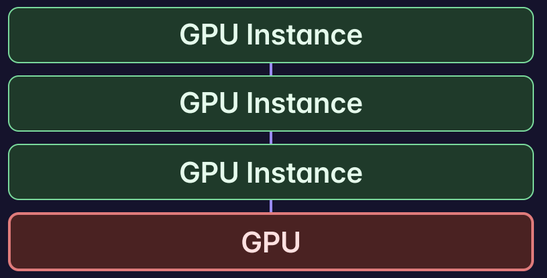
- One shared CUDA context, concurrent kernels
- Shared memory space
- No fault isolation

Time-Slicing



- Pods take turns on the whole GPU
- Context-switch tax, noisy neighbours
- No memory or fault isolation

MIG



- Dedicated SMs, memory and cache
- **Fault isolation between slices**
- Up to 7 slices (A100, H100), 4 on RTX PRO 6000

Change two: the scheduler can now read GPU attributes.

Dynamic Resource Allocation went GA in Kubernetes 1.34. For the first time, the scheduler can read what's *inside* a GPU, not just an integer count of devices.

WHAT "ATTRIBUTES" ACTUALLY MEANS

What the NVIDIA DRA driver publishes per device:

- `productName`: `NVIDIA A100-SXM4-40GB`, `NVIDIA H100`, …
- `architecture`: `Ampere`, `Hopper`, `Blackwell`
- `profile` + `parentUUID` (MIG devices): `3g.20gb`, `1g.5gb`, …
- topology: `resource.kubernetes.io/numaNode`, `pcieRoot`
- capacity, not attributes: `memory`, `multiprocessors`, copy engines

THE PATTERN IS BORROWED

"This enhancement took inspiration from dynamic provisioning for storage volumes."
Kubernetes 1.34 release blog

Storage (since 2017)	DRA (GA 2025)
`StorageClass`	`DeviceClass`
`PersistentVolume`	`ResourceSlice`
`PersistentVolumeClaim`	`ResourceClaim`
`volumeClaimTemplates`	`ResourceClaimTemplate`

If you've ever written a PVC, you already know how to ask for a GPU.



The cluster side.

The DRA driver and your platform team create these. App teams never write them by hand.

DEVICECLASS

An admin-defined filter that workloads reference by name. Same shape as `StorageClass`.

```yaml
apiVersion: resource.k8s.io/v1
kind: DeviceClass
metadata:
  name: large-gpu
spec:
  selectors:
  - cel:
      expression: |
        device.driver == "gpu.nvidia.com" &&
        device.attributes["gpu.nvidia.com"].type == "gpu" &&
        device.capacity["gpu.nvidia.com"].memory
          .compareTo(quantity("40Gi")) >= 0
```

RESOURCESLICE

The driver's auto-published inventory. One slice per node, listing every device with its attributes and capacity.

```yaml
# Published by the DRA driver.
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
spec:
  driver: gpu.nvidia.com
  nodeName: gpu-node-1
  pool:
    name: gpu-node-1
    generation: 1
    resourceSliceCount: 1
  devices:
  - name: gpu-0
    attributes:
      type: { string: "gpu" }
      productName: { string: "NVIDIA A100-PCIE-40GB" }
      architecture: { string: "Ampere" }
    capacity:
      memory: { value: "40Gi" }
```

The workload side.

What app teams actually write: a one-off claim, or a template per workload.

RESOURCECLAIM

A direct request, like a `PersistentVolumeClaim`. Every pod that references it shares the same device.

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
metadata:
  name: shared-gpu
  namespace: ml-team
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        deviceClassName: large-gpu
        count: 1
```

RESOURCECLAIMTEMPLATE

Stamps out a fresh ResourceClaim per pod. Same shape as `volumeClaimTemplates` in a StatefulSet.

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: single-gpu
  namespace: ml-team
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: large-gpu
          count: 1
          selectors:
          - cel:
              expression: |
                device.attributes["gpu.nvidia.com"]
                  .architecture == "Ampere"
```

So we start building it.





What we shipped them, in one picture (simplified).

AWS · EU-CENTRAL-1 · VPC + PRIVATE SUBNETS

EKS 1.37 · gpu-dra-cluster · IRSA · Pod Identity

system pool
m6i.xlarge × 2
CoreDNS · DRA controller · gateway

gpu pool
p4d.24xlarge × N
8× A100 40GB · MIG all-balanced

NVIDIA GPU Operator
MIG Manager · DCGM

NVIDIA DRA Driver
kubernetes-sigs · v0.5.1

Prometheus + Grafana
DCGM MIG dashboard · Loki · Tempo

Karpenter
GPU node autoscaling

cert-manager
TLS for webhooks

Argo CD
GitOps · app sync

External Secrets
Pulumi ESC · AWS SM

KEDA
queue-based scaling

Inference gateway
agentgateway · llm-d EPP

ECR · S3 · state · CloudWatch · Route 53

ResourceClaim ↑ ↓ carves MIG slice

MIG-TEST NAMESPACE · TENANT WORKLOADS

large
3g.20gb
ResNet-18 training

medium
2g.10gb
custom CNN training

serve-a
1g.5gb
vLLM · Qwen3

serve-b
1g.5gb
vLLM · Qwen3

With MIG, every slice is a device.

Same four objects as before. The driver just publishes smaller devices.

RESOURCESLICE: THREE SLICES, ONE GPU

```yaml
# Published by the DRA driver.
kind: ResourceSlice
spec:
  driver: gpu.nvidia.com
  devices:
  - name: gpu-0-mig-2g48gb-5-0
    attributes:
      type:       { string: "mig" }
      profile:    { string: "2g.48gb" }
      parentUUID: { string: "GPU-676c1adb-…" }
  - name: gpu-0-mig-1g24gb-14-6
    attributes:
      type:       { string: "mig" }
      profile:    { string: "1g.24gb" }
      parentUUID: { string: "GPU-676c1adb-…" }
  - name: gpu-0-mig-1g24gb-14-9   # 1g.24gb, same parent
```

RESOURCECLAIM: THREE SLICES, SAME GPU

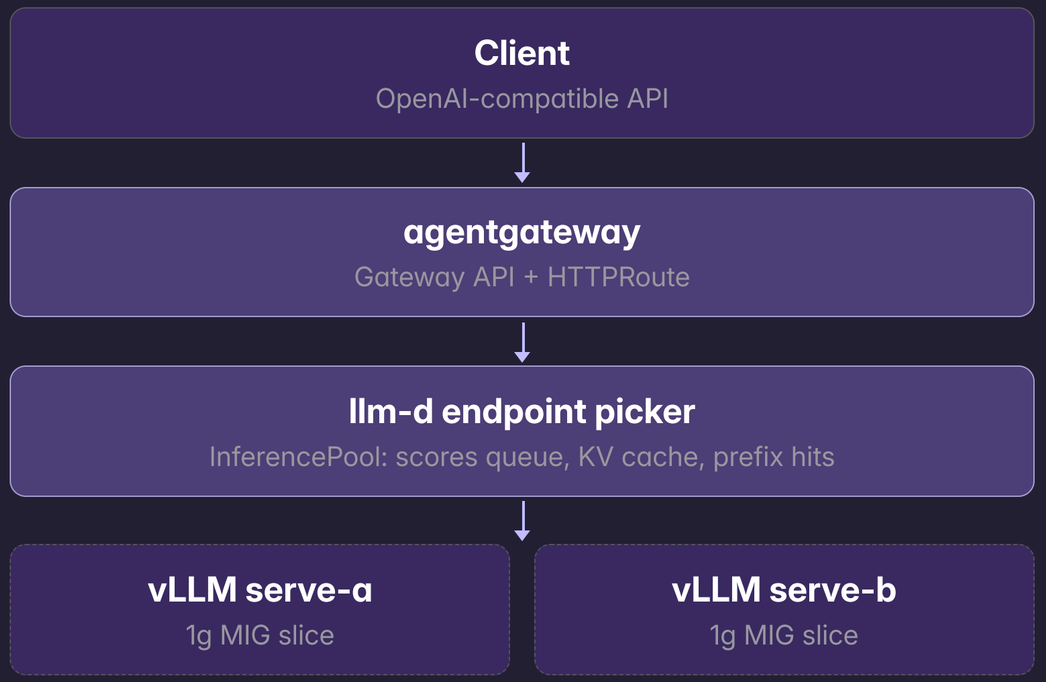
```yaml
kind: ResourceClaim
metadata: { name: shared-mig-claim }
spec:
  devices:
    requests:
    - name: mig-large                # training
      exactly:
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes["gpu.nvidia.com"]
                .profile == "2g.48gb"
    # mig-serve-a, mig-serve-b: same, "1g.24gb"
    constraints:
    - matchAttribute: gpu.nvidia.com/parentUUID
      requests: [mig-large, mig-serve-a, mig-serve-b]
```

Each pod picks its slice: `resources.claims: [{ name: mig, request: mig-serve-a }]`. The scheduler finds three slices that share one `parentUUID`.

Why an inference gateway?



One gateway, many slices.



A Service balances blind.

kube-proxy picks a pod per connection, at random. It can't see which replica is busy. On a 1g slice, one bad pick is a latency spike.

The picker reads the slices.

Per request, llm-d's endpoint picker scores each vLLM replica on queue depth, KV-cache use and prefix-cache hits, then sends it to the best one.

Standard APIs, swappable parts.

Gateway API Inference Extension defines the InferencePool. agentgateway is our gateway today; any conformant one works.

And all done in Pulumi!

❤️ ❤️ ❤️



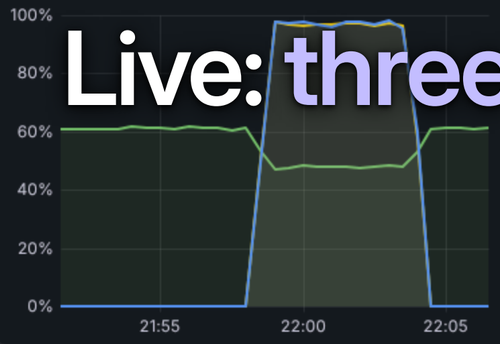
Live: one GPU, three pods.




$ #
0:00 / 1:24

Live: three workloads, climbing.

GPU Utilization



Name	Mean	Last *	Max
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	56.2%	21.2%	61.6%
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	4.7%	0%	97%
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	34.8%	0%	98.0%

GPU Framebuffer Mem Used



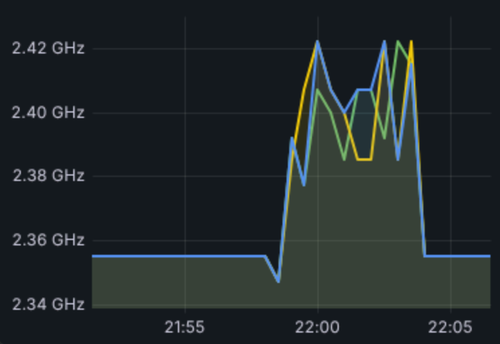
- NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 1 — Mean: 10.4 GB
- NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 5 — Mean: 20.8 GB
- NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 6 — Mean: 20.8 GB

Tensor Core Utilization



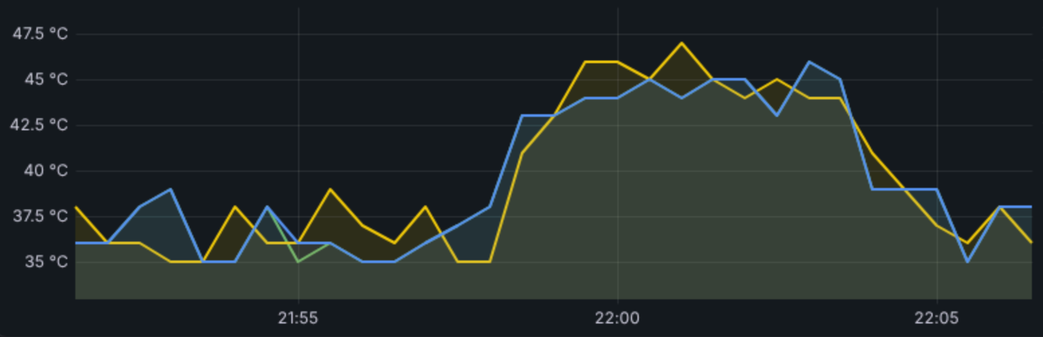
Name	Mean	Last *	Max
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	8.54%	8.64%	8.69%
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	1.89%	0%	5.42%
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0...	2.02%	0%	5.80%

GPU SM Clocks



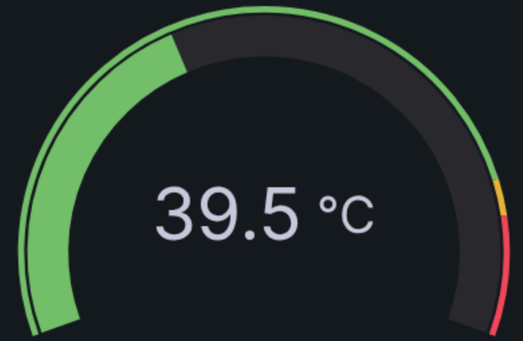
Name	Mean	Last *	Max
NVIDIA RTX PRO 6000 Blackwell Server Edition ...	2.37 GHz	2.36 GHz	2.42 GHz
NVIDIA RTX PRO 6000 Blackwell Server Edition ...	2.37 GHz	2.36 GHz	2.42 GHz
NVIDIA RTX PRO 6000 Blackwell Server Edition ...	2.37 GHz	2.36 GHz	2.42 GHz

GPU Temperature



Name	Mean	Last *	Max
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 1	39.5 °C	38 °C	46 °C
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 5	39.6 °C	36 °C	47 °C
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 6	39.5 °C	38 °C	46 °C

GPU Avg. Temp



GPU Power Usage



Name	Mean	Last *	Max
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 1	233 W	197 W	336 W
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 5	234 W	187 W	324 W
NVIDIA RTX PRO 6000 Blackwell Server Edition GPU ID: 0 Instance ID: 6	235 W	197 W	336 W

GPU Power Total



```
# The GPU node and its MIG layout.
$ kubectl get nodes -l node-role=gpu -L nvidia.com/gpu.machine,nvidia.com/mig.config
NAME                           STATUS   ROLES    AGE   VERSION               GPU.MACHINE   MIG.CONFIG
ip-10-0-107-255.ec2.internal   Ready    <none>   10h   v1.37.1-eks-6e7947a   g7e.4xlarge   all-balanced

# Which MIG profiles does this GPU support? (Instances = free/total)
$ kubectl exec -n gpu-operator ds/nvidia-mig-manager -c nvidia-mig-manager -- \
    nvidia-smi mig -lgip | grep -E 'Name|Free/Total|MIG [0-9]g\.[0-9]+gb '
| GPU   Name               ID    Instances   Memory     P2P    SM    DEC   ENC  |
|                           Free/Total   GiB             CE    JPEG  OFA  |
|   0  MIG 1g.24gb         14     0/4        23.62      No     46     1     1   |
|   0  MIG 2g.48gb          5     0/2        47.38      No     94     2     2   |
|   0  MIG 4g.96gb          0     0/1        95.00      No     188    4     4   |

# What the DRA driver publishes to the scheduler: the slices that exist now.
$ kubectl get resourceslices -o jsonpath='{"DEVICE\t\t\tPROFILE\tPARENT GPU\n"}{range .items[*].spec.devices[*]}...'
DEVICE                  PROFILE PARENT GPU
gpu-0-mig-2g48gb-5-0    2g.48gb GPU-676c1adb-1b9d-b501-13e6-05cc9b040785
gpu-0-mig-1g24gb-14-6   1g.24gb GPU-676c1adb-1b9d-b501-13e6-05cc9b040785
gpu-0-mig-1g24gb-14-9   1g.24gb GPU-676c1adb-1b9d-b501-13e6-05cc9b040785
```

```
# Which pod asks for which request of the shared ResourceClaim?
$ kubectl get pods -n mig-test -o custom-columns='POD:.metadata.name,NODE:.spec.nodeName,
    REQUEST:.spec.containers[0].resources.claims[0].request'
POD                                     NODE                           REQUEST
llm-d-inference-gateway-85bcc65df8-rsbld   ip-10-0-42-66.ec2.internal     <none>
mig-large-training-pod                  ip-10-0-107-255.ec2.internal   mig-large
qwen3-epp-5689fc9496-7ntvq              ip-10-0-42-66.ec2.internal     <none>
vllm-qwen3-a-6885d59876-n4g85           ip-10-0-107-255.ec2.internal   mig-serve-a
vllm-qwen3-b-6795466874-jj4n7           ip-10-0-107-255.ec2.internal   mig-serve-b

# Which slice did the scheduler give each request?
$ kubectl get resourceclaim shared-mig-claim -n mig-test -o jsonpath='{"REQUEST\t\tDEVICE\n"}...'
REQUEST         DEVICE
mig-large       gpu-0-mig-2g48gb-5-0
mig-serve-a     gpu-0-mig-1g24gb-14-6
mig-serve-b     gpu-0-mig-1g24gb-14-9

# Proof from inside: same GPU UUID, different MIG UUIDs.
$ for t in pod/mig-large-training-pod deploy/vllm-qwen3-a deploy/vllm-qwen3-b; do
>   kubectl exec -n mig-test $t -- nvidia-smi -L; done
GPU 0: NVIDIA RTX PRO 6000 Blackwell Server Edition (UUID: GPU-676c1adb-1b9d-...)
  MIG 2g.48gb     Device  0: (UUID: MIG-945cb005-349a-55fd-...)
GPU 0: NVIDIA RTX PRO 6000 Blackwell Server Edition (UUID: GPU-676c1adb-1b9d-...)
  MIG 1g.24gb     Device  0: (UUID: MIG-edf5d3ce-15cf-55d8-...)
GPU 0: NVIDIA RTX PRO 6000 Blackwell Server Edition (UUID: GPU-676c1adb-1b9d-...)
  MIG 1g.24gb     Device  0: (UUID: MIG-9cf287a1-f02c-5581-...)

# A real request through the gateway, answered from a 1g slice.
$ curl -sS http://$GW/v1/chat/completions -d '{"model":"Qwen/Qwen3-8B-FP8",...}' | jq -r ...
Sharing a GPU allows teams to efficiently utilize computational resources, reducing costs and improving access to high-performance computing. …
```

DAY 1

- Three pods land on three slices.
- One engineer. Three workloads. Beautiful.

DAY 30

- Fifty engineers want slices.
- Training jobs queue. Inference pods compete for the same slices.
- **Whose workload runs first?**

Our next steps after MIG and DRA.

NEXT QUARTER

- **Kueue**: MIG quota per team. Answers "whose workload runs first?"
- **Group-shared claims** (K8s 1.37, Beta): replace our hand-built shared claim.
- **Admission policy in-cluster**: the policy pack's rules for `kubectl` too.
- **llm-d, next layer**: KV-cache routing, autoscaling on EPP and vLLM metrics.
- **Pilots**: ConsumableShares and DynamicMIG on an H100 pool.

OPTIONS WE'RE WATCHING

- **KAI Scheduler**: GPU-aware fair share and gang scheduling. CNCF sandbox, v0.18.
- **NVIDIA Grove**: multi-pod inference topologies across NVLink. Beta, v0.2.
- **KServe** `LLMInferenceService`: llm-d behind one CR.
- **NVIDIA AICR**: validated, version-locked recipes for the GPU stack, with drift checks. Also available as Pulumi provider.



The elephant in the room.

Large, as in **Large** Language Model.

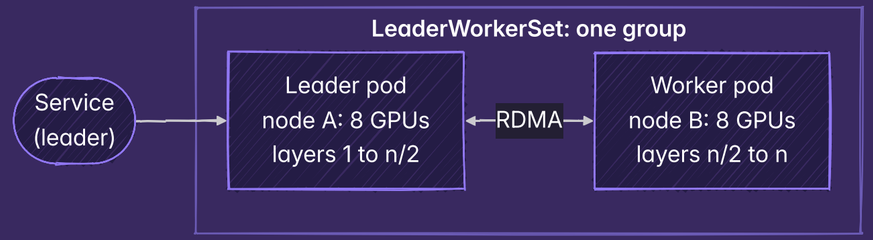



Address me

This is where it gets interesting.

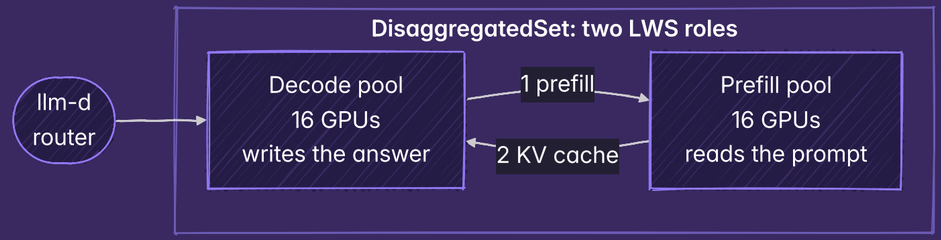
- **FITS A SLICE** — MIG + DRA. Everything you saw today.
- **FITS ONE NODE** — Whole GPUs. vLLM cuts every layer into pieces, one per GPU (tensor parallelism), ideally over NVLink.
- **BIGGER THAN THAT** — Two patterns:

BIGGER THAN ONE NODE



EXAMPLE	Llama 3.1 405B on 2 nodes × 8 GPUs.
SPLIT	Each layer across a node's 8 GPUs. The layers across the 2 nodes.
RUNS AS	One LWS group: created, updated, scaled and restarted together.
NEEDS	Fast links between nodes: InfiniBand, RoCE or EFA.

FRONTIER MIXTURE-OF-EXPERTS (MOE) MODELS



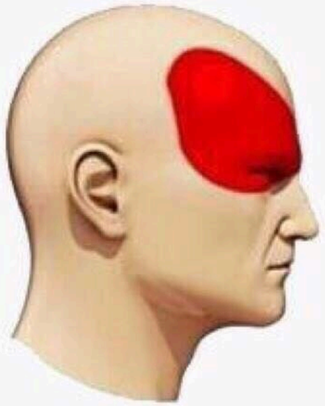
EXAMPLE	DeepSeek-R1 on 32× H200 (llm-d reference).
SPLIT	Experts spread over GPUs. Prompt and answer on separate pools.
RUNS AS	llm-d picks both pods. KV cache moves over NIXL.
NEEDS	RDMA. On GB200 NVL72, a DRA `ComputeDomain` : **same driver as our slices.**

Takeaways from our pain for you!

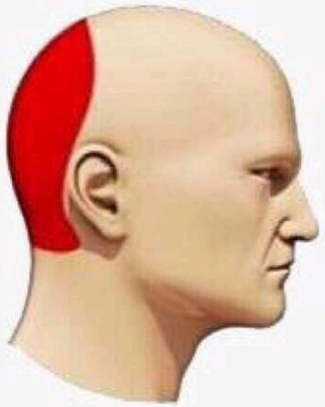


Types of Headaches

Migraine



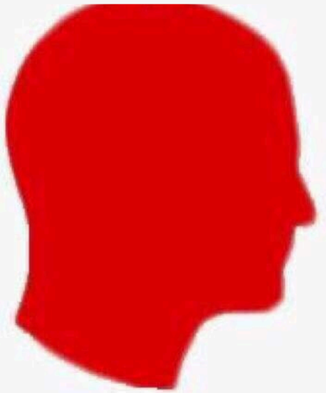
Hypertension



Stress



Creating an AI inference platform from scratch



When DRA + MIG beats time-slicing.

REACH FOR MIG + DRA WHEN

- Several teams or tenants share one card.
- You have latency targets: no neighbour may steal SMs or memory bandwidth.
- The model fits a slice: 5–20 GB on an A100 40GB, 24–48 GB on an RTX PRO 6000.
- Training and serving run on the same card.
- You want to charge per slice, not per card.

TIME-SLICING IS STILL FINE WHEN

- One team, trusted workloads.
- Notebooks, dev boxes, CI smoke tests.
- Bursty jobs that need the whole card for a moment.
- The card has no MIG: L4, A10G, consumer GPUs.
- You can live without memory or fault isolation.

If an OOM or a crash in one pod must not touch another, it's MIG.

How to roll it out safely.

01

Pick slice sizes per node pool

Label the pool, e.g. `mig.config=all-balanced`. Changing it later means emptying the GPUs first.

02

Give each slice size a name

The driver has one class for all MIG slices. Add one per size, e.g. `mig-1g-24gb`.

03

Guard twice

Policy pack in `pulumi preview`. Kyverno at the API server.

04

Start DRA after MIG

Every new GPU node races. Run the DRA plugin only where `mig.config.state=success`.

05

Measure, then bill

dcgm-exporter 4.9+ maps MIG metrics to pods. Bill on slice-hours, not utilization.

06

Bring old manifests along

The same DeviceClass maps `nvidia.com/mig-1g.24gb` to a slice (GA in 1.37). Old pods run unchanged.

Guardrails: bad slices die twice.

BEFORE DEPLOY: PULUMI POLICY PACK

```
// infra/mig-policy/index.ts
const blocked = ["7g.40gb", "4g.40gb",
                 "3g.40gb", "7g.80gb", "4g.96gb"];

new PolicyPack("mig-policy", { policies: [{
  name: "enforce-small-mig-profiles",
  enforcementLevel: "mandatory",
  validateResource: (args, report) => {
    if (!isResourceClaim(args)) return;
    for (const expr of celSelectors(args.props))
      for (const p of blocked)
        if (expr.includes(`"${p}"`))
          report(`MIG profile ${p} not allowed`);
  },
}]});
```

Runs on `pulumi preview` and `pulumi up`.

AT THE API SERVER: KYVERNO

```
# infra/kyverno/enforce-small-mig-profiles.yaml
apiVersion: policies.kyverno.io/v1
kind: ValidatingPolicy
spec:
  validationActions: [Deny]
  matchConstraints:
    resourceRules:
    - apiGroups: ["resource.k8s.io"]
      apiVersions: ["v1"]
      operations: ["CREATE", "UPDATE"]
      resources: ["resourceclaims",
                  "resourceclaimtemplates"]
  variables:
  - name: blocked
    expression: "['7g.40gb', '4g.40gb', ...]"
  validations:
  - expression: >-
      variables.requests.all(r, r.exactly.selectors
        .all(s, !variables.blocked.exists(p,
          s.cel.expression.contains(p))))
    message: MIG profile not allowed.
```

Catches `kubectl`, operators and Kueue.

It's open source.



github.com/dirien/stop-wasting-gpus-how-we-built-a-golden-path-for-gpu-sharing-on-kubernetes

`infra/`

The Pulumi program: EKS 1.37, GPU Operator, DRA driver, vLLM, gateway, Grafana.

`infra/mig-policy/`

The policy pack from this talk.

`infra/kyverno/`

The same rule at the API server.

`infra/dashboards/`

The fixed DCGM MIG dashboard.

`DEMO.md`

This demo, step by step, with recovery and teardown.

Swap the card with `gpuInstanceType` `gpuCapacityType` `gpuAvailabilityZone`

Stop wasting whole GPUs.

Pulumi

Questions?

Meet me and my colleagues at our booth.

BOOTH 200



Thanks.



Resources



Engin Diri

Pulumi

dirien engin-diri

Slides + Demo

github.com/dirien/stop-wasting-gpus-how-we-built-a-golden-path-for-gpu-sharing-on-kubernetes