KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Deep Dive Into the DRA Driver for NVIDIA GPUs: What Works Today, What's Coming and Why It Matters

Shiva Krishna Merla, NVIDIA
Varun Ramachandra Sekar, NVIDIA



Agenda

KubeCon CloudNativeCon North America 2026

You will leave knowing

- What works today
- What the upcoming release adds
- How to migrate from the device plugin
- Where we need your feedback

01 **Overview, Deployment**
GPU plugin, ComputeDomain controller and plugin

02 **GPU allocation, sharing & health**
Full GPU, MIG, MPS, time-slicing, shares

03 **ComputeDomains**
Multi-node NVLink on NVL72 · demo

04 **Virtualization**
GPU passthrough to KubeVirt VMs · demo

05 **Roadmap & migration**
v0.5.0, the upcoming release, moving off the device plugin

06 **Why DRA now**
Kubernetes 1.36

07 **Call to action**
Try it and tell us the gaps

Driver Overview | How the driver plugs into Kubernetes

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

CONTROL PLANE

- kube-scheduler
- kube-apiserver
- compute-domain-controller

API OBJECTS

- DeviceClass
- ResourceSlice · GPUs
- ResourceClaimTemplate
- ResourceClaim
- ComputeDomain
- ResourceSlice · channels
- ComputeDomainClique

GPU WORKER NODE

- gpu-kubelet-plugin
- kubelet
- compute-domain-kubelet-plugin
- IMEX daemon
- GPUs
- Workload pod

GPU allocation: one kubelet plugin. • **ComputeDomains:** a controller plus a kubelet plugin.

KubeCon CloudNativeCon
North America 2026

Deployment

Helm + GPU Operator



Standalone Helm Chart | Install and configure

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

HELM INSTALL

```
# Install from registry.k8s.io
helm install dra-driver-nvidia-gpu \

oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu \
  --version 0.5.1 \
  --namespace dra-driver-nvidia-gpu --create-namespace \
  --set gpuResourcesEnabledOverride=true
```

VALUES

```
# values.yaml
resources:
  gpus:
    enabled: true
  computeDomains:
    enabled: true
gpuResourcesEnabledOverride: true
featureGates:
  DynamicMIG: true
  NVMLDeviceHealthCheck: true
  ConsumableShares: true
```

Where to get it

registry.k8s.io and NGC. Or let the GPU Operator deploy it.

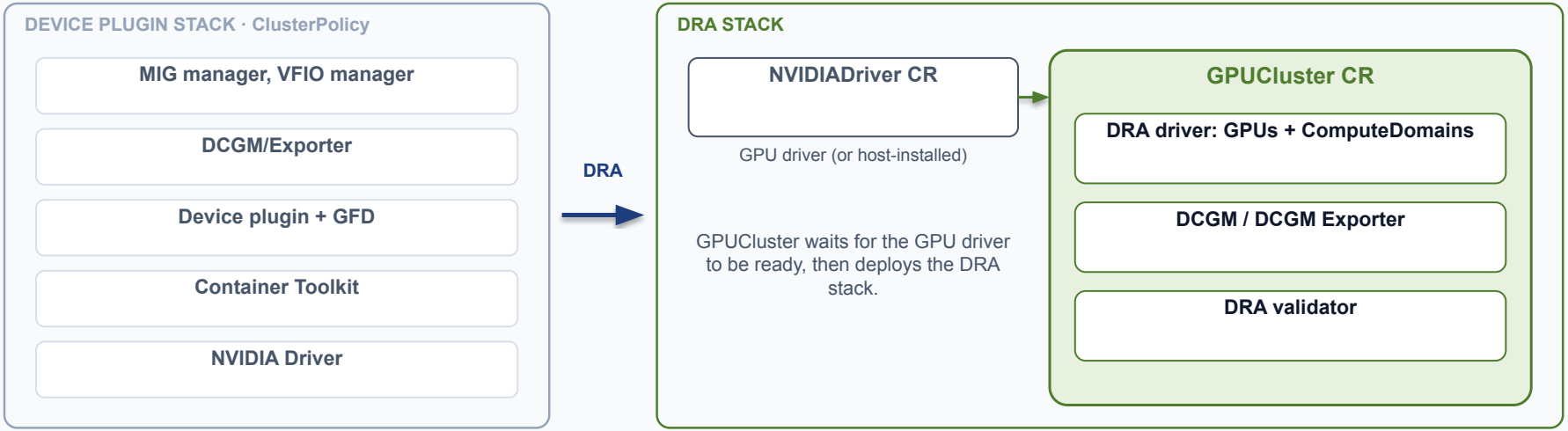
Both plugins on by default

GPU allocation and ComputeDomains. Turn either off in values.

GPU Operator | GPUCluster for DRA

TODAY · v26.7

KubeCon CloudNativeCon North America 2026



SWITCH TO DRA

```
# gpu-operator Helm values
clusterPolicy:
  deployCR: false
gpuCluster:
  deployCR: true
draDriver:
  version: v0.5.1
```

One or the other

ClusterPolicy and GPUCluster can't coexist in a cluster.

No device plugin, no GFD

ResourceSlices carry the device attributes instead of node labels.

Experimental

GPUCluster is off by default in the operator today.

https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/dra-intro-install.html

DRA Building Blocks | DeviceClass

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

1 DeviceClass — Helm install creates it

2 ResourceSlice — Driver publishes it

3 ResourceClaim — You request it

INSTALL THE DRIVER

```
helm install dra-driver-nvidia-gpu oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu
```

DEVICECLASSES THE CHART CREATES

`gpu.nvidia.com`	Full or shared GPU
`mig.nvidia.com`	MIG device
`vfio.gpu.nvidia.com`	GPU passthrough to VMs
`compute-domain-default-channel.nvidia.com`	IMEX channel
`compute-domain-daemon.nvidia.com`	IMEX daemon (internal)

DEVICECLASS · gpu.nvidia.com

```
apiVersion: resource.k8s.io/v1
kind: DeviceClass
metadata:
  name: gpu.nvidia.com
spec:
  selectors:
  - cel:
      expression: >-
        device.driver == 'gpu.nvidia.com' &&
        device.attributes['gpu.nvidia.com']
          .type == 'gpu'
  extendedResourceName: nvidia.com/gpu
```

DRA Building Blocks | ResourceSlice

TODAY · v0.5

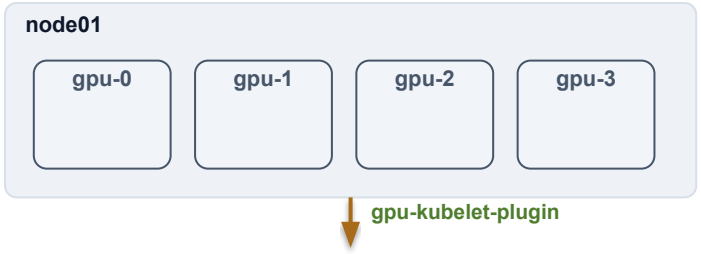
KubeCon CloudNativeCon North America 2026

1 **DeviceClass**
Helm install creates it

› 2 **ResourceSlice**
Driver publishes it

› 3 **ResourceClaim**
You request it

ONE OR MORE, PER DRIVER



- **Attributes**: type, productName, architecture, uuid, cudaComputeCapability, driverVersion, PCI and NUMA
- **Capacity**: memory; counters for MIG profiles
- **Also**: IMEX channels from compute-domain.nvidia.com

RESOURCESLICE · published by the driver

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
spec:
  driver: gpu.nvidia.com
  nodeName: node01
  pool:
    name: node01
  devices:
  - name: gpu-0
    attributes:
      type: {string: gpu}
      productName: {string: NVIDIA B200}
      architecture: {string: Blackwell}
      cudaComputeCapability: {version: 10.0.0}
    capacity:
      memory: {value: 180Gi}
```

DRA Building Blocks | ResourceClaim

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

1 **DeviceClass**
Helm install creates it

› 2 **ResourceSlice**
Driver publishes it

› 3 **ResourceClaim**
You request it

1 · YOU WRITE A CLAIM TEMPLATE

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: one-b200
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          selectors:
          - cel:
              expression: >-
                device.attributes[
                  'gpu.nvidia.com']
                  .productName ==
                  'NVIDIA B200'
```

2 · THE POD REFERENCES IT

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: train-0
spec:
  resourceClaims:
  - name: gpu
    resourceClaimTemplateName: one-b200
  containers:
  - name: ctr
    image: cuda-app
    resources:
      claims:
      - name: gpu
```

3 · SCHEDULER ALLOCATES

```yaml
# ResourceClaim (generated)
status:
  allocation:
    devices:
      results:
      - request: gpu
        driver: gpu.nvidia.com
        pool: node01
        device: gpu-0
  reservedFor:
  - resource: pods
    name: train-0
```

KubeCon CloudNativeCon
North America 2026

GPU Allocation & Sharing

Full GPUs, MIG, MPS, Time-Slicing, Consumable Shares



GPU Allocation Modes | One claim model for pods and VMs

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

Full GPU

Pod



`gpu.nvidia.com`

Whole GPU to one claim

v0.5.0

Shared GPU

Pods



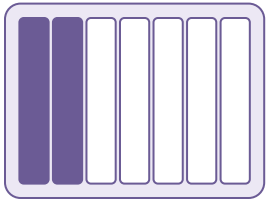
`gpu.nvidia.com`

Whole GPU shared: time-slicing takes turns, MPS runs concurrently

v0.5.0

MIG slice

Pod



`mig.nvidia.com`

Hardware-isolated memory and SMs, static or dynamic

v0.5.0

Passthrough

VM



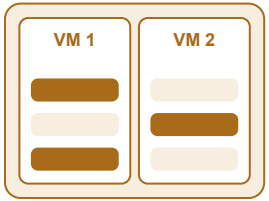
`vfio.gpu.nvidia.com`

Whole GPU bound to vfio-pci for KubeVirt

v0.5.0

vGPU

VM



`vGPU profiles`

Dedicated framebuffer per VM, time-sliced or MIG-backed compute

Upcoming

Horizontal bands = the whole GPU shared over time. **Vertical slices** = dedicated memory and SMs. **vGPU** = dedicated framebuffer, shared compute.

Every mode is a DeviceClass plus a ResourceClaim. Only the class and the config change.

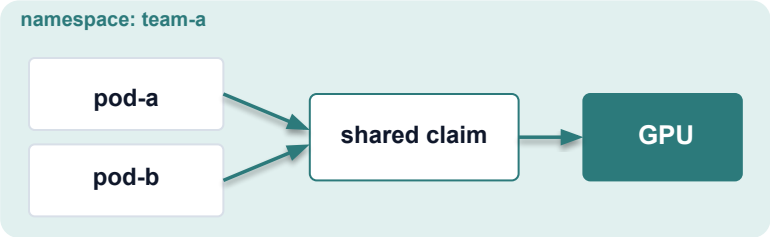
GPU Sharing | User-mediated vs system-mediated

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

User-mediated

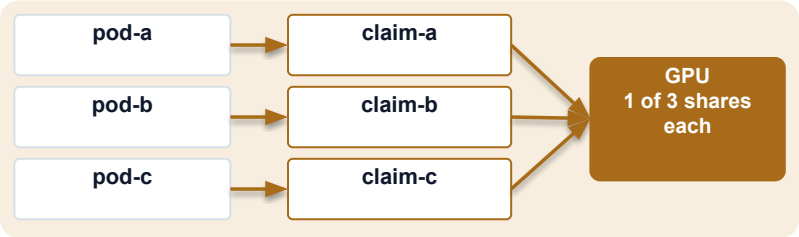
One claim, shared by the containers that reference it



- **MPS**: thread % and pinned memory limits (MPSSupport)
- **Time-slicing**: per-claim interval (TimeSlicingSettings)
- **Scope**: containers of one pod, or pods sharing one claim

System-mediated

Independent claims; the scheduler accounts for capacity



- **One claim per pod**: pod-a → claim-a, pod-b → claim-b, …
- **Across namespaces**: team-a, team-b, team-c each keep their own claim
- **Consumable Shares**: memory, N shares, or unlimited

Same namespace, shared claim: user-mediated.
Any namespace, different claims: system-mediated.

Consumable Shares | Sharing across namespaces

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

1 · DRIVER PUBLISHES SHARES

```
# ResourceSlice (driver)
# --consumable-shares=4
devices:
- name: gpu-0
  allowMultipleAllocations: true
  capacity:
    shares:
      value: "4"
    memory:
      value: 180Gi
```

2 · EACH NAMESPACE CLAIMS A SHARE

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: one-share
  namespace: team-a    # or team-b
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          capacity:
            requests:
              shares: "1"
```

Feature gates

Driver	Helm	Kubernetes
`ConsumableShares`	`consumableShares`	`DRAConsumableCapacity`
alpha	memory | N | unlimited	on in 1.36

Works on full GPUs and MIG

MPS cannot be combined with shares today. The driver rejects the claim.

KubeCon CloudNativeCon North America 2026

GPU Health Monitoring

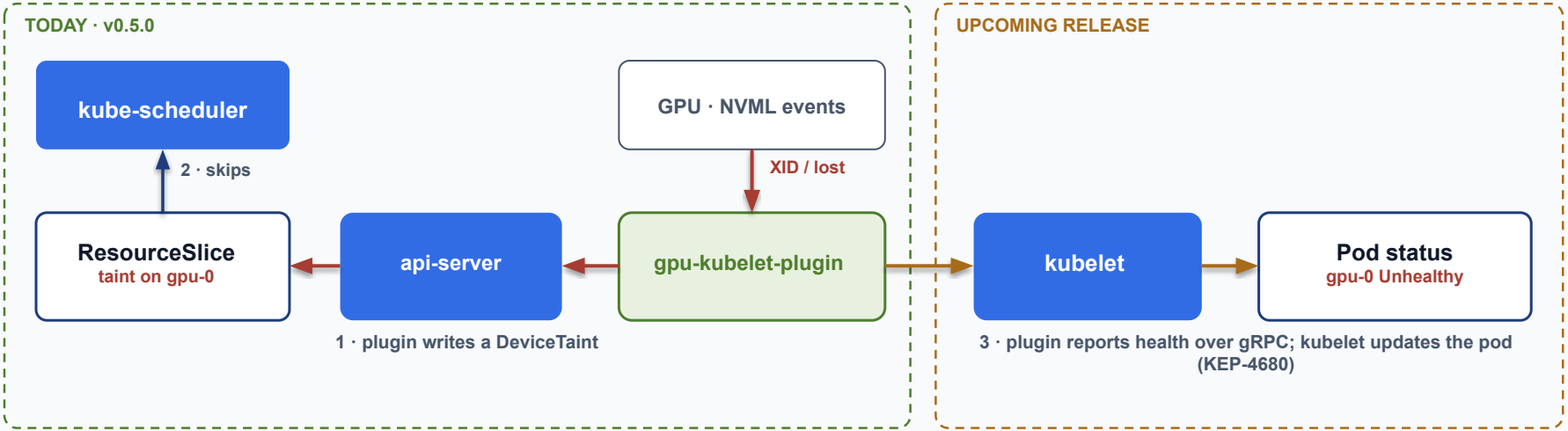
Device Taints and Resource Health Status



Health Status Propagation | GPU to slice and pod

v0.5.0 → UPCOMING

KubeCon CloudNativeCon North America 2026



ResourceSlice end

```
# ResourceSlice
devices:
- name: gpu-0
  taints:
  - key: gpu.nvidia.com/xid
    value: "79"
    effect: NoSchedule
    timeAdded: "2026-11-04T18:22:05Z"
```

Pod end

```
# Pod status
containerStatuses:
- name: cuda-ctr
  allocatedResourcesStatus:
  - name: claim:gpu/gpu
    resources:
    - resourceID: k8s.gpu.nvidia.com/claim=<claim-uid>-gpu-0
      health: Unhealthy
      message: fatal XID 79 reported by NVML
```

- **Feature Gates**: Driver: **NVMLDeviceHealthCheck** (alpha), K8s: **DRADeviceTaints** (on in 1.36)

KubeCon CloudNativeCon North America 2026

GPUs for Virtualization

KubeVirt VMs on the same DRA driver



- Graduated CNCF Sandbox Project
- Run VirtualMachines on Kubernetes
- Beta support for GPUs with DRA in v1.9

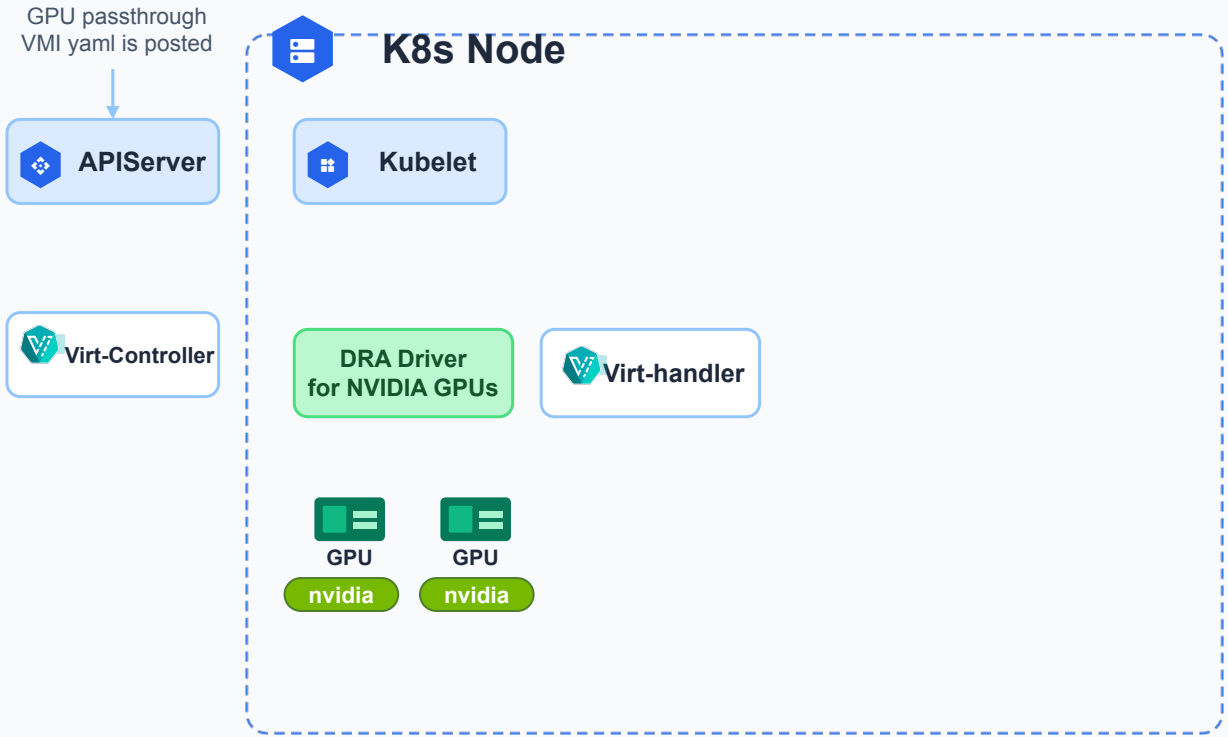


GPU Passthrough to VMs | KubeVirt with the DRA driver

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

```yaml
---
apiVersion: v1
kind:ResourceClaim
metadata:
  name: vfio
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        allocationMode: ExactCount
        count: 1
        deviceClassName:
          vfio.gpu.nvidia.com
---
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: vm-demo
spec:
  resourcClaims:
  - name: gpu-claim
    resourceClaimName:vfio
  domain:
    devices:
      gpus:
      - name: gpu-0
        claimName: gpu-claim
        requestName: gpu
```

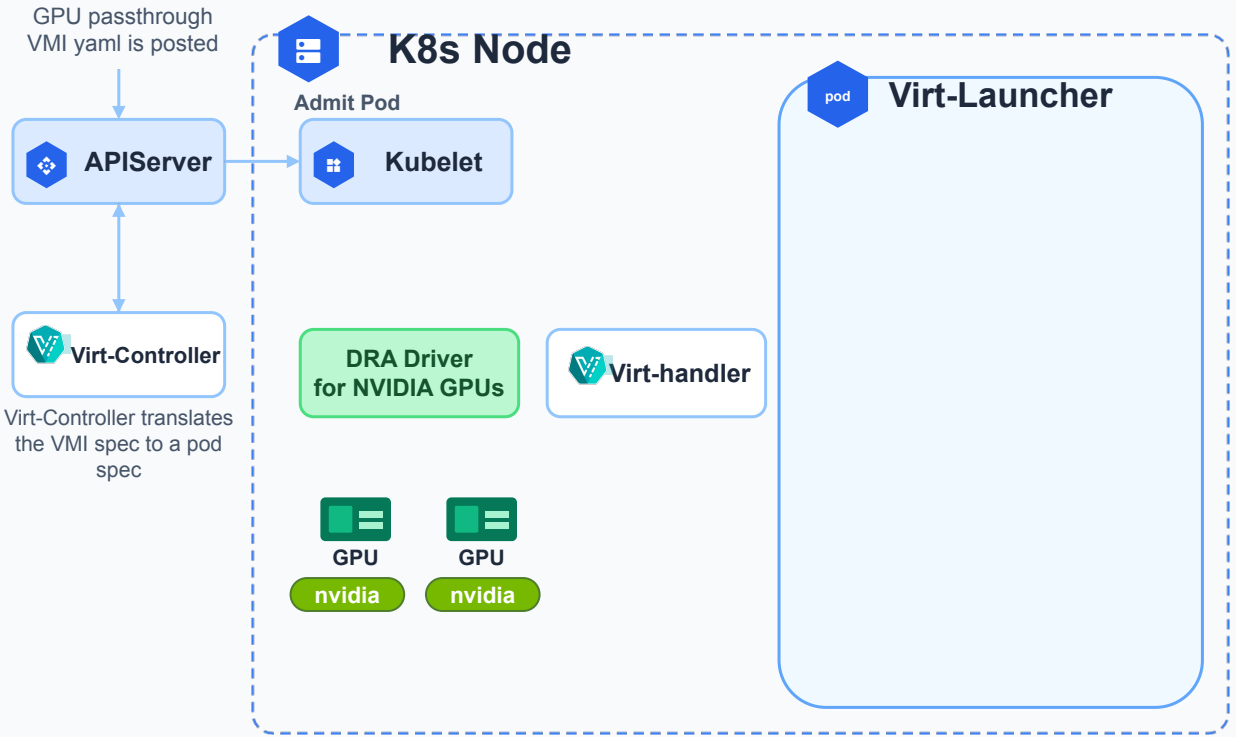


GPU Passthrough to VMs | KubeVirt with the DRA driver

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

```yaml
---
apiVersion: v1
kind:ResourceClaim
metadata:
  name: vfio
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        allocationMode: ExactCount
        count: 1
        deviceClassName:
          vfio.gpu.nvidia.com
---
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: vm-demo
spec:
  resourcClaims:
  - name: gpu-claim
    resourceClaimName:vfio
  domain:
    devices:
      gpus:
      - name: gpu-0
        claimName: gpu-claim
        requestName: gpu
```

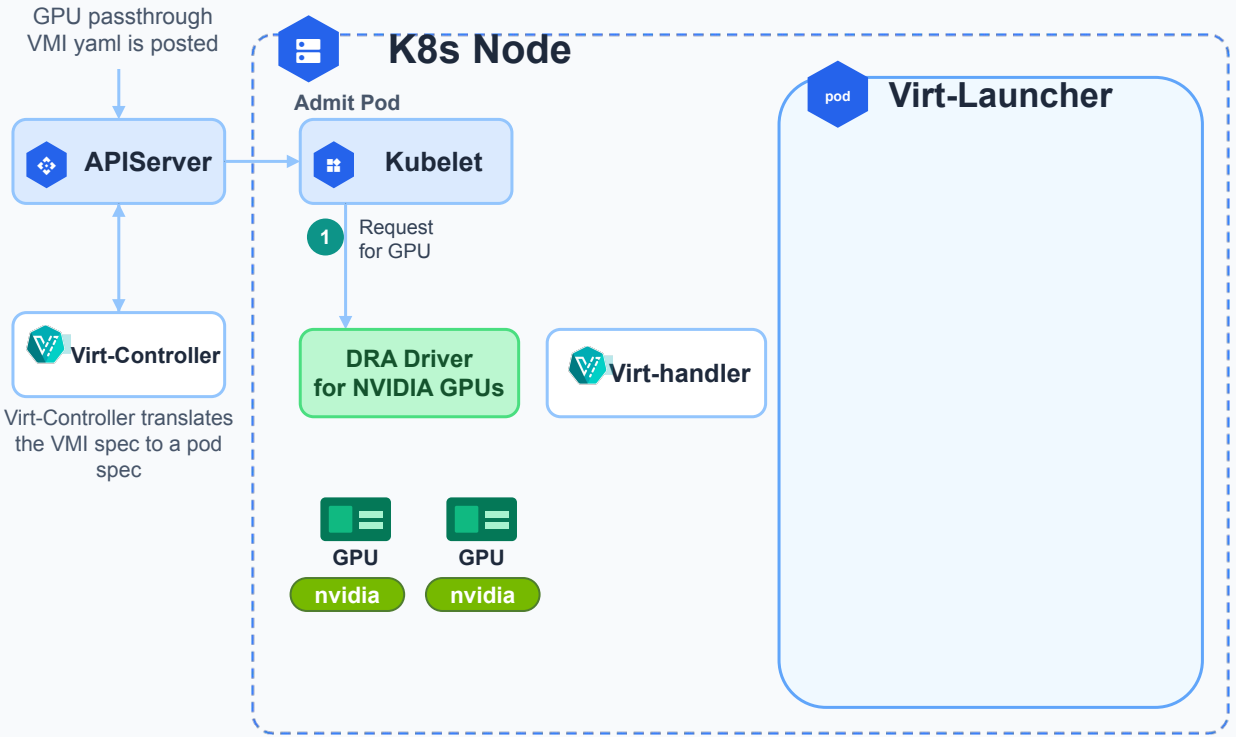


GPU Passthrough to VMs | KubeVirt with the DRA driver

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

```yaml
---
apiVersion: v1
kind:ResourceClaim
metadata:
  name: vfio
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        allocationMode: ExactCount
        count: 1
        deviceClassName:
          vfio.gpu.nvidia.com
---
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: vm-demo
spec:
  resourcClaims:
  - name: gpu-claim
    resourceClaimName:vfio
  domain:
    devices:
      gpus:
      - name: gpu-0
        claimName: gpu-claim
        requestName: gpu
```

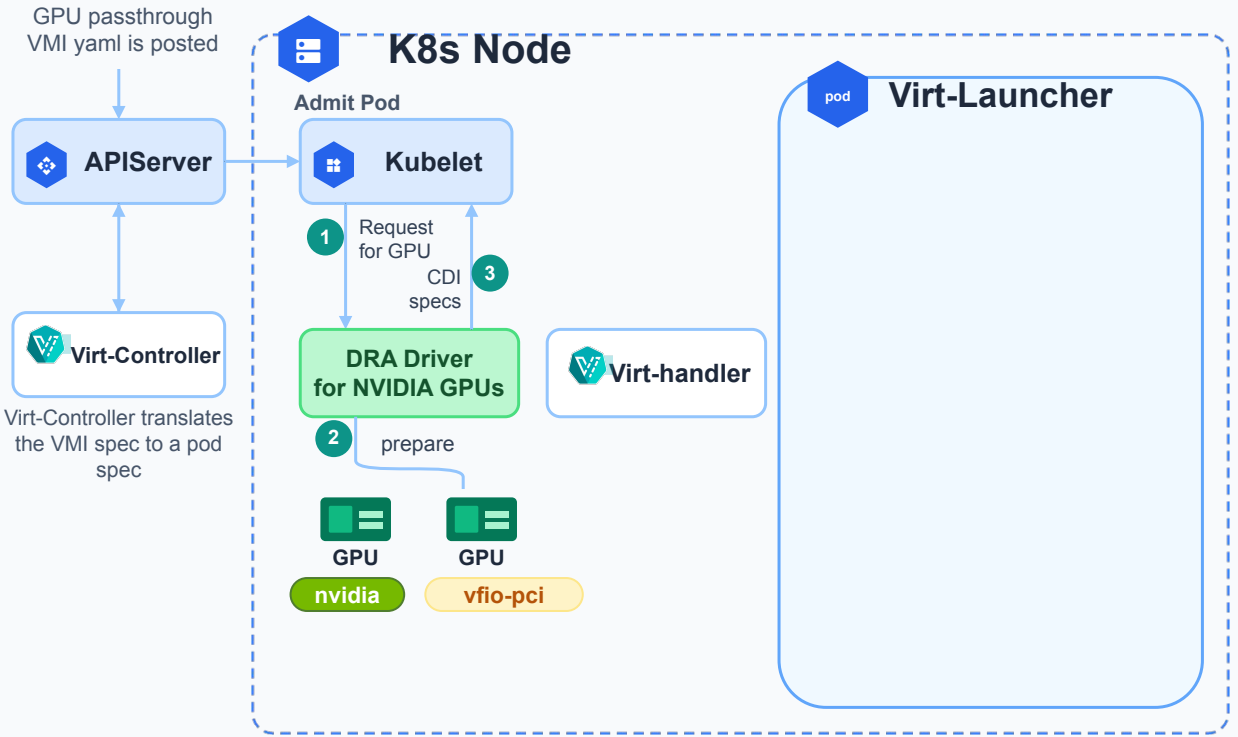


GPU Passthrough to VMs | KubeVirt with the DRA driver

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

```yaml
---
apiVersion: v1
kind:ResourceClaim
metadata:
  name: vfio
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        allocationMode: ExactCount
        count: 1
        deviceClassName:
          vfio.gpu.nvidia.com
---
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: vm-demo
spec:
  resourcClaims:
  - name: gpu-claim
    resourceClaimName:vfio
  domain:
    devices:
      gpus:
      - name: gpu-0
        claimName: gpu-claim
        requestName: gpu
```

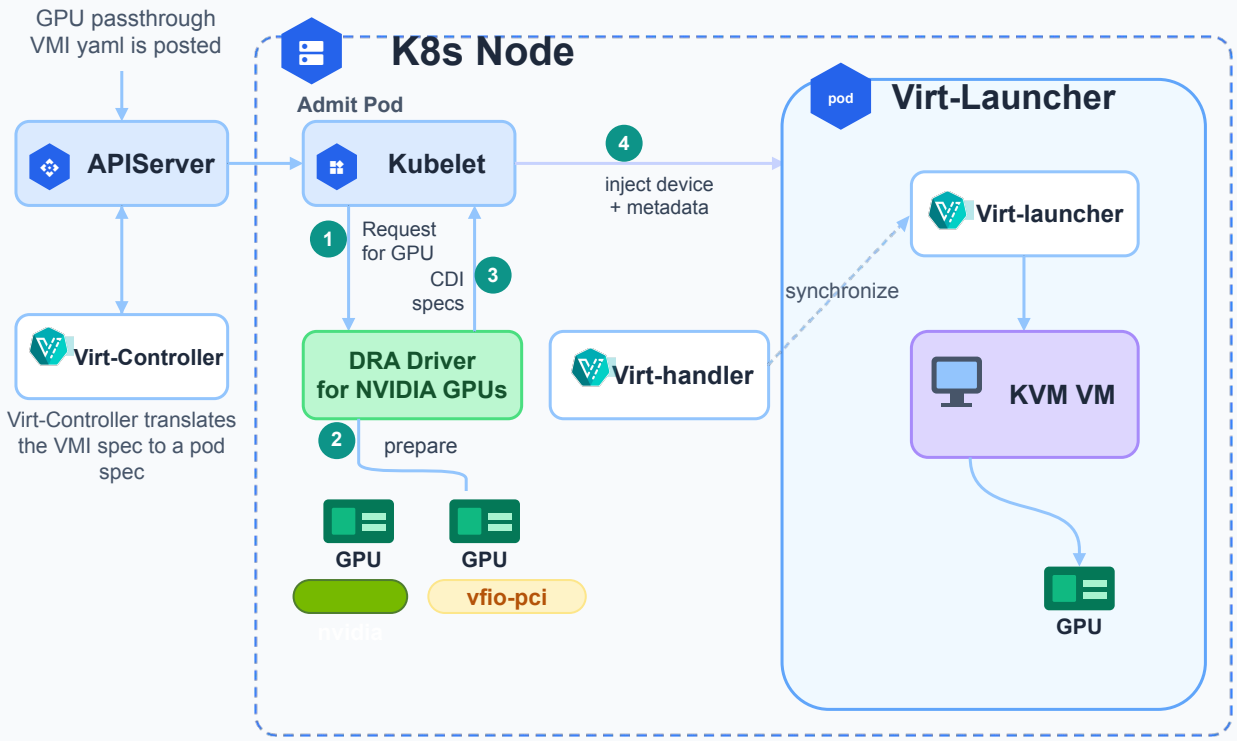


GPU Passthrough to VMs | KubeVirt with the DRA driver

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

```yaml
---
apiVersion: v1
kind:ResourceClaim
metadata:
  name: vfio
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        allocationMode: ExactCount
        count: 1
        deviceClassName:
          vfio.gpu.nvidia.com
---
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: vm-demo
spec:
  resourcClaims:
  - name: gpu-claim
    resourceClaimName:vfio
  domain:
    devices:
      gpus:
      - name: gpu-0
        claimName: gpu-claim
        requestName: gpu
```




```
/var/run/kubernetes.io/dra-device-attributes/resourceclaims/<claim>/<request>/<driver>-metadata.json
```

One GPU, Two Device Types | gpu and vfio

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

ResourceSlice · driver gpu.nvidia.com

```yaml
# v0.5.0
devices:
- name: gpu-0
  attributes:
    type:
      string: gpu
    resource.kubernetes.io/pciBusID:
      string: 0000:3b:00.0

- name: gpu-vfio-0
  attributes:
    type:
      string: vfio
    resource.kubernetes.io/pciBusID:
      string: 0000:3b:00.0
```

PassthroughSupport FeatureGate

v0.5.0 today

Both are advertised. The alternate device is removed after the first claim is prepared.

One GPU, Two Device Types | gpu and vfio

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

ResourceSlice · driver gpu.nvidia.com

```
# v0.5.0
devices:
- name: gpu-0
  attributes:
    type:
      string: gpu
    resource.kubernetes.io/pciBusID:
      string: 0000:3b:00.0

- name: gpu-vfio-0
  attributes:
    type:
      string: vfio
    resource.kubernetes.io/pciBusID:
      string: 0000:3b:00.0
```

PassthroughSupport FeatureGate

v0.5.0 today

Both are advertised. The alternate device is removed after the first claim is prepared.

Same physical GPU

gpu-0 and gpu-vfio-0 have the same PCI bus ID. Allocate one, not both.

One GPU, Two Device Types | gpu and vfio

v0.5 → UPCOMING

KubeCon CloudNativeCon North America 2026

ResourceSlice · driver gpu.nvidia.com

```yaml
# upcoming release: one shared counter
sharedCounters:
- name: gpu-0-counter-set
  counters:
    gpu-share: {value: "100"}
devices:
- name: gpu-0
  attributes:
    type: {string: gpu}
    resource.kubernetes.io/pciBusID: {string: "0000:3b:00.0"}
  consumesCounters:
  - counterSet: gpu-0-counter-set
    counters:
      gpu-share: {value: "100"}
- name: gpu-vfio-0
  attributes:
    type: {string: vfio}
    resource.kubernetes.io/pciBusID: {string: "0000:3b:00.0"}
  consumesCounters:
  - counterSet: gpu-0-counter-set
    counters:
      gpu-share: {value: "100"}
```

Upcoming: scheduler-enforced

Both consume the whole gpu-share counter, so the scheduler never co-allocates them.

Same physical GPU

gpu-0 and gpu-vfio-0 have the same PCI bus ID. Allocate one, not both.

One Request Flow | Template → Class → Pod or VM

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

RESOURCECLAIMTEMPLATE

```yaml
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          count: 1
          deviceClassName: gpu.nvidia.com
```

DEVICECLASS

gpu.nvidia.com

type == 'gpu'

POD

```yaml
kind: Pod
spec:
  resourceClaims:
  - name: gpu-claim
    resourceClaimTemplateName: gpu-template
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu-claim
        request: gpu-template
```

One Request Flow | Template → Class → Pod or VM

TODAY · v0.5

KubeCon CloudNativeCon North America 2026

RESOURCECLAIMTEMPLATE

```
kind: ResourceClaimTemplate
metadata:
  name: gpu-template
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          count: 1
          deviceClassName: gpu.nvidia.com
```

DEVICECLASS

gpu.nvidia.com

type == 'gpu'

POD

```
kind: Pod
spec:
  resourceClaims:
  - name: gpu-claim
    resourceClaimTemplateName: gpu-template
  containers:
  - name: ctr
    resources:
      claims:
      - name: gpu-claim
        request: gpu-template
```

RESOURCECLAIMTEMPLATE

```
kind: ResourceClaimTemplate
metadata:
  name: vfio-template
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          count: 1
          deviceClassName: vfio.gpu.nvidia.com
```

DEVICECLASS

vfio.gpu.nvidia.com

type == 'vfio'

KUBEVIRT VM

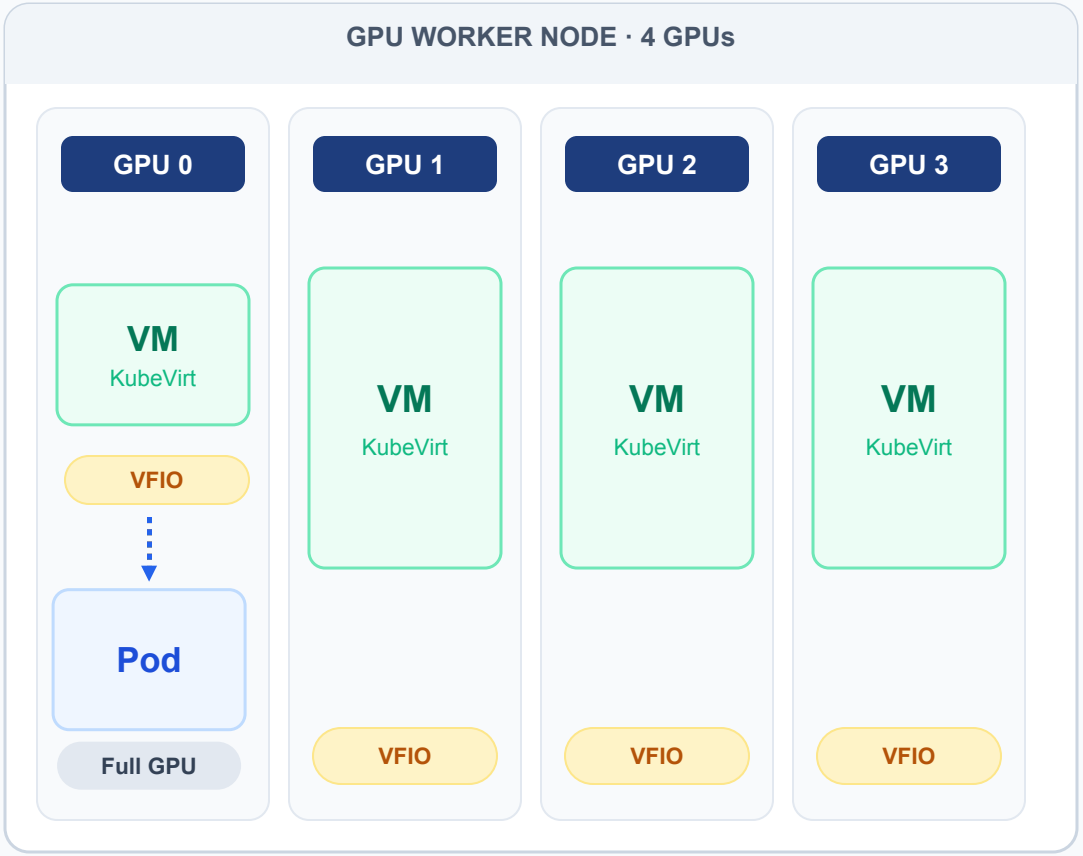
```
kind: VirtualMachine
spec:
  template:
    spec:
      resourceClaims:
      - name: vfio-claim
        resourceClaimTemplateName: vfio-template
      domain:
        devices:
          gpus:
          - name: gpu0
            claimName: vfio-claim
            requestName: gpu
```

Containers and VMs use similar resourceclaims. Only the DeviceClass differs.

Demo: Mixed Workloads | Pods and VMs on one node

TODAY · v0.5

KubeCon CloudNativeCon North America 2026



What we show

1. VMs claim GPUs 0–2
2. Pod with full-GPU claim in Pending state
3. Delete a VM: its GPU returns to nvidia driver
4. Pod goes to Running

KubeCon CloudNativeCon North America 2026

ComputeDomains

Multi-node NVLink on GB200/GB300 and VR NVL72



ComputeDomains and IMEX | What they are

IMEX

The NVIDIA service that lets GPUs share memory across nodes

- **NVLink clique**: nodes on one NVLink fabric, e.g. one NVL72 rack (18 trays × 4 GPUs)
- **IMEX domain**: the IMEX daemons of one clique for an ensemble
- **IMEX channel**: /dev/nvidia-caps-imex-channels/channelN, the handle a workload uses to import memory

ComputeDomain

A Kubernetes resource that makes IMEX an implementation detail

- **You create** a ComputeDomain and reference its claim template from your pods
- **The controller** creates IMEX DaemonSet and updates ComputeDomain statuses
- **The plugin** starts IMEX daemons where pods land and injects an IMEX channel into each workload container

```yaml
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: my-compute-domain
spec:
  channel:
    resourceClaimTemplate:
      name: my-cd-channel
```

```yaml
# Pod spec
resourceClaims:
- name: imex-channel
  resourceClaimTemplateName: my-cd-channel
```

ComputeDomains Today | One pod per node

TODAY · v0.5

KubeCon CloudNativeCon North America 2026



How it works in v0.5.0

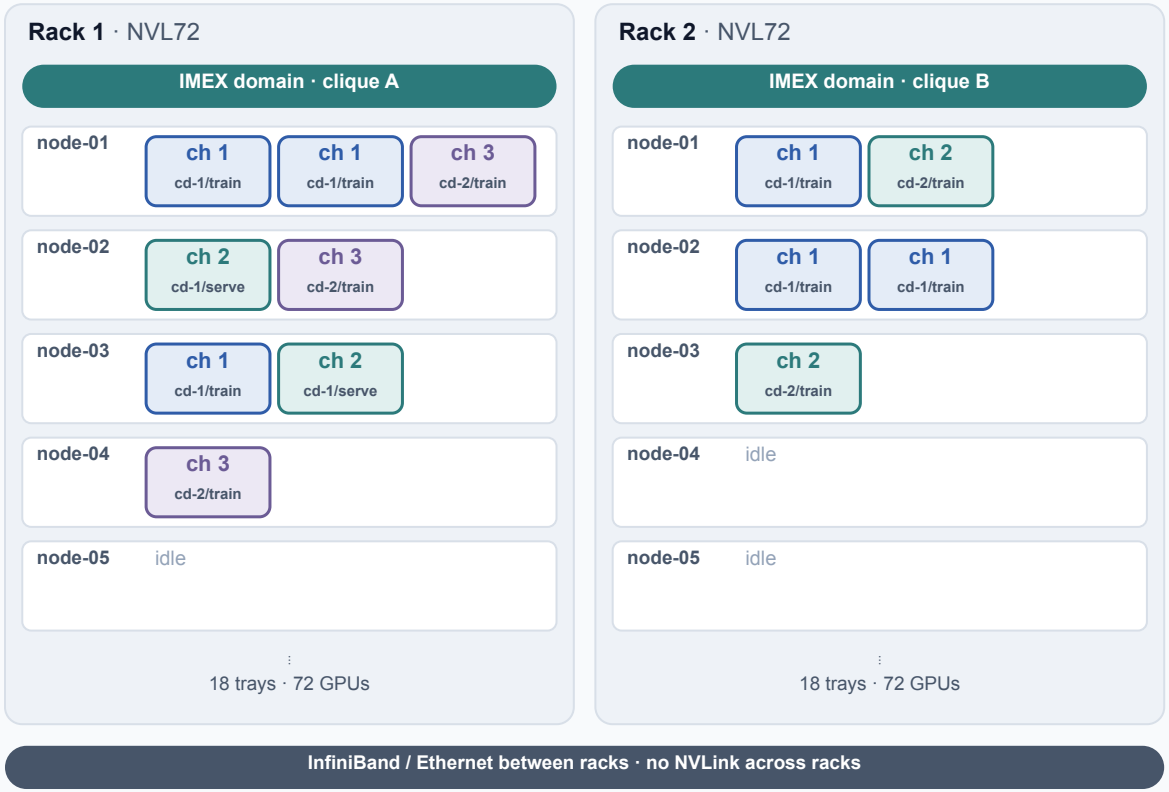
- **Every pod gets channel 0** of its IMEX domain
- **One pod per node** per ComputeDomain
- **One ComputeDomain** per node at a time
- **cd-1 spans two racks**: one IMEX domain per clique

Cost: idle GPUs and IMEX daemon startup on every job

ComputeDomains: Coming Next | Channels per template

UPCOMING

KubeCon CloudNativeCon North America 2026



Channel table

Clique A

ch 1	`cd-1/train`
ch 2	`cd-1/serve`
ch 3	`cd-2/train`

Clique B

ch 1	`cd-1/train`
ch 2	`cd-2/train`

- **Same template, same clique** → same channel on every node
- **Unique within a clique**; each clique numbers its own

Static IMEX Daemons | Take IMEX off the job-start path

UPCOMING

KubeCon CloudNativeCon North America 2026

The scale problem

- Every job start creates a DaemonSet, cliques and node labels
- Each node runs one ComputeDomain at a time
- Daemon bootstrap sits on the job-start path

Shipped already: ComputeDomainCliques scope daemon coordination to one NVLink clique (beta, on).

computedomaindaemonset.yaml

```yaml
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomainDaemonSet
metadata:
  name: default
spec:
  maxNodesPerClique: 18
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailableCliques: 1
      minReadySeconds: 30
```

Fast starts

One IMEX daemon per node, already running. A job only reserves a channel.

Shared by every CD

ComputeDomains mix on a node, isolated by channel.

Safe operations

Admin-owned, not restarted by driver upgrades; rolled out clique by clique.

Helm: `imex.mode=driverManaged` · `imex.scope=cluster` · `imex.isolation=channel` · gates `ComputeDomainDaemonSet` + `ComputeDomainChannelReservations (alpha)`

Same-Node NVLink Between Pods | No hostIPC

UPCOMING

KubeCon CloudNativeCon North America 2026

Today: sharing GPU memory across pods

Disaggregated inference (KV cache), RL weight sync

CUDA IPC handles

Pods must share an IPC/PID namespace: hostIPC, or one big pod

POSIX FD handles

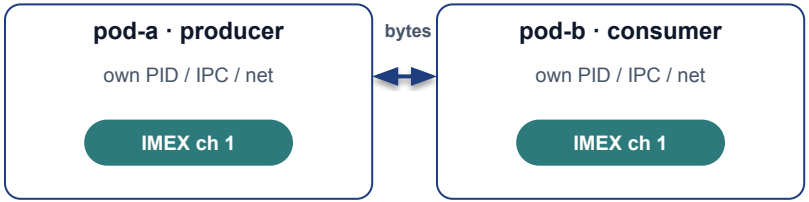
Need a shared socket path (hostPath) and FD passing

Copy through host or network

Leaves NVLink bandwidth unused

Next: CUDA fabric handles + IMEX channel

The same mechanism multi-node NVLink already uses



- **Export** a fabric handle: plain bytes, send them over any transport
- **Import** works only in a process that holds the same IMEX channel
- **Non-privileged pods**: no hostIPC, no hostPath, separate namespaces

Prototype on 8× A100 HGX: two non-privileged pods, separate PID/IPC/net namespaces, ~255 GB/s peer writes; without the channel, import fails with CUDA_ERROR_NOT_PERMITTED.

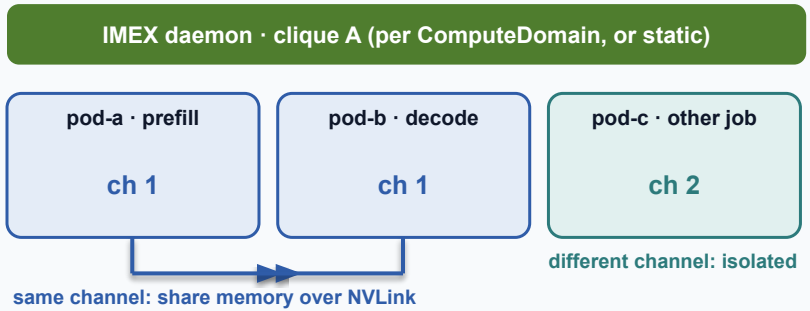
One API, MNNVL and HGX | ComputeDomain everywhere

UPCOMING

KubeCon CloudNativeCon North America 2026

MNNVL · GB200 / GB300 NVL72

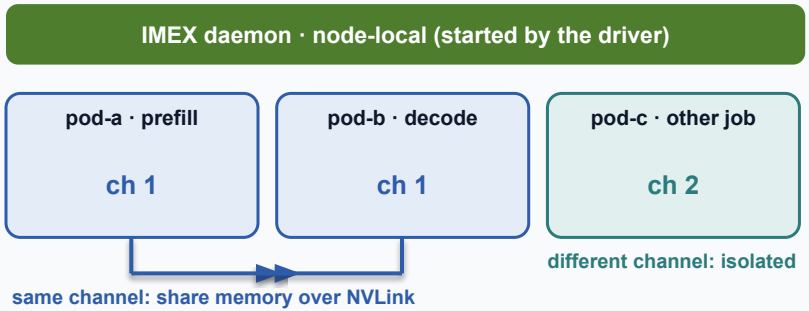
Node in an NVLink clique, the IMEX domain spans the clique



Channels unique per clique; cross-node too

HGX · single-node NVLink

No MNNVL clique, local IMEX daemon



Today: no IMEX on these nodes. Next: NodeLocalFabricIPC

SAME COMPUTEDOMAIN SPEC ON BOTH

```yaml
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: inference
spec:
  channels:
  - resourceClaimTemplate:
      name: kv-transfer
  - resourceClaimTemplate:
      name: other-job
```

Channel reservations

One channel per template. Same template, same channel, different templates are isolated.

Multiple allocations

One shared channel device per node, so many pods of a template land on one node.

IMEX already running

Static daemons on MNNVL, a node-local daemon on HGX. A pod just gets its channel.

Same-Node NVLink | What changes, by system

v0.5 → UPCOMING

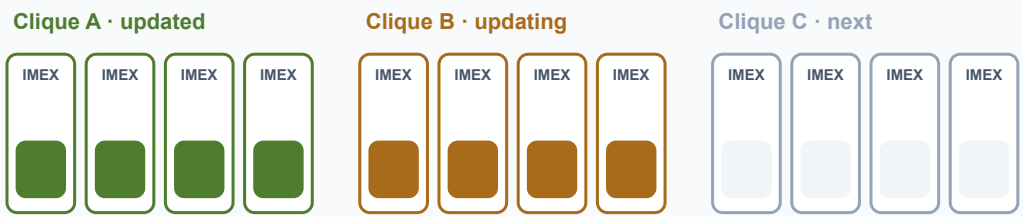
	MNNVL (GB200 / GB300 NVL72)	HGX (single-node NVLink)
IMEX today (v0.5)	Per ComputeDomain, channel 0	Not started, no channel
IMEX next	Per ComputeDomain, or static DaemonSet	Node-local daemon started by the driver
Pods of one CD on a node	Today one, next many (shared channel)	Next: many (shared channel)
Isolation	Channel per template, unique per clique	Channel per template on the node
Cross-node	Yes, within the clique	No (single node)
Driver gates (alpha, in review)	`ComputeDomainChannelReservations`, `ComputeDomainDaemonSet`	`NodeLocalFabricIPC`

Demo: ComputeDomains | Many CDs per node, unique channels

UPCOMING

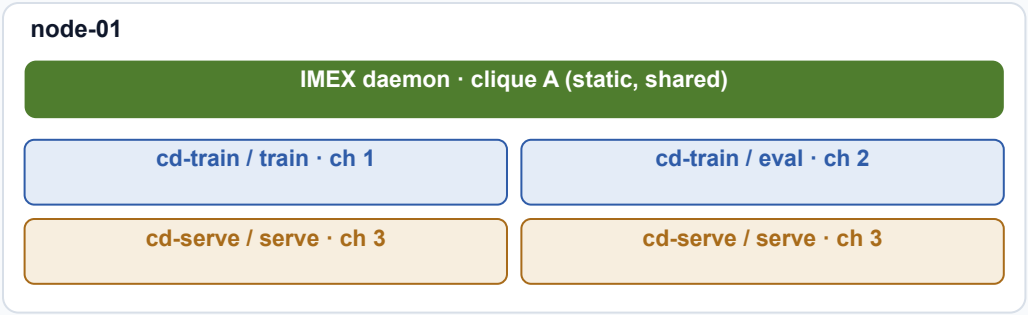
KubeCon CloudNativeCon North America 2026

1 · ONE IMEX DAEMONSET FOR THE WHOLE CLUSTER, ROLLED OUT CLIQUE BY CLIQUE



maxUnavailableCliques: 1 · daemons already running when jobs start

2 · TWO COMPUTEDOMAINS ON ONE NODE, EACH WORKLOAD ON ITS OWN CHANNEL



What we show

1 Create the default ComputeDomainDaemonSet: IMEX starts on every NVL72 node

2 Create cd-train and cd-serve, each with its own claim templates

3 Pods from both land on the same node

4 Each claim's status shows ChannelAdmitted and a channel unique in the clique

5 Update the daemon image: rolled out one clique at a time

KubeCon CloudNativeCon North America 2026

Roadmap & Migration

Currently supported in v0.5.0, the upcoming release, and moving off the device plugin



Feature Gates in v0.5.0 | Currently supported

TODAY · v0.5.0

KubeCon CloudNativeCon North America 2026

Area	Driver feature gate	Stage	Default	Kubernetes requirement
GPU sharing	`TimeSlicingSettings`	Alpha	off	—
	`MPSSupport`	Alpha	off	—
	`ConsumableShares`	Alpha	off	DRAConsumableCapacity (on in 1.36)
MIG	`DynamicMIG`	Alpha	off	DRAPartitionableDevices (on in 1.36)
Health	`NVMLDeviceHealthCheck`	Alpha	off	DRADeviceTaints (on in 1.36)
Virtualization	`PassthroughSupport`	Alpha	off	—
	`DeviceMetadata`	Alpha	off	—
	`FabricManagerPartitioning`	Alpha	off	—
ComputeDomains	`ComputeDomainCliques`	Beta	on	—
	`IMEXDaemonsWithDNSNames`	Beta	on	—
	`CrashOnNVLinkFabricErrors`	Beta	on	—
	`HostManagedIMEXDaemon`	Alpha	off	—

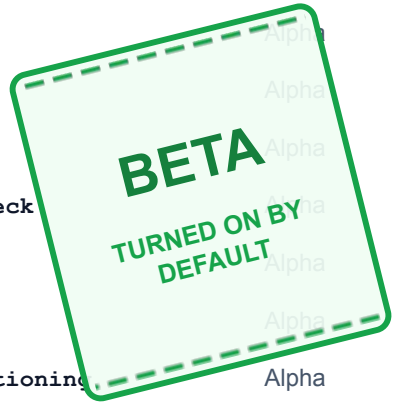
Set in Helm values: featureGates.<Name>=true. On Kubernetes 1.36 every upstream requirement above is already on.

Feature Gates in v0.5.0 | Upcoming release

UPCOMING RELEASE

KubeCon CloudNativeCon North America 2026

Area	Driver feature gate	Stage	Default	Kubernetes requirement
GPU sharing	`TimeSlicingSettings`	Alpha	off	—
	`MPSSupport`	Alpha	off	—
	`ConsumableShares`	Alpha	off	DRAConsumableCapacity (on in 1.36)
MIG	`DynamicMIG`	Alpha	off	DRAPartitionableDevices (on in 1.36)
Health	`NVMLDeviceHealthCheck`	Alpha	off	DRADeviceTaints (on in 1.36)
Virtualization	`PassthroughSupport`	Alpha	off	—
	`DeviceMetadata`	Alpha	off	—
	`FabricManagerPartitioning`	Alpha	off	—
ComputeDomains	`ComputeDomainCliques`	Beta	on	—
	`IMEXDaemonsWithDNSNames`	Beta	on	—
	`CrashOnNVLinkFabricErrors`	Beta	on	—
	`HostManagedIMEXDaemon`	Alpha	off	—



Set in Helm values: featureGates.<Name>=true. On Kubernetes 1.36 every upstream requirement above is already on.

What's Next | Graduations and new features

UPCOMING RELEASE

KubeCon CloudNativeCon North America 2026

Area	Currently supported (v0.5.0)	Upcoming release (planned)
Dynamic MIG	Alpha	**Beta, on by default; static MIG preserved**
Health checks	Alpha, taints	**Toward beta; health in pod status**
Consumable Shares	Alpha	**MPS compatibility, per-node config**
Passthrough	Alpha	**Shared counters, works with Dynamic MIG; per-node GPU config**
Fabric Manager partitioning	Alpha	**Beta**
vGPU	—	**Alpha: VGPUSupport, profiles advertised**
ComputeDomain channels	Channel 0	**Alpha: ComputeDomainChannelReservations**
Static IMEX daemons	Per ComputeDomain	**Alpha: ComputeDomainDaemonSet**

Upcoming release: graduate the core, add vGPU, and scale ComputeDomains. Plans can change until release.

Migrating from the Device Plugin | Same request

TODAY · v0.5.0

KubeCon CloudNativeCon North America 2026

POD

```
# 1 · Pod spec stays the same
spec:
  containers:
  - name: train
    resources:
      limits:
        nvidia.com/gpu: 1
```

DEVICECLASS

```
# 2 · Shipped in the Helm chart
kind: DeviceClass
metadata:
  name: gpu.nvidia.com
spec:
  extendedResourceName: nvidia.com/gpu
```

POD STATUS

```
# 3 · Scheduler writes
status:
  extendedResourceClaimStatus:
    resourceClaimName: train-gpu-x7k9m
    requestMappings:
    - containerName: train
      resourceName: nvidia.com/gpu
```

No workload changes

Existing charts and manifests keep working. The scheduler creates the ResourceClaim.

Kubernetes

DRAExtendedResource: beta and on by default in 1.36.

One or the other

Disable the device plugin on nodes that run the DRA driver.

Move to DRA first, then adopt claims, CEL and sharing when you're ready.

Migration Status | Device plugin requests, mapped to DRA

TODAY · v0.5.0

KubeCon CloudNativeCon North America 2026

GPU shape	Device plugin request	Through extended resources	Native DRA
Full GPU	`nvidia.com/gpu: 1`	✓ **Works: chart DeviceClass**	`gpu.nvidia.com`
MIG slice	`nvidia.com/mig-1g.12gb: 1`	✓ **Works: custom DeviceClass per MIG profile needed**	`mig.nvidia.com + profile`
Time-sliced GPU	`nvidia.com/gpu.shared + Timeslicing config`	**Not Supported, Need a claim**	`GpuConfig TimeSlicing`
MPS	`nvidia.com/gpu.shared + MPS config`	**Not Supported, Need a claim**	`GpuConfig MPS`

MIG via extended resources · demo/specs/extended-resources

```yaml
kind: DeviceClass
metadata:
  name: mig-1g-12gb.gpu.nvidia.com
spec:
  extendedResourceName: nvidia.com/mig-1g.12gb
```

Tell us what's missing

Which device plugin setups block your move?
Time-slicing and MPS need an explicit claim today.

Why DRA Now? | On by default in 1.36

KUBERNETES 1.36

KubeCon CloudNativeCon North America 2026

DRA core APIs are GA since 1.34. In 1.36 the features below are beta and on by default, so no cluster flags are needed.

BETA · ON

DRAPartitionableDevices

Dynamic MIG: slices created on demand

BETA · ON

DRAConsumableCapacity

Consumable Shares: many claims share a GPU

BETA · ON

DRADeviceTaints

Health checks: unhealthy GPUs are skipped

BETA · ON

DRAExtendedResource

Migration: nvidia.com/gpu served by DRA

BETA · ON

DRADeviceBindingConditions

ComputeDomain channel admission

BETA · ON

ResourceHealthStatus

GPU health in pod status (KEP-4680)

Upgrade to 1.36 and every driver feature in this talk works without touching cluster feature gates.

Related DRA Talks

- All Things DRA in One Place: DRA Example Driver Deep Dive
- Two Drivers, One Topology: GPU:NIC Rail Alignment for RDMA VMs with KubeVirt DRA
- Time to Learn New Tricks to Improve Performance on DRA-Managed Devices
- The Latest in GPU, TPU, NIC and Other Device Support - WG Device Management
- Why Are You Still Wasting Whole GPUs? Hardware-Level Sharing, Scheduled with DRA
- GPU Virtualization: First Setup and Understanding the Device Plugin / DRA
- Time to Learn New Tricks to Improve Performance on DRA-Managed Devices
- Here be DRAgons: Bringing native zero-copy networking to Pods with Cilium and netkit

KubeCon CloudNativeCon
North America 2026

Thank You

Questions?

Deep Dive Into the DRA Driver for NVIDIA GPUs: What Works Today, What's Coming and Why It Matters

