DRA Reserved Capacity - Design Doc for Review

Field	Value
Enhancement tracking	kubernetes/enhancements#5719
SIGs / WG	SIG Node, SIG Scheduling, WG Device Management
Authors	Vishal Anarase @John Belamaric
Reviewers requested	@pohly @klueska @mortent jbelamaric@google.com
Target release	1.38
Status	Draft for discussion

Summary

DRA drivers can publish **reserved** (hold-back) device capacity so spare accelerators stay visible for capacity planning and opportunistic use, while normal workloads cannot consume them unless they explicitly tolerate reservation, using existing device taints and tolerations rather than a parallel exclusion mechanism.

Candidate release note: DRA now allows vendors to model reserved capacity, which can be used to manage failures in collections of accelerators or other devices.

Problem statement

Accelerator fabrics (multi-GPU nodes, multi-node training slices, rack-scale NPUs) often keep **spare devices** to absorb failures without draining an entire job. Today, DRA drivers typically either:

1. **Omit spares** from what they publish to Kubernetes, the scheduler never sees them, so there is no standard way to plan capacity or use spares opportunistically
2. **Publish everything** and rely on ad hoc attributes or custom selectors, no consistent semantics for “this device is hold-back inventory.”

Neither approach gives the ecosystem a clear answer to:

- **Capacity planning**: How many devices are held back vs available for normal workloads
- **Safe oversubscription**: Using spares for batch or preemptible work until repair needs them
- **Repair:** Allocating a spare when a device in a coordinated set fails

Enhancement #5719 asks whether DRA needs special handling for example, modeling reserve resources so they require a **device toleration**, allowing low-priority scheduling unless the spare is needed for repair.

This document proposes a concrete design for that idea and lists open questions for maintainers before a KEP PR is opened.

Goals and non-goals

Goals

- Let drivers mark devices as **reserved** in published device inventory with clear scheduler semantics
- Ensure default workload/device-class consumers **do not** get reserved devices unless they add an appropriate device toleration
- Document patterns for **opportunistic** use (preemptible / low-priority workloads that tolerate reservation)
- Expose reserved capacity in **pool status** where operators plan capacity (builds on ResourcePoolStatus work)
- Gate new behavior behind a dedicated feature gate for Alpha

Non-goals (for initial Alpha)

- Vendor-specific repair orchestration (failover logic stays in drivers / operators)
- Node-level reservation or preemption policies unrelated to DRA devices
- Changing claim-level “reserved for pod” semantics (that is a different concept, who may use an already-allocated claim)
- Soft scheduling preference without a DRA scoring story (future work)
- Automatic preemption of opportunistic workloads when repair starts

Proposed approach (recommended model)

Build on **device taints and tolerations** (DRA device taints, beta in recent releases) instead of inventing a second exclusion system.

Layer 1: Explicit `reserved` on devices (driver-facing)

Add an optional boolean on each device in a resource slice:

- `reserved: true` means the device is hold-back / spare capacity, not intended for default production claims.
- `reserved: false` **or unset** existing behavior.

This is **declarative metadata** about the device's role. It does not mean "currently unallocated." A reserved device may still be allocated to an opportunistic claim.

Layer 2: Well-known taint (scheduling)

When `reserved: true` and the feature gate is enabled, the API server **defaults** a well-known device taint if the driver did not already set an equivalent one:

Field	Field
Key	`resource.kubernetes.io/reserved`
Value	`true`
Effect	`NoSchedule`

Drivers may set this taint themselves instead of using the `reserved` field, validation should accept either if semantics match.

Scheduling behavior: unchanged core algorithm devices with `NoSchedule` (or `NoExecute`) are not allocated unless the claim tolerates them. Reserved devices are **not hidden** from the allocator, they are filtered like any other tainted device.

Layer 3: Tolerations (workload-facing)

Consumer	Suggested toleration
Opportunistic / batch / low-priority	`key=resource.kubernetes.io/reserved,operator=Exists`, `effect=NoSchedule`
Repair (explicit)	Same key, or vendor-specific value if repair must be distinguished from batch

Production device classes and templates for normal workloads must not include these tolerations.

Layer 4: Pool status (operators)

Extend pool status (ResourcePoolStatusRequest / PoolStatus) with something like:

- **`ReservedDevices`** - Count of reserved devices in the pool that are not allocated.
- **`UnavailableDevices`** - Should include unallocated reserved devices for default consumers (today this may be stubbed at zero in the controller).

User stories

Platform capacity planning
As a cluster admin, I want pool status to show how many devices are reserved for failure replacement so I can size training pools and avoid surprise shortages

Opportunistic use of spares
As a batch platform team, I want preemptible jobs to use reserved GPUs when primary capacity is full, accepting that repair may evict us when a failure occurs

Multi-node repair
As a driver author, I want to hold back N devices per rack, mark them reserved in DRA, and have repair automation allocate a spare only when claims opt in via toleration without hiding spares from the API entirely.

Example flows (YAML sketches)

Driver publishes a spare

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
spec:
  driver: vendor.example/accelerator
  pool:
    name: rack-7
    generation: 3
  devices:
  - name: gpu-spare-0
    reserved: true
```

Normal claim - no reservation toleration → spare never selected.

Opportunistic claim - includes toleration for `resource.kubernetes.io/reserved` → may allocate spare when useful.

Terminology (avoid confusion)

- Reserved device: Device published for hold-back / spare use
- Primary device: Device in the same pool for normal allocation
- Opportunistic consumer: Claim that tolerates reservation
- Repair consumer: Claim or operator action that needs a spare after failure

Claim “reserved for pod”** | **Different concept** — which pods may use an already-allocated claim; not hardware hold-back |

Dependencies and feature gate

Prerequisites:

- Dynamic Resource Allocation (core DRA)
- DRA device taints (reservation uses taints for scheduling)
- Recommended: DRA resource pool status for operator visibility

Proposed feature gate:

- Name: `DRAReservedCapacity`
- Default: `false` (Alpha)
- Components: API server (defaulting/validation), pool status controller, scheduler largely unchanged if scheduling is taint-driven

Target milestone: TBD with WG, earliest realistic release: 1.38

Implementation outline (after KEP approval)

1. API: `Device.reserved`, defaulting to well-known taint, validation.
2. Pool status controller: `ReservedDevices` and fix `UnavailableDevices` counting when gate on.
3. Tests: API defaulting, allocator with/without toleration, integration/e2e with test driver.
4. Docs: driver guide, platform patterns for opportunistic vs repair.

No change to core allocator matching logic beyond validation/metrics—existing taint matching already applies.

Open questions for reviewers

1. **API shape:** Is `Device.reserved` plus well-known taint defaulting the right split, or should Alpha be **taints-only** with no new field?
2. **Toleration keys:** Should opportunistic and repair use the same toleration key, or distinct values under `resource.kubernetes.io/reserved`?
3. **Repair escalation:** Does Beta need standardized `NoExecute` behavior for repair, or is driver-driven deallocation enough for Alpha?
4. **Gate dependency:** Must `DRADeviceTaints` be GA before this leaves Alpha?
5. **Scope for 1.38:** Is Alpha in 1.38 realistic, or should this stay design-only until a driver commits to a reference implementation?
6. **Overlap with DeviceTaintRule:** Should cluster admins mark spare inventory at scale via taint rules instead of (or in addition to) per-device `reserved`?
7. **Preemption:** Any requirement to integrate with PriorityClass for opportunistic eviction, or explicitly defer?

References

Enhancement issue #5719
DRA device taints (KEP-3063 area)
DRA resource pool status / visibility (KEP-5677 area)