The Future of Kubernetes Node Lifecycle

Lucy Sweet
Dawn Chen

D

Kubernetes should be boring!

Use cases migrated from:

- **2016:** Simple Web Frontends
- **2020:** Stateful Databases & Dev Environments
- **2025:** Standardized GPU Orchestration

The 2025 Wins (What we promised last year):

Pod Level Resources

Advanced sharing models within pods are now standard.

In-Place Pod Resizing

Reliability achieved; VPA no longer requires disruptive restarts.

Node Swap

Workload-specific swap limits providing a safety net for memory-heavy AI.

D

Same Kubernetes, Different Context

WORKLOAD TYPES

- Accelerators
- Batch
- Stateful

The Reality of Modern Workloads

WORKLOAD TYPES

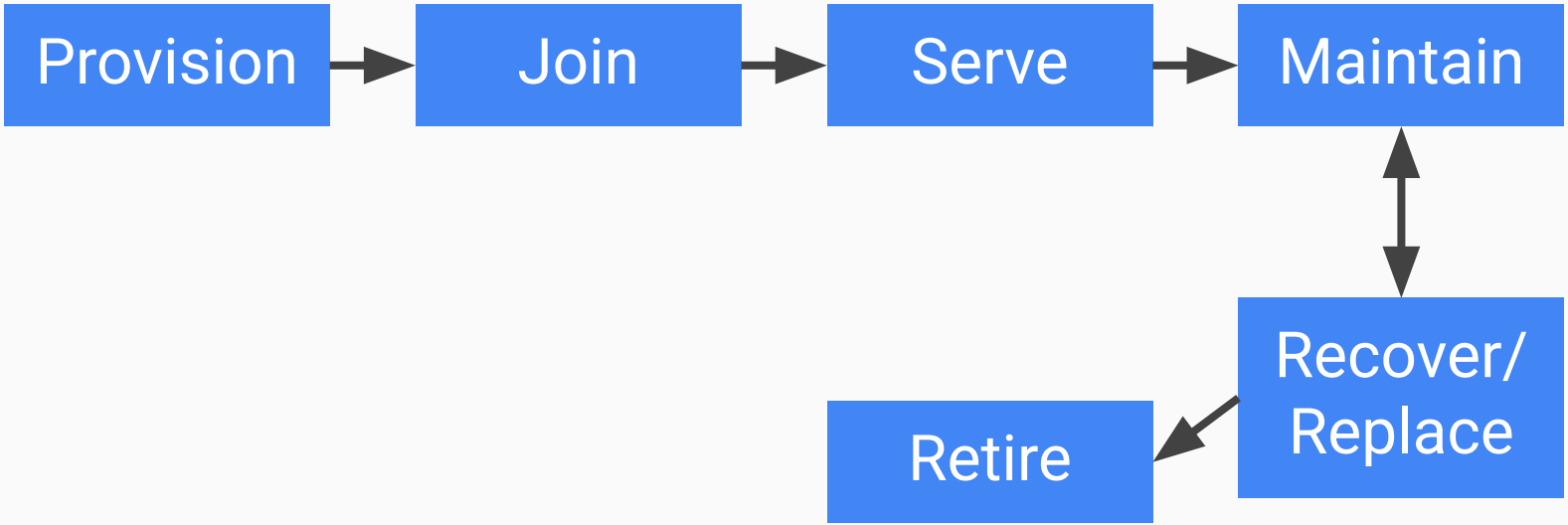
- Accelerators
- Batch
- Stateful

CHALLENGES

- Long Lived
- More Brittle
- Stricter SLOs
- Data Gravity

D

What Do We Mean By “Node Lifecycle"?



Node Challenges

Signals

Readiness vs Serving

Feature Discovery

Actions

Node Maintenance

Safe Replacement

Stateful + Batch Aware Disruption

Coordination

Disruption Planning

Node Observability

Maintenance = Drain

1. Planned Action

Nodes running **AI inference** require routine updates or hardware checks.

Today, there is no nuanced "maintenance" state.

2. The "Drain" Trigger

The system treats planned maintenance exactly like a **node failure**.

Forces an immediate, aggressive **node drain**.

3. Operational Loss

High volume of **underutilized resources** during the draining process.

Stalls workloads unnecessarily before the hardware is even offline.

NodeReady != Ready for everything

1. False Confidence

A "Ready" status from Kubernetes only confirms core kubelet health.

It does **not** guarantee that a node is prepared for specific workload requirements.

2. Silent Dependencies

Helper agents (logging, security, storage) must be functional before serving.

Critical dependencies are often **invisible** to the standard scheduler.

3. Fragmented Tooling

Today, infrastructure teams are forced to build **bespoke versions** of readiness checks.

Leads to maintenance overhead and operational fragility.

Feature Discovery Is Inconsistent

Closing the Information Gap for Specialized Hardware

The "Black Box" Problem

Kubelet reports limited labels, leaving the scheduler blind to deep topology.

Prevents intelligent placement for accelerators.

Label Fragmentation

Inconsistent labels hinder portable scheduling policies:

- `accelerator=nvidia`
- `gpu.present=true`

Stalls infrastructure automation.

The Cost of Inconsistency

Improper placement degrades performance:

- **Noisy Neighbors** on bus
- **Shattered** training runs

2026 SOLUTION Standardized signals via **Node Declared Features** and **Node Readiness Gates**.

D

Application Replica Availability Isn’t Guaranteed

The Coordination Gap

Reconciliation Latency

The "Blackout Period" between node failure detection and pod rescheduling.

Intent Blindness

PDBs can block disruption, but they can't signal intent to allow an app to prepare (e.g., checkpointing).

Uncoordinated Chaos

Hardware failures and OOM-killing happen in silos, forcing the orchestrator to react to a crash rather than coordinate a move.

2026 SOLUTION Moving toward **Managed Intent** with **EvictionRequest** and **Pod Replacement Policies**.

D

2026 Roadmap: What We’re Doing

Building a Coordinated Node Lifecycle

Better Disruption Signaling

Moving beyond binary Ready status with **Node Readiness Gates** and **Pod Disruption Conditions**.

Safer Maintenance Flows

Replacing the "Drain Hammer" with the **EvictionRequest API** to enable workload-aware negotiations.

Stronger Scheduling Signals

Eliminating label fragmentation through **Node Declared Features** to provide a "Standard Language" for hardware.

Better Observability

Closing the "Intent Gap" by making infrastructure changes transparent and predictable for the application.

D

Eviction Request

1. Declaration

Controllers declare control via Pod annotations (class & priority) before eviction begins.

Key Actors: Evictors (requesting move) and Eviction Controllers (managing state).

2. Negotiation

The **EvictionRequest Operator** populates controllers and runs them by priority.

Controllers report progress, expected finish time, and non-cancellable "commit" points.

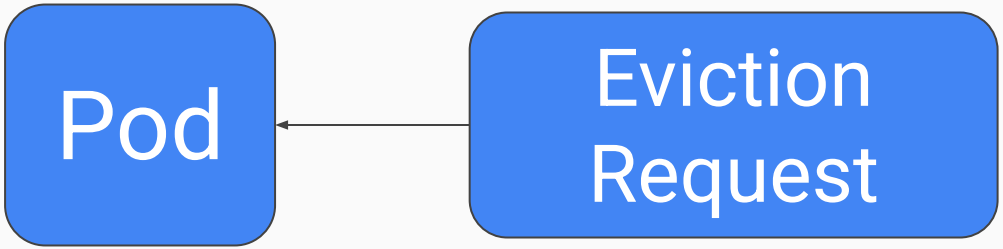
3. Execution

Once all controllers complete, the operator initiates API-driven eviction with exponential backoff.

Ensures workloads prepare (e.g., checkpointing) before termination.

Eviction Request

Eviction Request



Eviction Request

Eviction Request

Eviction Request

Node Declared Features

The Problem: "Label Spaghetti"

- **Version Skew:** Control planes enable features that Kubelets can't execute, creating "Black Hole" scheduling.
- **Manual Mapping:** Fragmented vendor labels (e.g., GKE vs. EKS) require manual pod compatibility tracking.

The Mechanism: node.status.declaredFeatures

- **Authoritative Reporting:** Kubelet automatically reports supported features to the API server.
- **Automatic Inference:** Scheduler matches Pods to capabilities without manual **nodeSelector**.

SYSTEM-GUARANTEED CAPABILITIES: Eliminating "Black Box" scheduling through Kubelet-native feature reporting.

D

Node Readiness Gates

The Signal Fix

Moving from a binary "Ready" to a verified "Ready-to-Serve" status.

Declarative Verification

Defining **ReadinessRules** that must be met before a node joins the cluster.

Dependency Awareness

Ensures GPU drivers, security agents, and data caches are healthy before the first pod lands.

THE SPECTRUM OF READINESS: Ensuring nodes are functionally verified before accepting production workloads.

D

Why this isn’t enough?

Beyond the Single Node

Modern maintenance happens across zones, pools, and entire fleets, not just one node at a time.

Coordination Gaps

Lack of a "unified brain" to coordinate between node maintenance, workload intent, and global capacity.

Workload-First Reality

Users care about Application Availability. Draining a node might still violate a fleet-wide SLO.

The Next Horizon

Moving toward Managed Intent with "all-or-nothing" semantics for multi-node jobs (e.g., Ray clusters).

MOVING FROM NODE-CENTRIC TO WORKLOAD-CENTRIC: Bridging individual node management to unified fleet-level orchestration.

D

Users Care About Their Applications

Abstracting Infrastructure

Nodes are an implementation detail of Kubernetes that shouldn't burden the end user.

Application-First Design

The primary goal is functional software; infrastructure should serve the workload, not define it.

The Goal: Nodes as a Detail

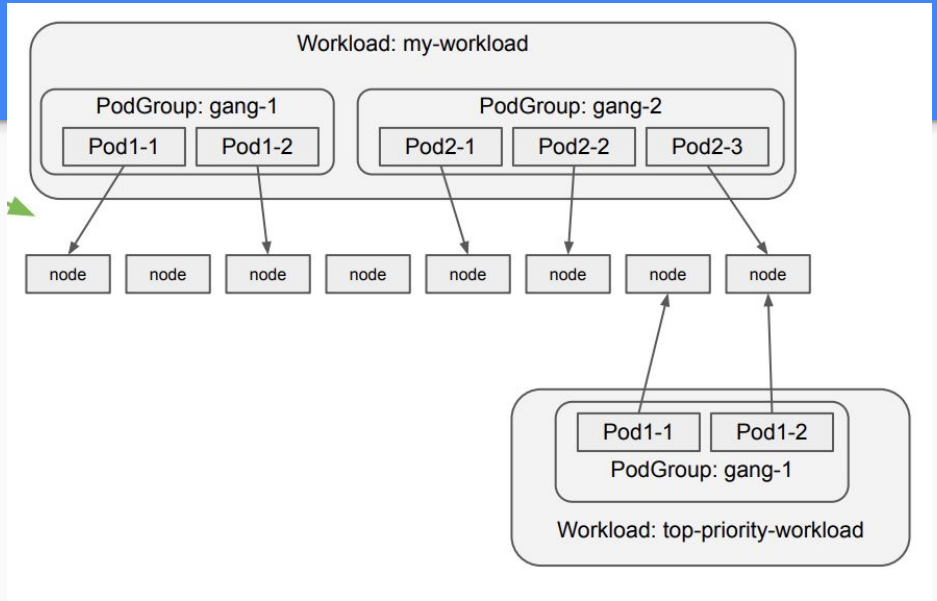
Moving toward a future where "Nodes" are fully abstracted from the user experience.

Ongoing Efforts

Workload API

DRA

Node Group



Dynamic Resource Allocation

ⓘ **FEATURE STATE:** Kubernetes v1.35 [stable](enabled by default)

This page describes *dynamic resource allocation (DRA)* in Kubernetes.


Hello 1+1
Helium
Forum

Thank you!

Questions?