Volume 11, Issue 5, May – 2026 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1495

A Reference Architecture for Scalable, Reliable, and GPU-Optimized AI Model Execution on Kubernetes

Shekar Rao Lakavath[1]

[1]Principal Engineer

[1]Providence Health and Services, USA.

Publication Date: 2026/06/11

Abstract: AI inference workloads stress infrastructure through bursty demand, long model initialization times, heterogeneous GPU requirements, and strict latency constraints. Kubernetes provides a programmable control plane for deployment, scaling, scheduling, and recovery, making it a strong substrate for production inference. This paper introduces KARB (Kubernetes AI Reliability Blueprint), a reference architecture and decision framework that connects service-level objectives (SLOs) to GPU-aware placement, multi-layer autoscaling, and governance controls. We explain GPU scheduling via device plugins, node labels, taints/tolerations, and optional GPU sharing strategies (MIG or time-slicing). We also present a unified scaling approach combining Horizontal Pod Autoscaler (HPA), KEDA event-driven scaling, and node autoscaling to control latency while minimizing GPU cost. Finally, we outline Kubernetes-native MLOps integration using Kubeflow Pipelines and KServe and provide reproducible templates and checklists suitable for enterprise platform engineering teams.

Keywords: *Kubernetes, AI Inference, GPU Scheduling, Autoscaling, KEDA, HPA, Node Autoscaling, Kubeflow, KServe, MLOps, SRE, Platform Engineering, and Reliability.*

How to Cite: Shekar Rao Lakavath (2026) A Reference Architecture for Scalable, Reliable, and GPU-Optimized AI Model Execution on Kubernetes. *International Journal of Innovative Science and Research Technology*, 11(5), 4018-4022. https://doi.org/10.38124/ijisrt/26may1495

I. INTRODUCTION

Modern AI inference systems are rapidly scaling in complexity, driven by real-time applications and large GPU-dependent models, but they introduce significant operational challenges such as unpredictable traffic, resource fragmentation, and strict latency requirements. While Kubernetes provides a robust control plane for deployment and scaling, existing production environments often rely on disconnected approaches to autoscaling, GPU utilization, and service observability, resulting in inefficiencies and reliability risks. This gap highlights the need for a cohesive architecture that aligns infrastructure behavior with service-level outcomes. The Kubernetes AI Reliability Blueprint (KARB) is proposed as a structured framework to address these challenges by enabling coordinated control across scaling, scheduling, and governance in production AI platforms.

Statement of Original Contribution (Novelty)

This paper is not a generic overview. It contributes three practitioner-oriented constructs designed to be reusable across organizations:

- KARB (Kubernetes AI Reliability Blueprint): a layered reference architecture for inference that separates ingress, routing, buffering, model execution, and observability, with explicit interfaces and failure domains.
- GPU-Aware Scaling Control Loop (GASCL): a multi-layer scaling model that couples workload scaling (HPA/KEDA) with infrastructure provisioning (node autoscaling), using service-level signals (p95 latency, queue depth, GPU utilization) as first-class metrics.
- Safety & Governance Invariants for AI Serving on Kubernetes: a minimal set of enforceable policies (Pod Security Standards, admission control patterns, RBAC least privilege, and NetworkPolicy boundaries) to reduce blast radius and regressions.

These constructs are packaged with operational templates and a reproducibility section so platform teams can implement, measure, and compare outcomes.

Background: Why Kubernetes Fits AI Inference

Kubernetes supports horizontal scaling through the HorizontalPodAutoscaler control loop, which periodically adjusts replica counts based on observed metrics such as resource utilization or custom metrics. For event-driven

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1495

workloads, KEDA extends Kubernetes scaling by mapping external event sources to HPA-compatible metrics and enabling scale-to-zero patterns. For capacity provisioning, node autoscaling adds nodes when pods are unschedulable and consolidates nodes to optimize cost. For GPUs, Kubernetes uses the device plugin framework to expose accelerators as schedulable resources, and supports placement control using labels, selectors, taints, and tolerations.

➢ *KARB Reference Architecture for Production AI Inference*

KARB partitions the system into five layers with independent scaling and failure isolation: (L1) Edge ingress and authentication, (L2) routing and policy, (L3) buffering/async queue, (L4) model serving and GPU execution, and (L5) observability and SLO enforcement. This separation prevents burst pressure from directly collapsing GPU serving capacity and enables SLO-driven scaling.

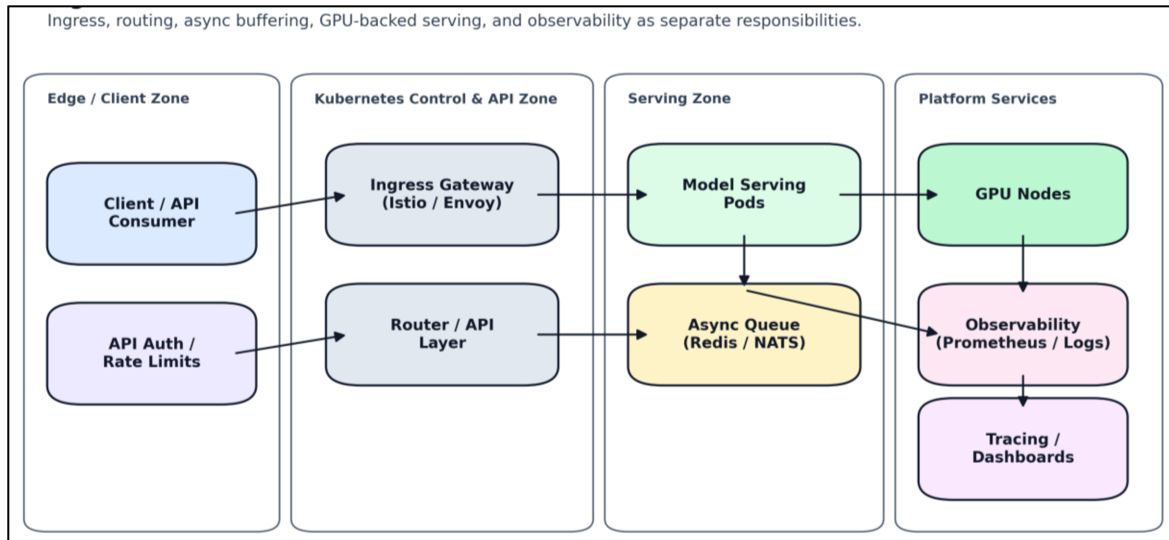

Fig 1 KARB Layered Inference Architecture (Diagram Omitted in Text Export).

Figure 1. Production Kubernetes architecture for AI inference. Ingress, routing, async buffering, GPU-backed serving, and observability are modeled as separate architectural responsibilities.

- L1 Edge: Ingress gateway/service mesh terminates TLS and enforces authentication and rate limits.
- L2 Routing: API layer routes requests to synchronous inference or to an async queue when backpressure is needed.
- L3 Buffering: Queue absorbs bursts and provides explicit queue-depth signals for event-driven scaling.
- L4 Serving: Model server pods run on GPU nodes with explicit accelerator requests and placement constraints.
- L5 Observability: Metrics (latency, saturation, GPU utilization), logs, and tracing drive alerting and autoscaling decisions.

II. AUTOSCALING AS A CONTROL SYSTEM (GASCL)

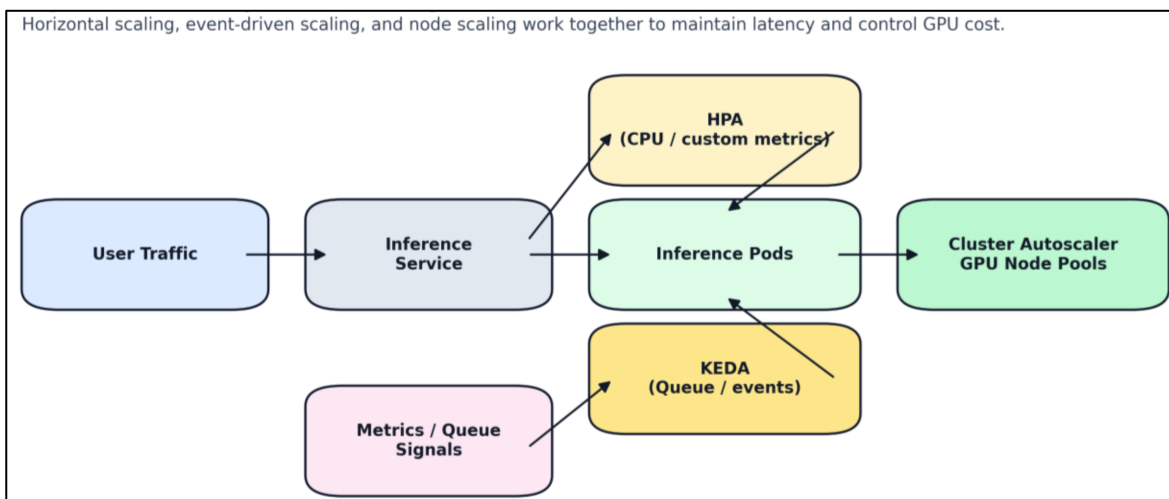

Fig 2 Multi-Layer Autoscaling Model for AI Workloads on Kubernetes. Pod-Level, Event-Driven, and Node-Level Autoscaling Work Together to Preserve Latency while Controlling GPU Cost.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1495

AI inference scaling must be treated as a control system. Pod-level scaling without node capacity is ineffective, while node scaling without rapid pod scheduling does not reduce latency. GASCL defines a coordinated loop: (1) Observe service signals, (2) Decide replica and node targets, (3) Act via HPA/KEDA and node autoscaling, and (4) Stabilize using cooldown windows to prevent oscillation.

➢ *Pod-Level Scaling with HPA (Service-Level Metrics)*

HPA is implemented as a Kubernetes API resource and controller that periodically computes desired replicas from observed metrics; it supports CPU/memory and custom metrics through the metrics APIs. For GPU-bound inference, CPU is often a weak proxy; service metrics such as latency and queue depth are better signals.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
name: inference-hpa
spec:
scaleTargetRef:
apiVersion: apps/v1
kind: Deployment
name: inference-deployment
minReplicas: 2
maxReplicas: 30
metrics:
- type: Pods
pods:
metric:
name: request_latency_p95_ms
target:
type: AverageValue
averageValue: "250"
behavior:
scaleDown:
stabilizationWindowSeconds: 300
policies:
- type: Percent
value: 10
periodSeconds: 60
```

➢ *Event-Driven Scaling with KEDA*

KEDA provides event-driven autoscaling by monitoring external triggers (for example, queue depth or stream lag) and driving scaling through Kubernetes mechanisms, including scale-to-zero where appropriate.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
name: inference-queue-scaler
spec:
scaleTargetRef:
name: inference-deployment
pollingInterval: 15
cooldownPeriod: 300
minReplicaCount: 0
maxReplicaCount: 50
triggers:
- type: prometheus
metadata:
serverAddress: http://prometheus.monitoring:9090
metricName: queue_depth
threshold: "100"
query: sum(queue_depth{service="inference"})
```

➢ *Node Autoscaling (Capacity Provisioning)*

Node autoscaling provisions new nodes when pods cannot be scheduled and consolidates capacity when nodes are underutilized. For GPU cost control, dedicated GPU node pools can scale from zero with placement constraints that ensure only GPU workloads trigger GPU node provisioning.

III. GPU SCHEDULING AND ALLOCATION

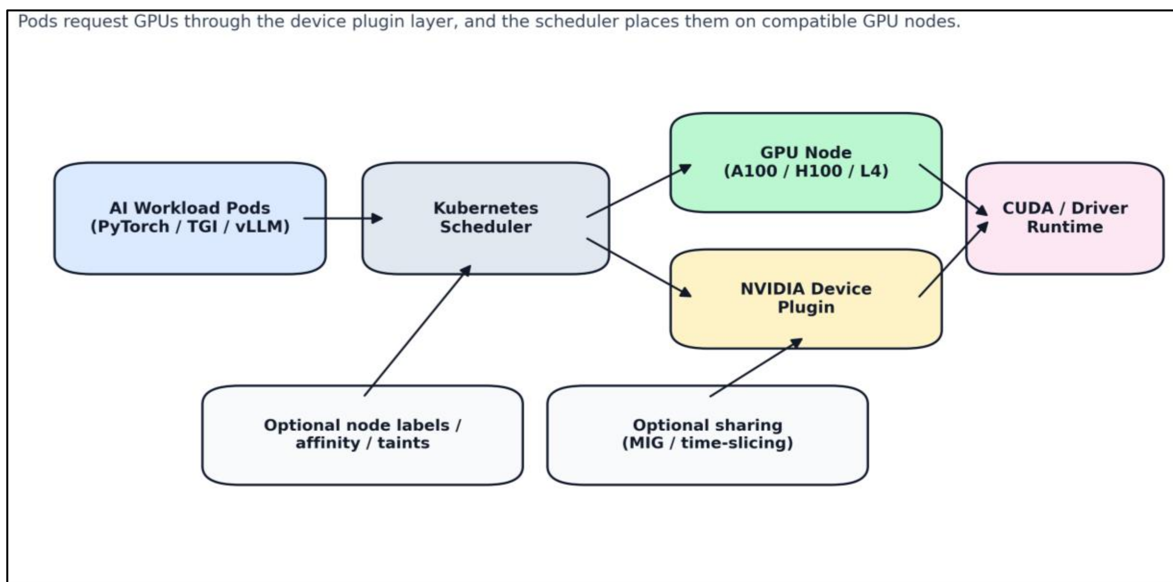

Fig 3 GPU Scheduling and Allocation for AI Workloads in Kubernetes. The Device Plugin Layer Exposes GPUs to the Scheduler, which Places Compatible Workload Pods on GPU-Enabled Nodes.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may1495

Kubernetes schedules GPUs using the device plugin framework, exposing accelerators as extended resources (for example, nvidia.com/gpu). Pods consume GPUs by specifying limits for the GPU resource, and placement can be controlled with labels/selectors and taints/tolerations.

```
apiVersion: v1
kind: Pod
metadata:
name: example-gpu-inference
spec:
tolerations:
- key: "nvidia.com/gpu"
operator: "Exists"
effect: "NoSchedule"
nodeSelector:
accelerator: "nvidia"
containers:
- name: server
image: your-registry/your-inference-image:latest
resources:
limits:
nvidia.com/gpu: 1
cpu: "4"
memory: "16Gi"
```

➢ *GPU Sharing Options (MIG vs Time-Slicing)*

For improved utilization, GPU sharing can be implemented through hardware partitioning (MIG) or software-level time-slicing. MIG provides hardware-level partitioning with memory and fault isolation, while time-slicing enables oversubscription without memory isolation; both are supported through NVIDIA's Kubernetes GPU stack and configuration.

➢ *Kubernetes-Native MLOps: From Pipelines to Serving*

Kubeflow Pipelines provides a Kubernetes-native workflow engine to compose ML workflows as graphs of containerized steps executed on Kubernetes. For serving, KServe provides Kubernetes CRDs that encapsulate model serving lifecycle concerns such as autoscaling, networking, and rollout patterns for predictive and generative inference.

Figure 2. End-to-end lifecycle: data prep → training → validation → registry → deployment → inference (diagram omitted).

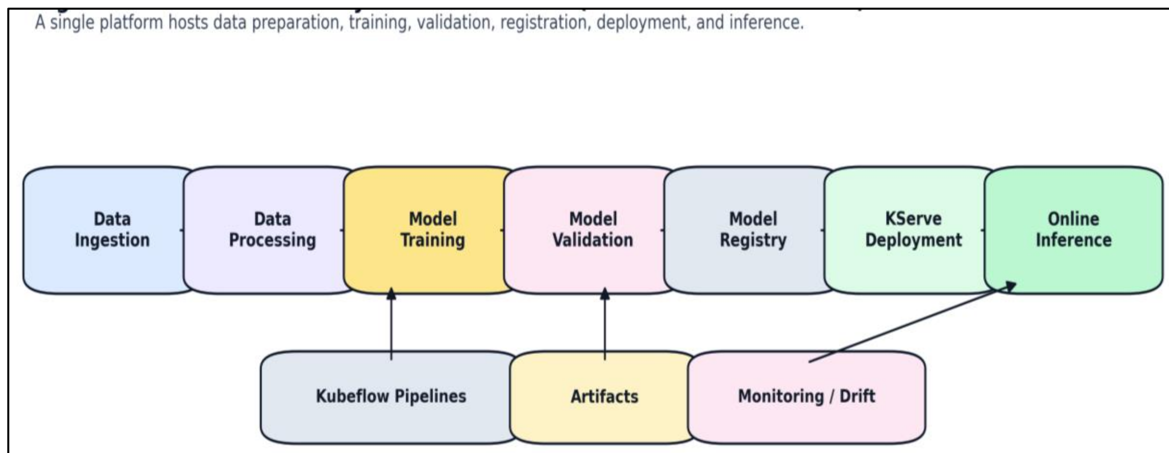

Fig 4 End-to-End AI Lifecycle on Kubernetes. A Kubernetes-Native Platform can Host Data Preparation, Training, Validation, Registration, Deployment, and Online Inference within One Operational Model.

IV. GOVERNANCE AND SAFETY INVARIANTS (PRODUCTION READINESS)

To be evidence-grade, architecture must be enforceable. The following invariants are designed to be measurable and policy-enforced:

- Least-privilege control plane access using RBAC Roles/RoleBindings, with explicit scoping and review cadence.
- Namespace-level pod hardening using Pod Security Standards (Privileged/Baseline/Restricted) with audit→warn→enforce rollout.
- Network segmentation using Kubernetes NetworkPolicies to implement default-deny and explicit allow rules per trust boundary.
- Admission control for safety checks using validating webhooks for guardrails (resource limits, image policy, required labels).

➢ *Example: Default-Deny NetworkPolicy Skeleton*

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: default-deny
namespace: inference
spec:
podSelector: {}
policyTypes:
```

- Ingress
- Egress

➢ *Example: Admission Webhook Strategy (Conceptual)*

Kubernetes supports dynamic admission control through validating and mutating admission webhooks configured at runtime. For inference platforms, validating

webhooks are commonly used to enforce guardrails such as mandatory resource limits and approved runtime classes.

➢ *Reproducibility and Evaluation Plan*

To enable independent validation, this section defines an evaluation method that can be run on any conformant Kubernetes cluster.

- Workload: deploy a GPU-backed inference service with a fixed model and a synthetic traffic generator that produces step and burst patterns.
- Metrics: p50/p95 latency, error rate, queue depth, GPU utilization, cold-start time, and cost proxy (GPU node-hours).
- Treatments: (A) CPU-based HPA only, (B) HPA on service metrics, (C) KEDA queue-driven scaling + HPA stabilization, (D) GASCL full stack with node autoscaling.
- Success criteria: maintain p95 latency under the SLO threshold during bursts while minimizing GPU node-hours; avoid oscillation and prevent scale-induced failures.

All YAML templates in Appendix A can be used as a baseline. Replace metrics queries and labels to match your environment.

➢ *Operational Checklist (Copy/Paste)*

- Define SLOs: p95 latency, error rate, and cold-start budget; select signals for scaling (queue depth, latency, GPU utilization).
- Instrument: expose metrics and ensure a metrics pipeline that supports custom/external metrics.
- GPU enablement: install GPU drivers and a vendor device plugin; label GPU nodes and apply taints for isolation.
- Autoscaling: implement pod autoscaling (HPA/KEDA) and ensure node autoscaling can add compatible GPU nodes.
- Governance: enforce Pod Security Standards at namespace level; apply NetworkPolicies; lock down RBAC for service accounts.
- Rollouts: use gradual rollout (canary) and monitor SLO regression; keep a warm pool for strict latency.
- Cost controls: separate GPU node pools and scale-from-zero where feasible; evaluate MIG/time-slicing for utilization gains.

V. CONCLUSION

Kubernetes enables reliable AI inference when treated as an architecture platform rather than a container launcher. KARB provides a reusable blueprint that separates responsibilities, aligns autoscaling with service-level signals, and makes GPU scheduling and governance explicit. By combining HPA, KEDA, and node autoscaling into a coordinated control loop, teams can meet latency objectives while controlling GPU cost. The included templates and evaluation plan support reproducible adoption and measurable improvement in production AI platforms.

REFERENCES

[1]. Kubernetes Documentation — Horizontal Pod Autoscaling. https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/
[2]. Kubernetes Documentation — Autoscaling Workloads (overview). https://kubernetes.io/docs/concepts/workloads/autoscaling/
[3]. KEDA Documentation — Kubernetes Event-driven Autoscaling. https://keda.sh/
[4]. Microsoft Learn — KEDA add-on for Azure Kubernetes Service. https://learn.microsoft.com/en-us/azure/aks/keda-about
[5]. Kubernetes Documentation — Node Autoscaling. https://kubernetes.io/docs/concepts/cluster-administration/node-autoscaling/
[6]. Kubernetes Documentation — Schedule GPUs (device plugins). https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
[7]. Kubernetes Documentation — Taints and Tolerations. https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/
[8]. NVIDIA Docs — NVIDIA GPU Operator (overview). https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/index.html
[9]. NVIDIA Docs — GPU sharing and time-slicing. https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html
[10]. NVIDIA Docs — MIG Support in Kubernetes. https://docs.nvidia.com/datacenter/cloud-native/kubernetes/latest/index.html
[11]. KServe — Inference platform for Kubernetes. https://kserve.github.io/website/
[12]. Kubeflow Documentation — KServe Introduction. https://www.kubeflow.org/docs/components/kserve/introduction/
[13]. Kubeflow Documentation — Pipelines concept. https://www.kubeflow.org/docs/components/pipelines/concepts/pipeline/
[14]. Kubernetes Documentation — Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/
[15]. Kubernetes Documentation — RBAC Authorization. https://kubernetes.io/docs/reference/access-authn-authz/rbac/
[16]. Kubernetes Documentation — Pod Security Standards. https://kubernetes.io/docs/concepts/security/pod-security-standards/