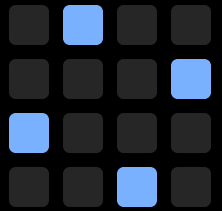
OpenAI

CLOUD NATIVE AI + INFERENCE DAY 2026

Rolling Out LLM Inference Engines on Kubernetes

Capacity, Topology, and Deployment Verification



Zihan (Gavin) Zheng
Inference

Meng Dong
Scheduling

An LLM engine breaks four rollout assumptions

A Deployment rollout starts a new Pod, waits for Ready, removes an old one, and repeats.

A Kubernetes rollout assumes	An LLM inference engine
Any node with room will do	Free GPUs can still have no room for one replica
A new Pod is Ready in seconds	A new replica takes tens of minutes to become Ready
There is spare capacity to surge into	Almost no spare GPUs, even off-peak
A bad version is cheap to roll back	Rolling back takes as long as rolling out

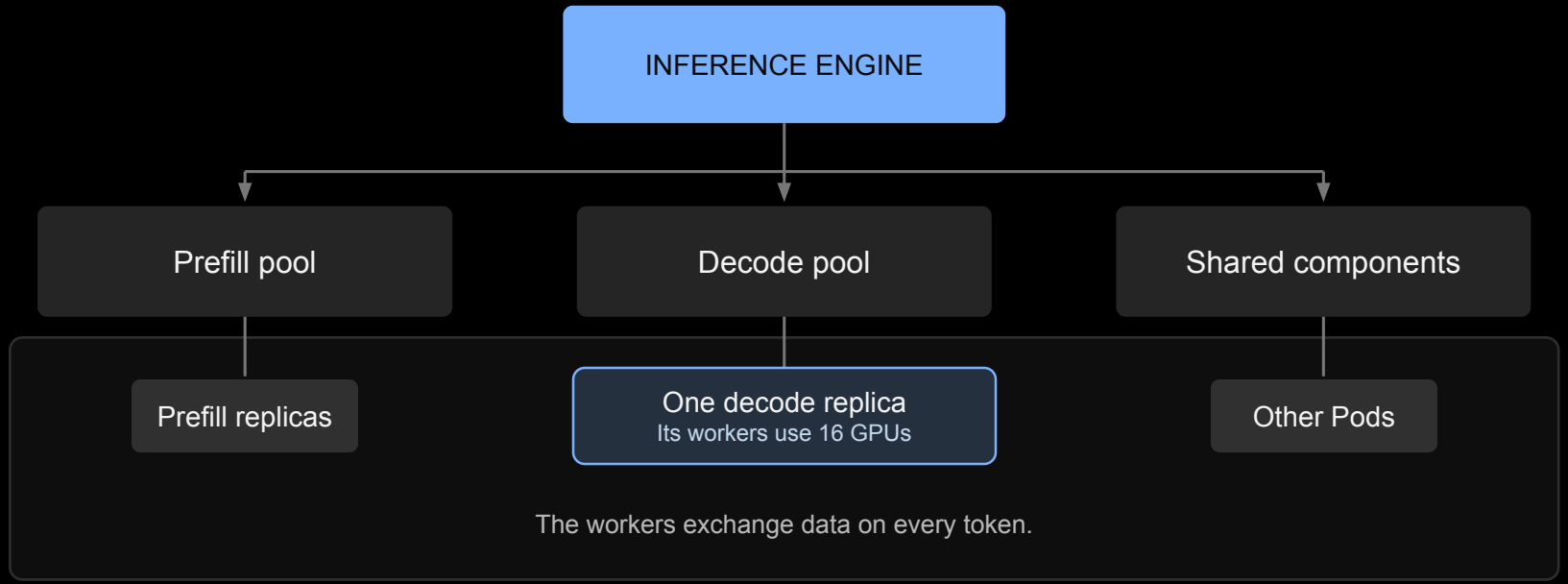
Today: placement, startup, and spare capacity first. Then how we verify each release.

Why GPU rollouts are hard

Placement depends on topology, startup is slow, and spare capacity is scarce.

Placement: a replica’s GPUs must share one NVLink domain

Kubernetes assumes any node with room will do. Here, prefill and decode each run several replicas.



Any free GPUs won’t do. They have to be in the same NVLink domain.

Startup: a new replica takes tens of minutes to start

Kubernetes assumes a new Pod is Ready in seconds. Most of our startup is weights, CUDA graphs, and warmup.

01.
Placed
Scheduler

Capacity admitted → Physical fit found → Workers placed

02.
Starting
Runtime

Containers launch → Weights load → Workers connect

03.
Ready
Engine

Capture CUDA graphs → Warm up → Pass health checks

If this took seconds, we could replace replicas in place and the rollout would be fast.

Spare capacity: both ways to start a replica cost GPUs

Kubernetes assumes there is spare capacity to surge into. We have almost none, even off-peak.

Use spare capacity

Start the new replica on spare GPUs

Keep the old replicas serving

Needs enough free GPUs in one domain

Make room first

Stop one old replica if enough stay healthy

Start the new replica on its GPUs

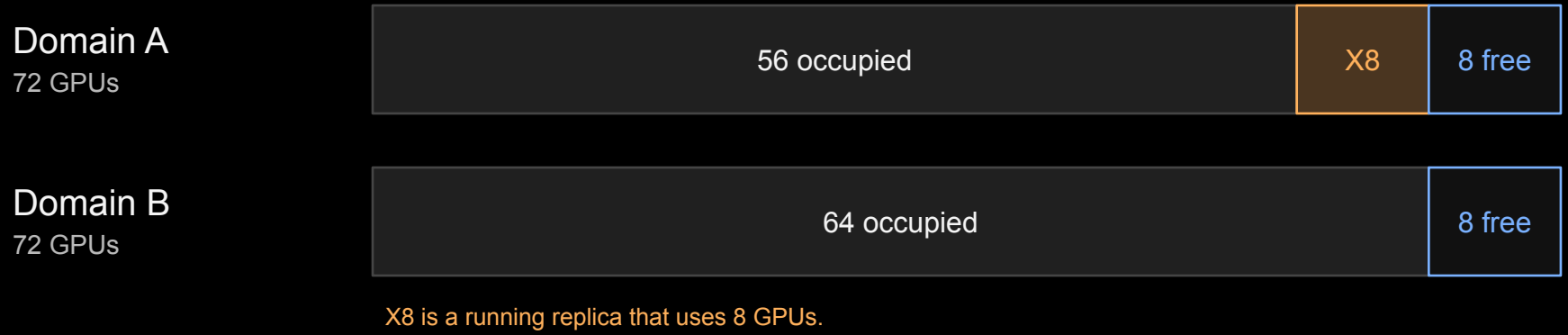
One fewer replica serves during startup

The Scheduling Decision

Can we make room for the new replica
while the cluster keeps serving requests?

16 free GPUs. No room for the new replica.

In this example, each NVLink domain has 72 GPUs. The new replica needs 16 in one domain.



New replica
16 GPUs in one domain

→

8 + 8 GPUs free across two domains

0 complete placements

X8 would fit in the eight free GPUs in B.

The planner checks health before choosing removals

It checks physical fit and the combined effect of proposed removals in one pass.

With a separate health controller

Observe → Classify health → Publish health → Re-read health → Choose action

Health can change between publishing and reading.

Our scheduler uses one observed engine view

Derive component health → Check fit and all planned removals → Apply allowed actions

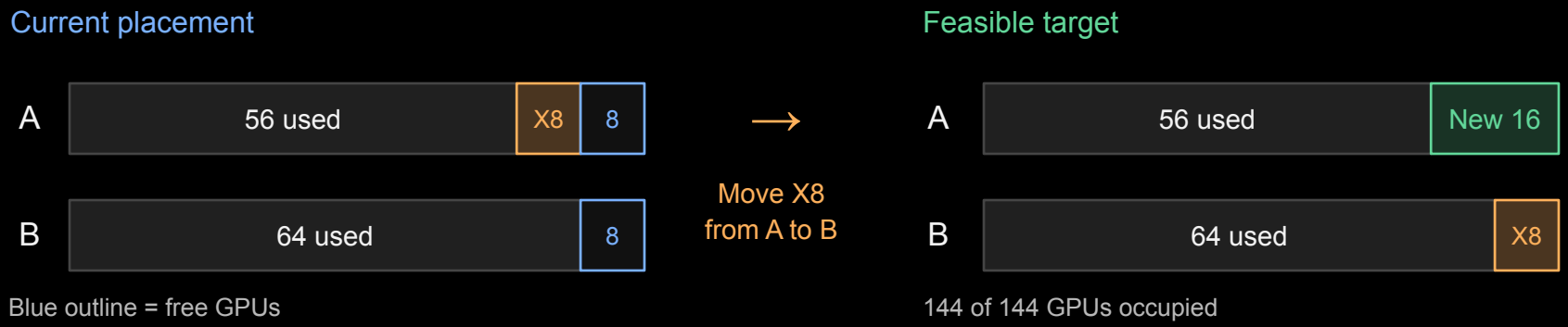
Ordinary repair · X8's pool needs 6 healthy replicas

7 healthy − 1 planned = 6 left → Next healthy removal from this pool waits.

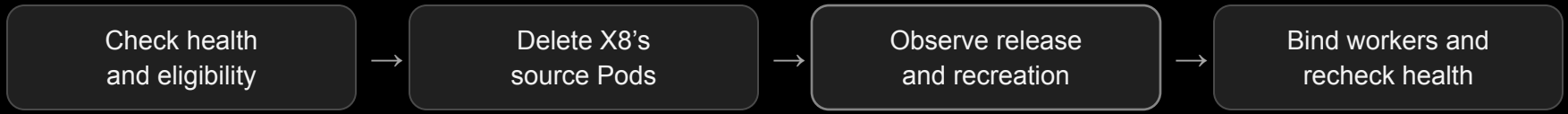
Worker observations and execution remain asynchronous.

Moving X8 makes room for the 16-GPU replica

In this same-priority example, X8 can run in B, its pool can spare one replica, and the new replica can run in A.



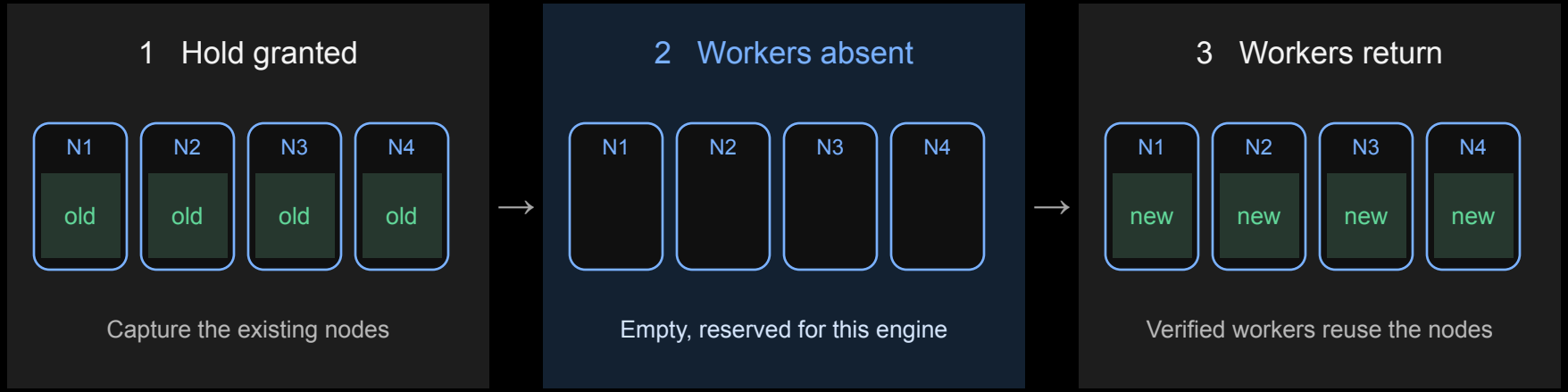
How the scheduler executes the move



Moving X8 costs a restart. Each dependent step waits for an observed state change.

Keep the nodes reserved while workers restart

The controller waits for the scheduler to grant the hold before deleting the workers.



The hold lasts until its deadline, even after the workers return.

Restart benefit

Workers can reuse the same nodes.

Sharing cost

Other engines wait for those nodes.

Four whole nodes shown. Any additional capacity needs a new placement.

Four policies guide the next scheduling decision

The planner checks every proposed move against the same disruption budget.

	Policy	Tradeoff
Place	Look for a fit first Reuse → preferred domain → another eligible domain	Avoids churn. A bounded search can miss a fit.
Repair	Limit same-priority disruption Default per target/domain attempt · up to 4 victim units and 32 GPUs	Protects serving capacity. Placement can take longer.
Preempt	Reclaim eligible lower-priority work Move caps can be relaxed. Health exceptions are explicit.	Recovers priority capacity. Disposable work may stop.
Progress	Give a forming unit time to fit After 3 stalled cycles, release the formation hold and retry in 30 s.	Saves repeated work. Held nodes limit sharing.

Each removal spends part of the budget. Fleet quota and fairness are decided upstream.

How schedulers choose a feasible plan

Simulated victim selection. Each scheduler applies its own placement and policy checks.

Scheduler	Search candidates	Selection rule	Policy checks
In-cluster scheduler	Combinations of victim groups within one topology domain	Lowest-cost feasible plan found within the search budget	Component health, move caps, prefill/default ratio, holds, and destination fit for relocated units
KAI	Ordered victim groups and candidate node sets	First feasible plan from ordered simulations	Topology, queue rights, priority, group minima, and validators. Reallocate victims if possible.
Volcano	Ranked victim-group prefixes across candidate domains	First feasible placement for the incoming group	Topology, queue rights, priority, group minima and surplus. Incoming placement only.
Kueue	Sets of whole Workload victims and quota/topology assignments	First feasible admission, then prune unneeded victims	Topology, resource quota, priority, borrowing rights, and enabled fair-share rules

Our planner checks the combined health impact while it can still choose another set.

In-cluster quality mode · KAI 0.18.6 · Volcano 1.16.0, gang actions · Kueue 0.20.2

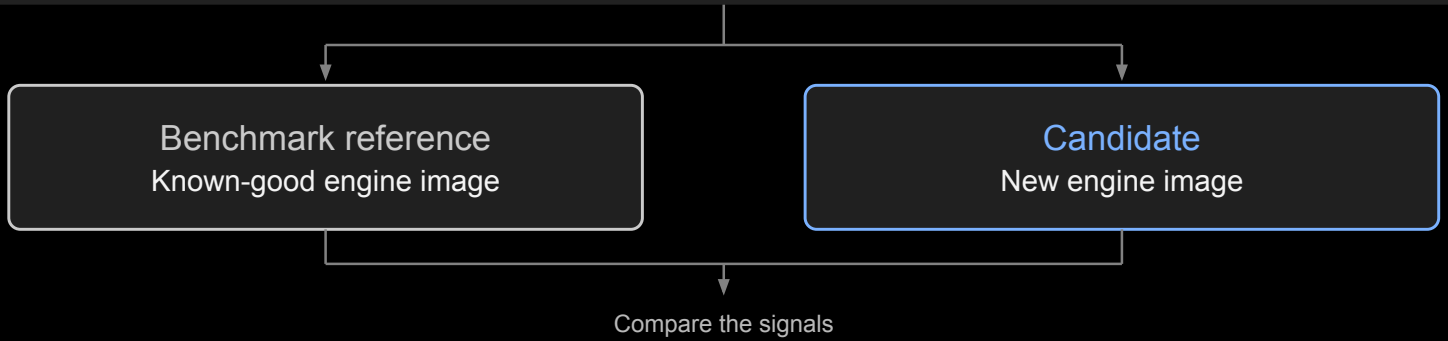
Test before rolling out

Rolling back takes as long as rolling out.

How we verify an engine release candidate

Test both images with the same model, configuration, and request mix at each load level.

Shared test load · Load sweep, peak load, overload and recovery



Startup	Reliability	Latency · p50 and p95	Numerical behavior
Readiness and startup time	Request success and worker restarts	Time to first token (TTFT) Time between tokens (TBT)	Output divergence on matched inputs

Continue
All required checks passed for this configuration.

Hold and investigate
For example, peak-load p95 TTFT exceeds its limit.

Both engines run on the production cluster, so this test needs two full placements.

What we do about each assumption

The four assumptions from the start, with what we do about each.

A Kubernetes rollout assumes	What we do
Any node with room will do	Place whole replicas in one NVLink domain, and move work to open a block
A new Pod is Ready in seconds	Hold the nodes while workers restart
There is spare capacity to surge into	Make room by moving or reclaiming work, without dropping a pool below its minimum
A bad version is cheap to roll back	Test each release candidate against a known-good engine before the rollout

Thank you. Questions?

Where can the planner check serving health?

A shutdown veto can protect workers while the incoming workload keeps waiting.

Stack	Where to check	What to add
Kueue	Custom Job stop and preemption selection	Delay the stop safely. Choose other victims during selection.
KAI	Full-scenario validators for preempt and reclaim	Count all selected removals. Cover each enabled disruption path.
Volcano	UnifiedEvictableFn filters the candidate pool	Build a safe pool or validate the final set of victims.

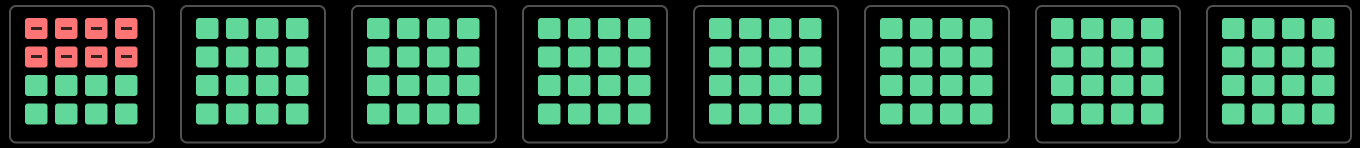
The Eviction API checks PodDisruptionBudgets. Direct Pod deletion bypasses that check.

Gang-aware eviction · KAI by Mar 2025 · Volcano v1.15, Jun 2026 (opt-in Alpha)

Failure location changes how many units can fit

Each unit needs 16 healthy hosts in one domain. A dash marks a failed host.

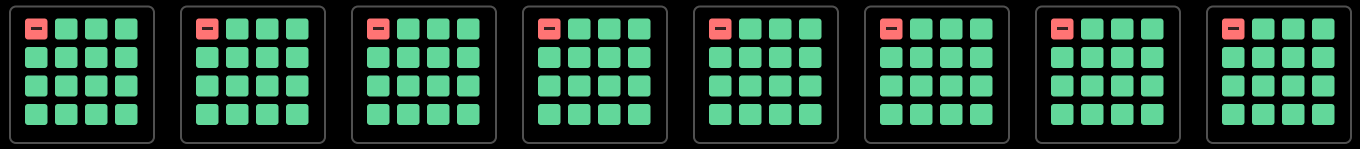
Eight failures in one domain



7 complete domains

120 healthy hosts

One failure in each domain



0 complete domains

120 healthy hosts

A reserved domain may sit idle. It can host this unit only while all 16 hosts are healthy.

What does each node hold preserve?

One holds the nodes an engine already uses. The other helps assemble a new placement.

	Rollout hold	Formation hold
Purpose	Keep the engine’s nodes while workers restart	Keep the nodes found so far for the pending unit
Ownership	Fixed node set for verified workers of this engine	The planner grows the set as it finds eligible nodes
Lifetime	Until the current deadline, with bounded renewal if enabled	After 3 stalled cycles, release the hold and retry in 30 s

When either hold ends, running workers stay in place.

Rollout renewal is off by default. Success stops renewal. The current hold lasts to its deadline.

The failed checkpoint tells you who acts next

Identify the last observed state before choosing a recovery action.

Observed blocker	Decision owner	Next action
No domain fits the whole unit	Capacity / scheduler	Wait for a fit or plan a safe move.
Workers placed, one peer missing	Engine / runtime	Inspect worker startup and connectivity.
Candidate exceeds a test limit	Deployment gate	Hold and inspect the failed check.

Advance when the required evidence is available.

Safe rollouts need three kinds of proof

Compatible capacity. A healthy engine. A verified candidate.

