KubeCon CloudNativeCon India 2026

#KubeCon #CloudNativeCon

# Why We Ditched Kube-proxy

One job per browser — and the connection waits until one is free.

*Routing 10M daily browser sessions on the EndpointSlice API.*

**Rajat Khanna**

Senior Tech Lead · CommerceIQ

# What we run

**8–16M**
**pages scraped / day**
*(10M typical)*

**1,000+**
**retailers, daily**

**~8K**
**concurrent browser pods**
*live count varies with queue*

**100%**
**of workers on spot / preemptible**
*proxy + control plane on a stable pool*

A pod serves **one session at a time**, but is reused across many — recycled only when HPA scales down or spot reclaims its node.

# Why this platform exists

PLATFORM v1

**Browsers coupled to business logic**

Scaled on pub/sub events — workers pulled jobs off queues. No service routing anywhere near the hot path.

PLATFORM v2 (browser-farm)

**Browser automation isolated as a service**

Request in → rendered page out. Clean API; business logic stays out of the browser fleet.

**The trade: queue-pull became request-dispatch.**

And dispatching to browsers isn't load-balancing — each request needs a **specific free pod, held until one exists**. That's the routing requirement v2 created on day one — the one a Service can't express.

# The requirement we had to route

*A request doesn't want a load-balancer. It wants one specific, free browser.*

**EXCLUSIVE** one job per browser at a time. A browser mid-job is not a shared backend — no second caller, ever.

**HELD** the incoming connection waits until a browser is free. None free? Start one and keep holding.

**A Kubernetes Service expresses none of these. Everything else in this talk follows from that.**

# How kube-proxy + Services work

**Normally: kube-proxy load-balances each connection across all the ready pods. Any ready replica will do.**


## It can't hold.

A Service answers now or never — there is no "wait until a browser is free." Our caller often must wait, sometimes for one that does not exist yet.

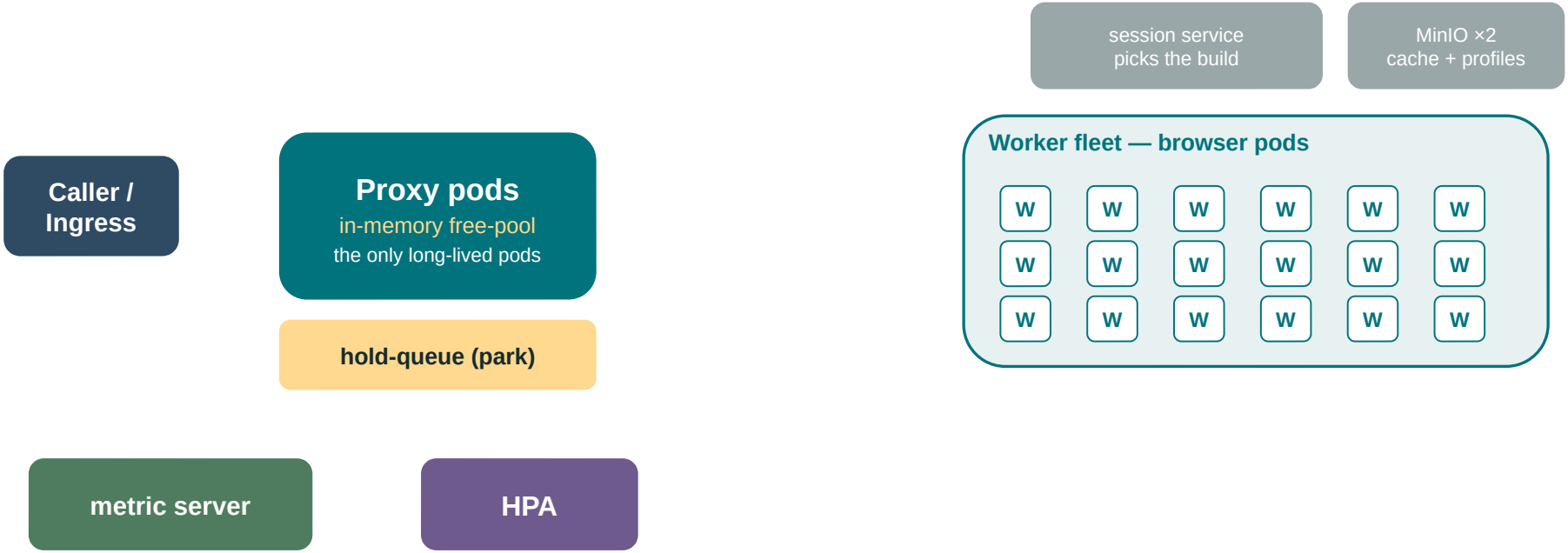
## It can't lease.

It spreads connections across every ready browser. We need one free browser, given to one request, untouchable until the job is done.

**Same EndpointSlice data kube-proxy reads — we just hold + lease.** kube-proxy isn't broken; a Service is the wrong shape for this one hop.

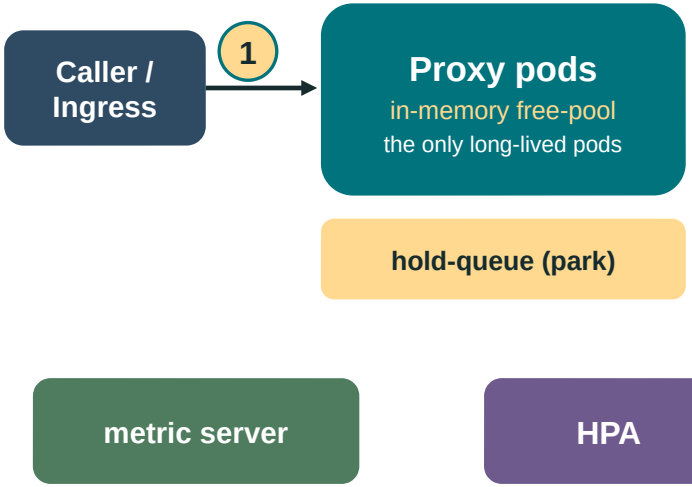
**"Free" flips on every job — so availability lives in the proxy's memory, not in the dataplane.**

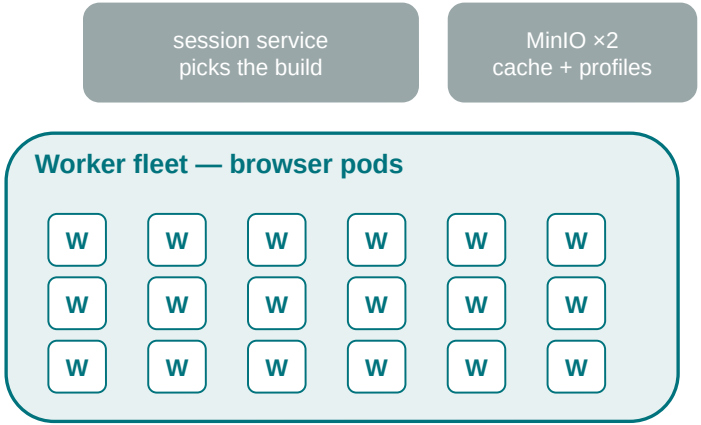
# browser-farm: the whole machine




We scale on the queue, not CPU · workers run on spot

# browser-farm: the whole machine

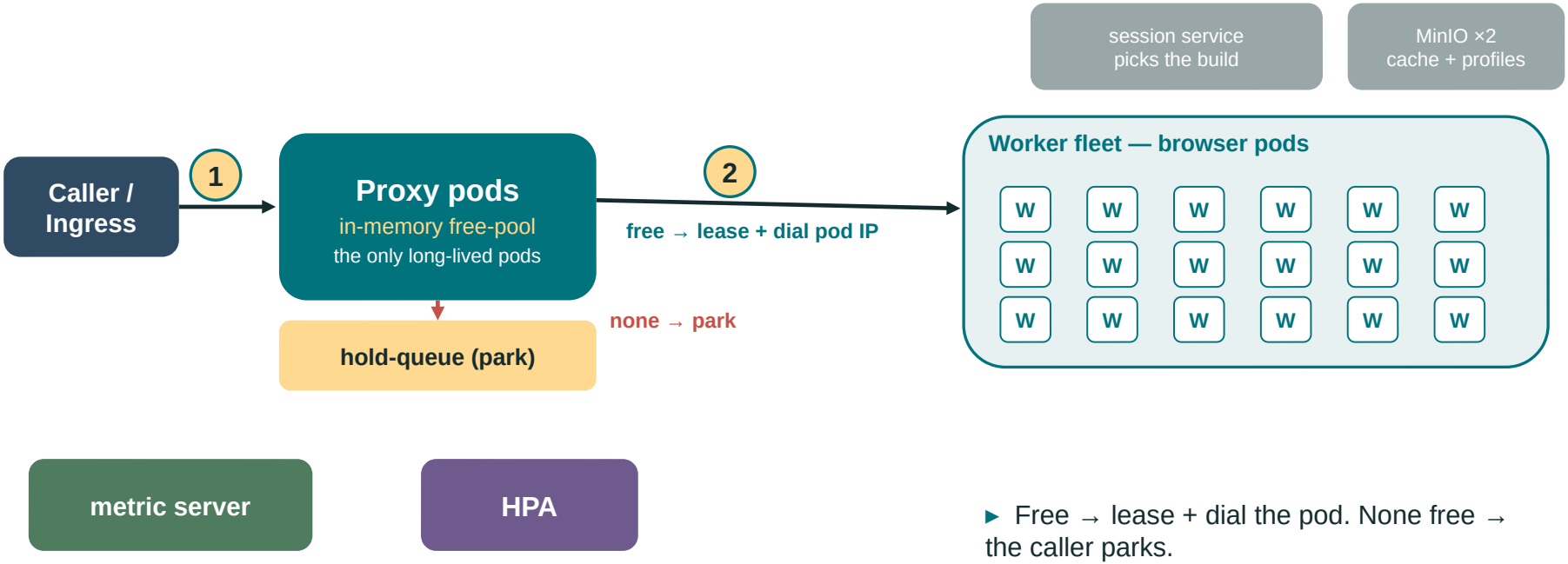



- A request arrives — it needs its own browser.

We scale on the queue, not CPU · workers run on spot

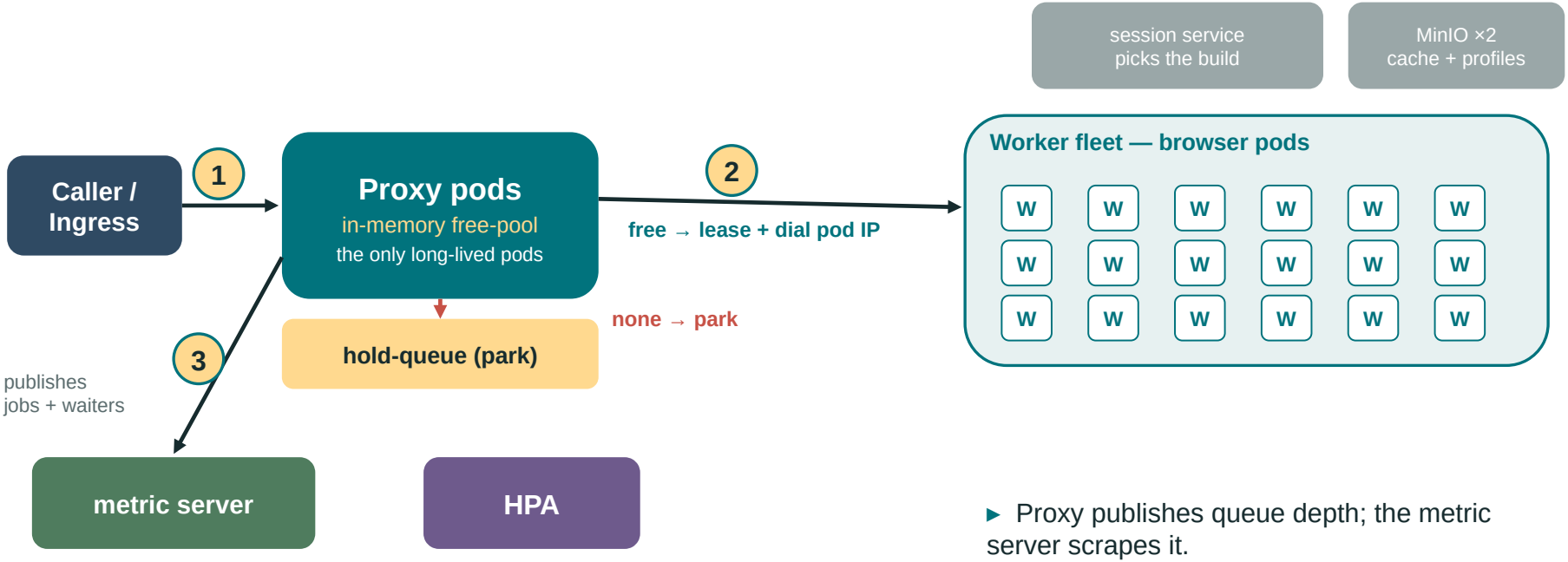
# browser-farm: the whole machine




► Free → lease + dial the pod. None free → the caller parks.

We scale on the queue, not CPU · workers run on spot

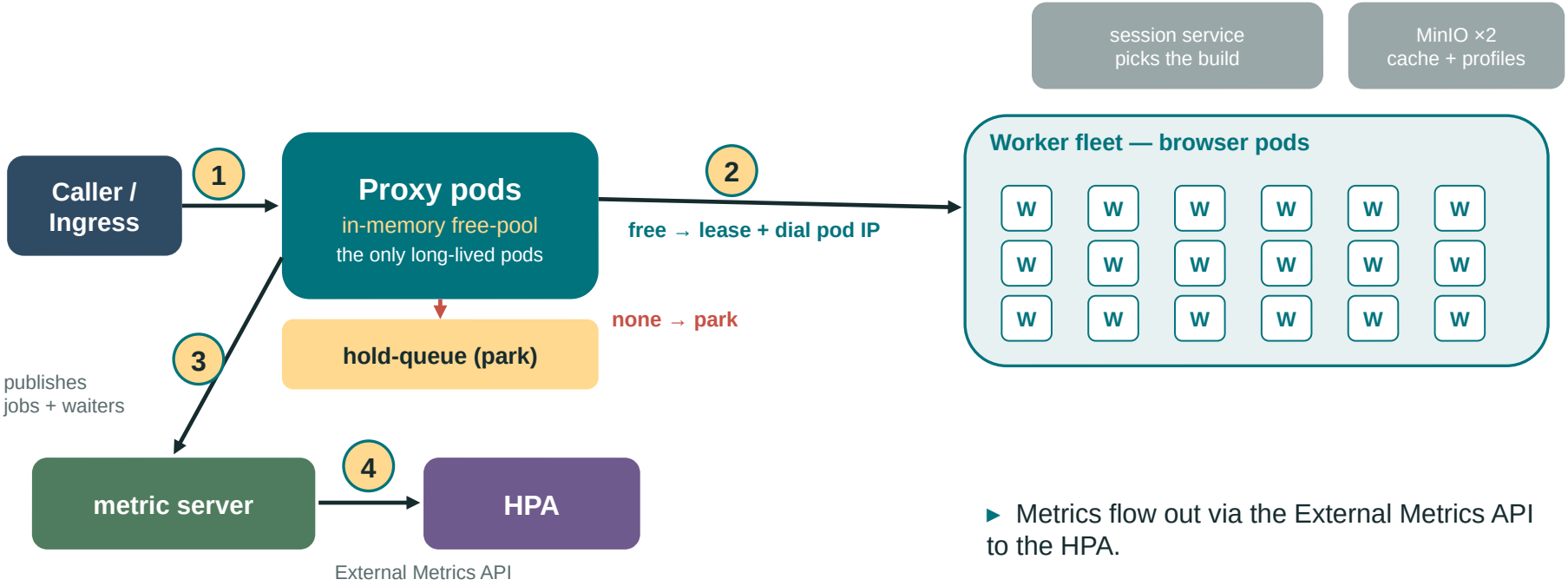
# browser-farm: the whole machine




- Proxy publishes queue depth; the metric server scrapes it.

We scale on the queue, not CPU · workers run on spot

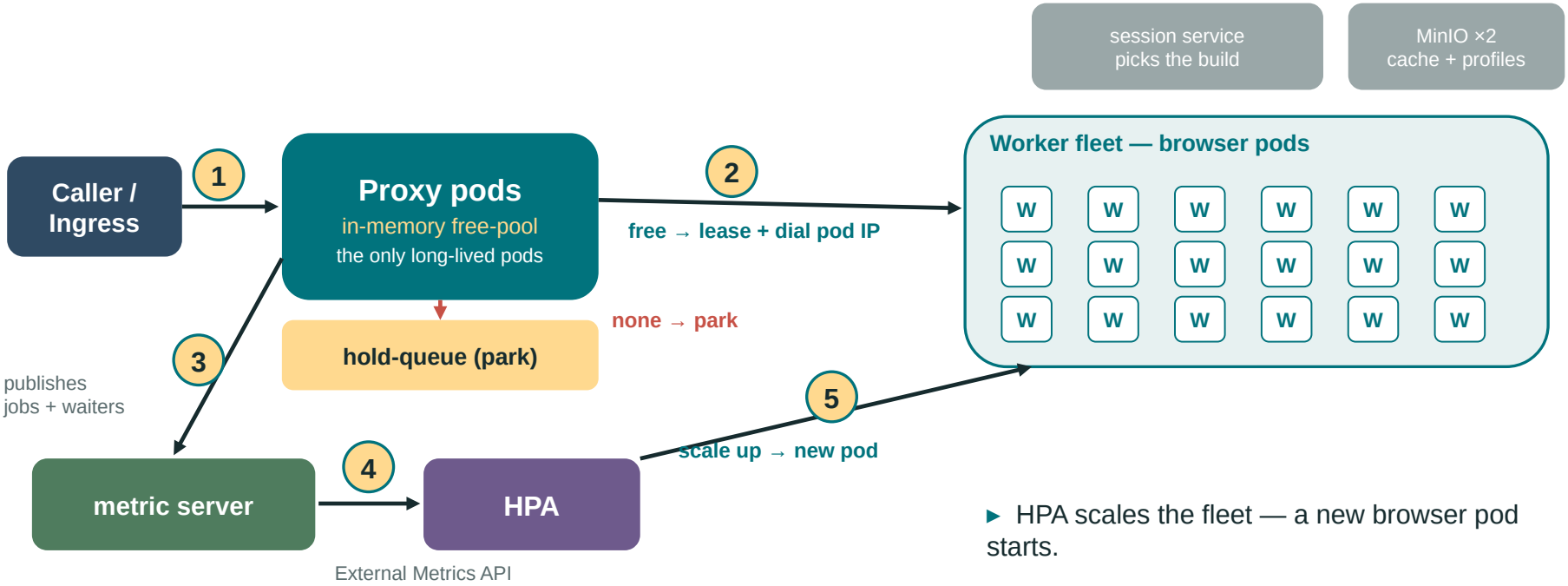
# browser-farm: the whole machine




- Metrics flow out via the External Metrics API to the HPA.

**We scale on the queue, not CPU · workers run on spot**

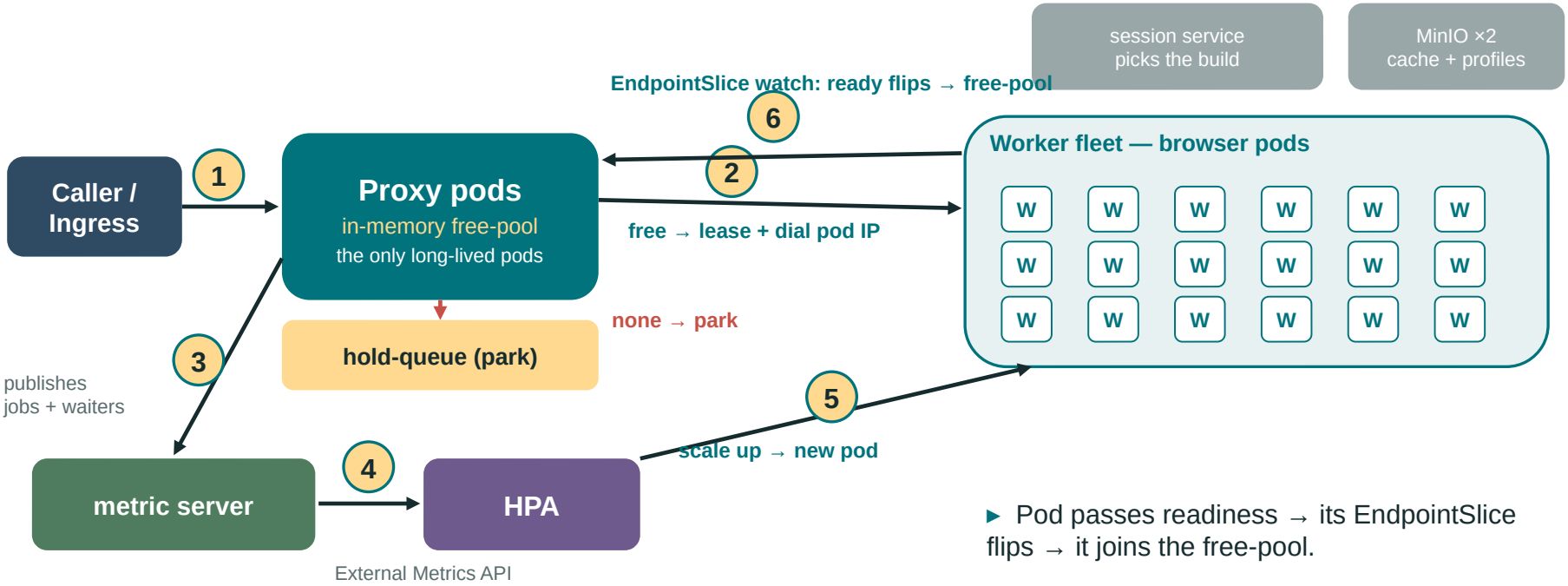
# browser-farm: the whole machine




- HPA scales the fleet — a new browser pod starts.

**We scale on the queue, not CPU · workers run on spot**

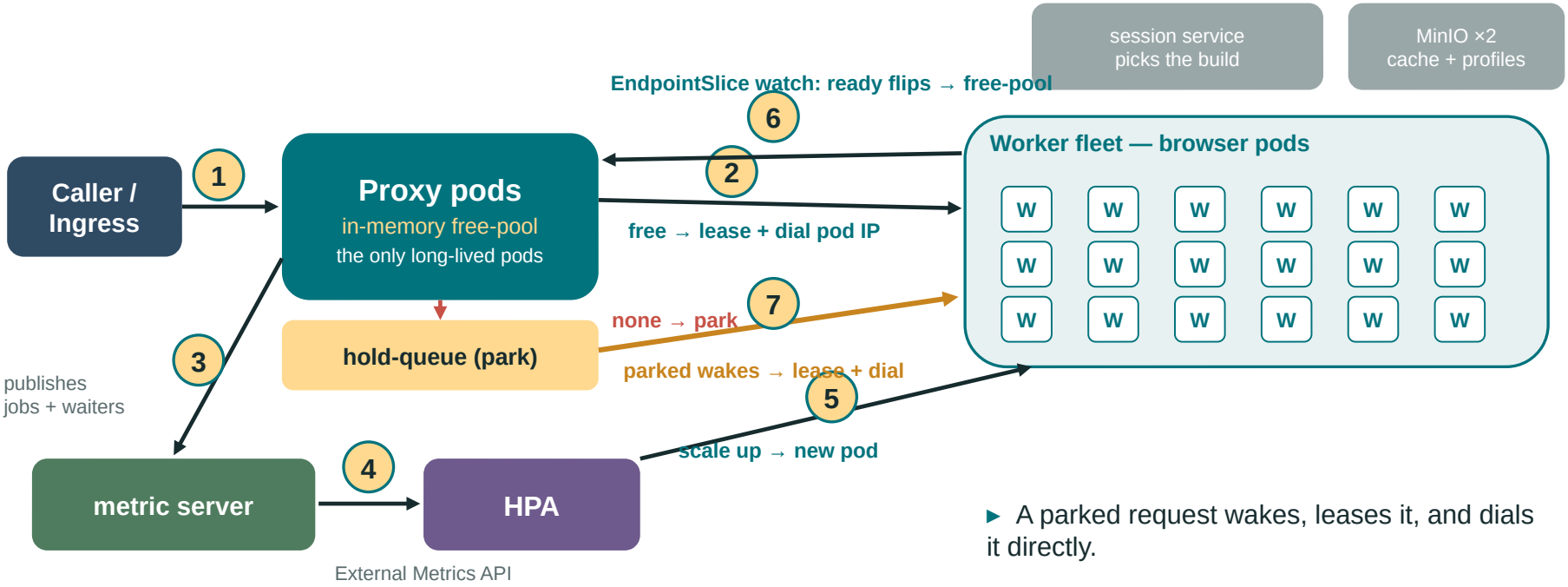
# browser-farm: the whole machine




► Pod passes readiness → its EndpointSlice flips → it joins the free-pool.

**We scale on the queue, not CPU · workers run on spot**

# browser-farm: the whole machine




► A parked request wakes, leases it, and dials it directly.

**We scale on the queue, not CPU · workers run on spot**

# The sharp edge

**Readiness is ours to define.** We make it mean “free for a job” — so it flips every job, and we only ever target a browser that is genuinely free.

**That leaves exactly one rough edge — and it comes from the watch, not from readiness.**

## Browsers that vanish

The free-pool is built from an EndpointSlice watch — always a half-step behind. A browser that just went busy or got reclaimed can still sit in our cached list, so a dial can land on the wrong pod.

## The fix

A shutting-down pod drops from the pool. A failed dial → set that pod aside a couple seconds → re-queue to the next free browser.

Invisible — it happens before the session starts, so the caller never sees it.

**Mid-session death is different: that one job is lost and retried fresh upstream — blast radius of one. We don’t fake a resume.**


# DEMO

Minikube · the same watch → free-pool → lease → direct-dial design as production, run live

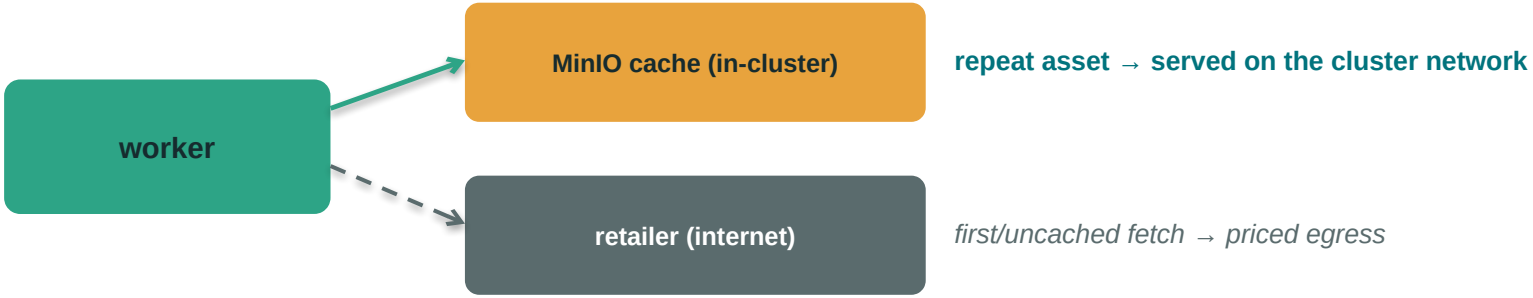
Act 1 — a request arrives with no free pod: the connection is held, a pod spins up, the request lands on that exact pod.

Act 2 — under churn: leases flip occupancy live; a stale endpoint is caught and re-queued invisibly; a pod lost mid-session fails that one job — blast radius of one; kill the watcher and it recovers in seconds.

**Code & README** → github.com/legalimpurity/kubecon-endpointslice-demo

# MinIO, twice: cache + profile store

10M pages a day re-fetch the same JS bundles, fonts, and images from the same retail sites — over and over, across the priced public internet.




- An S3-compatible cache sits next to the workers; a repeat asset is served in-cluster and never crosses the internet again.
- Locality is the whole point — a managed bucket would re-introduce the egress you're trying to kill.
- **The bigger the fleet, the higher the repeat rate: the cache earns more at scale, not less.**
- It also stores Chrome profiles — warm cookies + fingerprints outlive ephemeral pods, so recycled pods don't look ephemeral.

**The savings are real and material — the mechanism is the point.**

# What we'd tell you

1. Match the primitive to the problem. A Service load-balances; we needed an exclusive, availability-aware lease.
2. The primitives are public. EndpointSlice is the same API kube-proxy reads. You can read it too.
3. Bypass surgically. One hot path on the EndpointSlice API; everything else stays on stock kube-proxy.
4. “Ready” is not ready. Design for stale endpoints and cold starts from day one.
5. No fancy tooling — just deep Kubernetes. No mesh, no eBPF rewrite, no new CRDs.

KubeCon CloudNativeCon India 2026

# Thank you

Demo code → github.com/legalimpurity/kubecon-endpointslice-demo

Rajat Khanna · Senior Tech Lead, CommerceIQ

*Slides on Sched · session feedback genuinely helps*

Connect on LinkedIn


# Q&A

*questions, and the why-not-X discussion, welcome*