

# Orchestrating, Discovering, and Autoscaling Distributed Multi-Agent AI Systems Natively in K8s

Derek Wang · Intuit

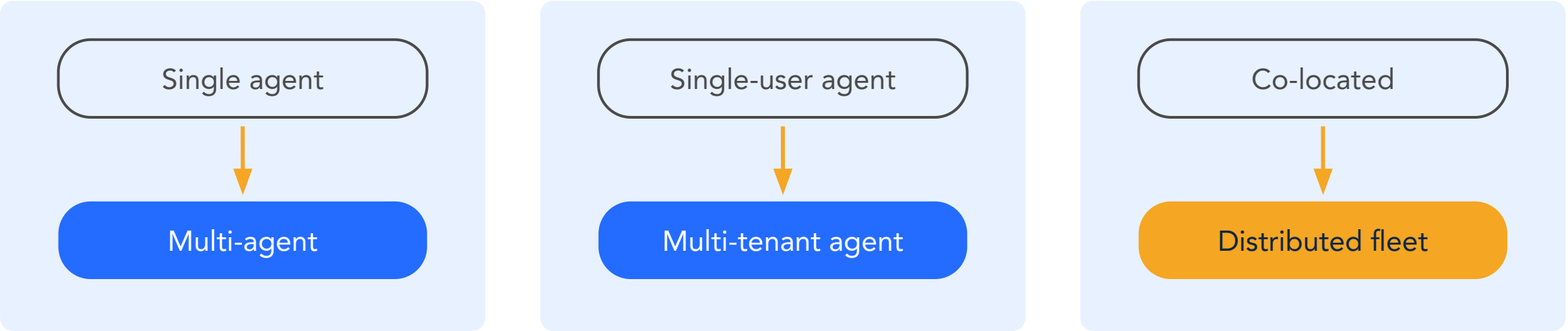
Agentics Day: MCP + Agents · November 2026

AGENDA

# What we'll cover

1. **Setting the stage**
   Distributed agent services, different kinds of agentic workload
2. **Problems**
   What every team re-solves: orchestrate, discover, scale
3. **Kynomesh**
   Orchestration, discovery, autoscaling and safe rollouts
4. **Demo**
   A travel planner on Kubernetes, with drift reload and external agents
5. **Takeaways**

# Agents are becoming distributed services




## At its core, an agent is a distributed service powered by an LLM.

It deserves the same operational rigor as any microservice.

# Agents come in two workload types

## Task agents

Like a Kubernetes Job: coding, research, ops runs

| | |
|---|---|
| Trigger | Async: a person, schedule or event |
| Serves | One request, one principal |
| Lifecycle | Runs to completion, then exits |
| Duration | Minutes to hours; nobody waits |
| Scales by | Number of runs in flight |

## Service agents

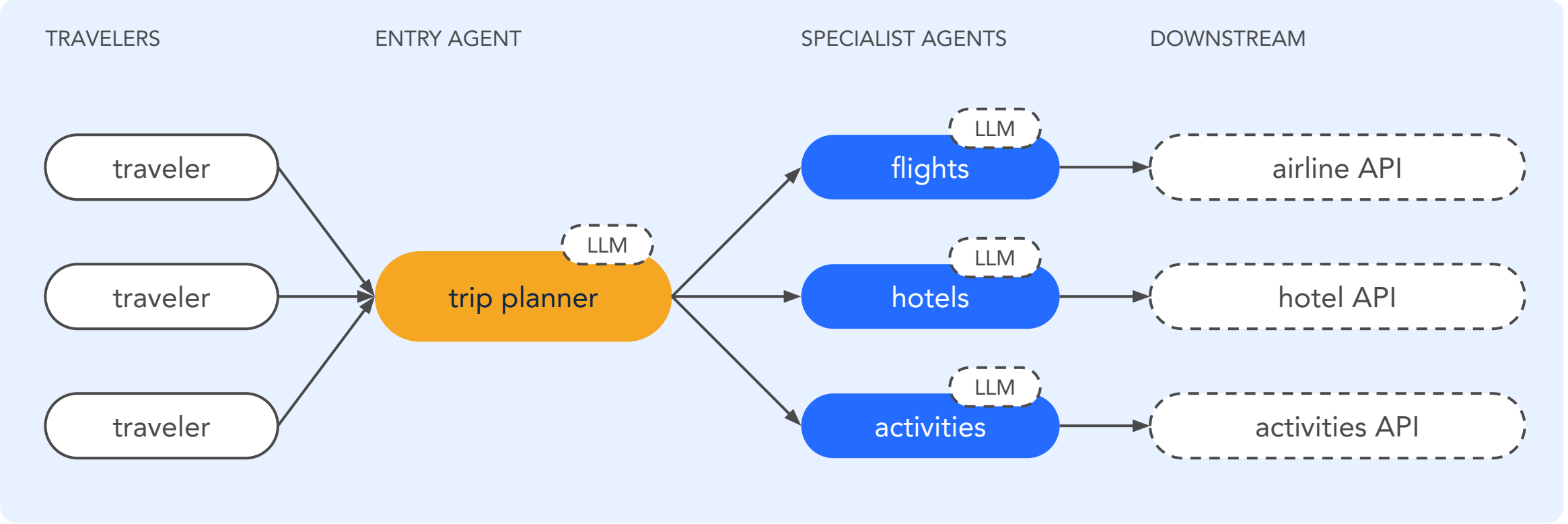
Like a Deployment: assistants, copilots, A2A peers

| | |
|---|---|
| Trigger | Sync request and response |
| Serves | Many concurrent requests and tenants |
| Lifecycle | Always on; never completes |
| Duration | Seconds per request; a caller waits |
| Scales by | Concurrent requests per replica |

Most agent tooling targets the left. Kynomesh is built for the right.

A REAL EXAMPLE

# Example: a trip-planning service




## Many travelers, four LLM-backed agents, hard limits downstream.

Each agent is its own deployment, owned by its own team, possibly in its own language.

THE PROBLEM

# Every team re-solves the same three problems

01

## Orchestrate

How do agents coordinate without hand-wiring each one?

A trip planner calls flight, hotel and activities agents, wired by copy-pasted URLs.

02

## Discover

How do agents find peers and stay in sync as peers change?

The hotel agent gains a loyalty skill. The planner keeps its stale AgentCard.

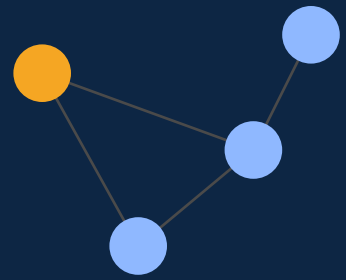
03

## Scale

How do we scale when CPU looks idle under full load?

Holiday rush: agents wait on the LLM, CPU stays at 10%, bookings queue.

None of this is agent logic. It's plumbing, and every team lays it again for every project.



# Kynomesh

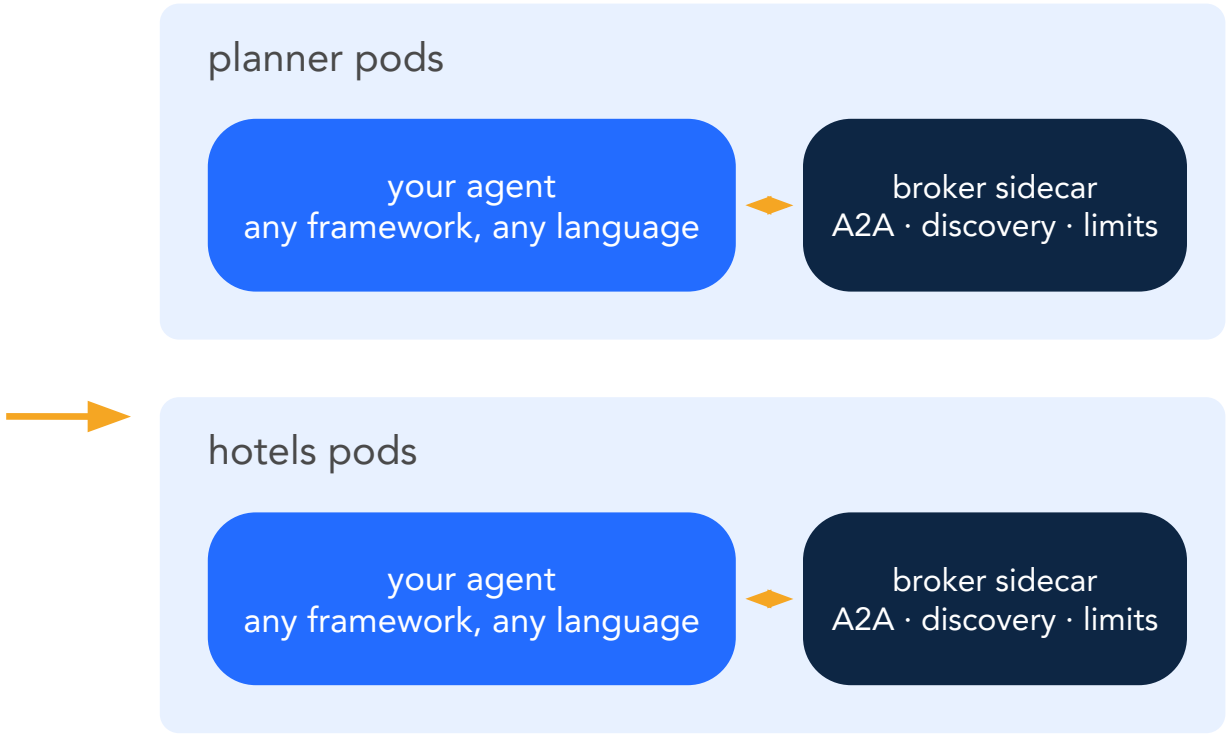
Distributed multi-agent orchestration, native to Kubernetes

Open source · github.com/kynoproj/kynomesh

# Declare the agent set. Kubernetes runs it.

DECLARE · AgentSet: the topology

```
kind: AgentSet
metadata:
  name: trip-planner
spec:
  pattern: Supervisor
  entry: planner
  agents:
    - name: planner
      container:
        image: planner:1.4
    - name: hotels
      container:
        image: hotels:2.0
```




Agent code stays plain A2A, in any framework or language. The platform handles placement, discovery, scaling and rollout.

LangGraph · Google ADK · CrewAI · Claude Agent SDK · any language with A2A. Open source, Apache 2.0.

# Your agent is a plain A2A server

WRITE · your agent: plain A2A code

```
from kynomesh import server

class HotelsExecutor(AgentExecutor):
    async def execute(self, context, queue): ...

card = AgentCard(name="hotels", ...)

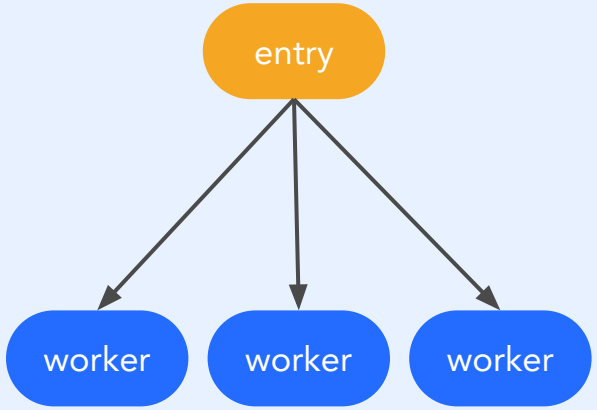
await server.start(HotelsExecutor(), card)
```

- Implement the A2A executor and describe the agent with an AgentCard
- start() picks the listeners and mounts the transports your card advertises
- It registers the agent for discovery; the broker sidecar fronts it

Any language: the SDK is a convenience, not a requirement. Any A2A server runs on Kynomesh. Python and Go SDKs today.

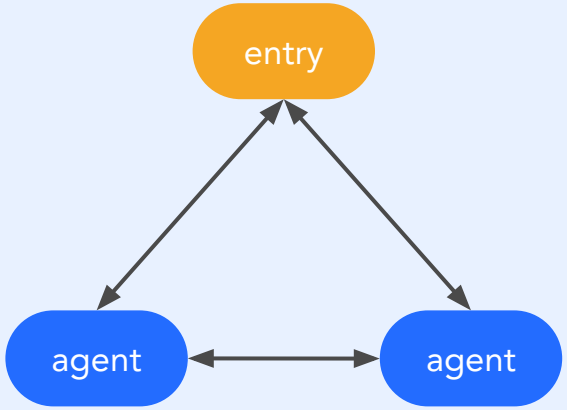
# Name the pattern; the platform wires the peers

## Supervisor




Orchestrator delegates to workers

## Handoff




Any agent hands off to any other

## Sequential




Each stage feeds the next

External agents join as just a name and a URL.

# Agents find peers by name and adapt

WRITE · your agent: plain A2A code

```
from kynomesh import client

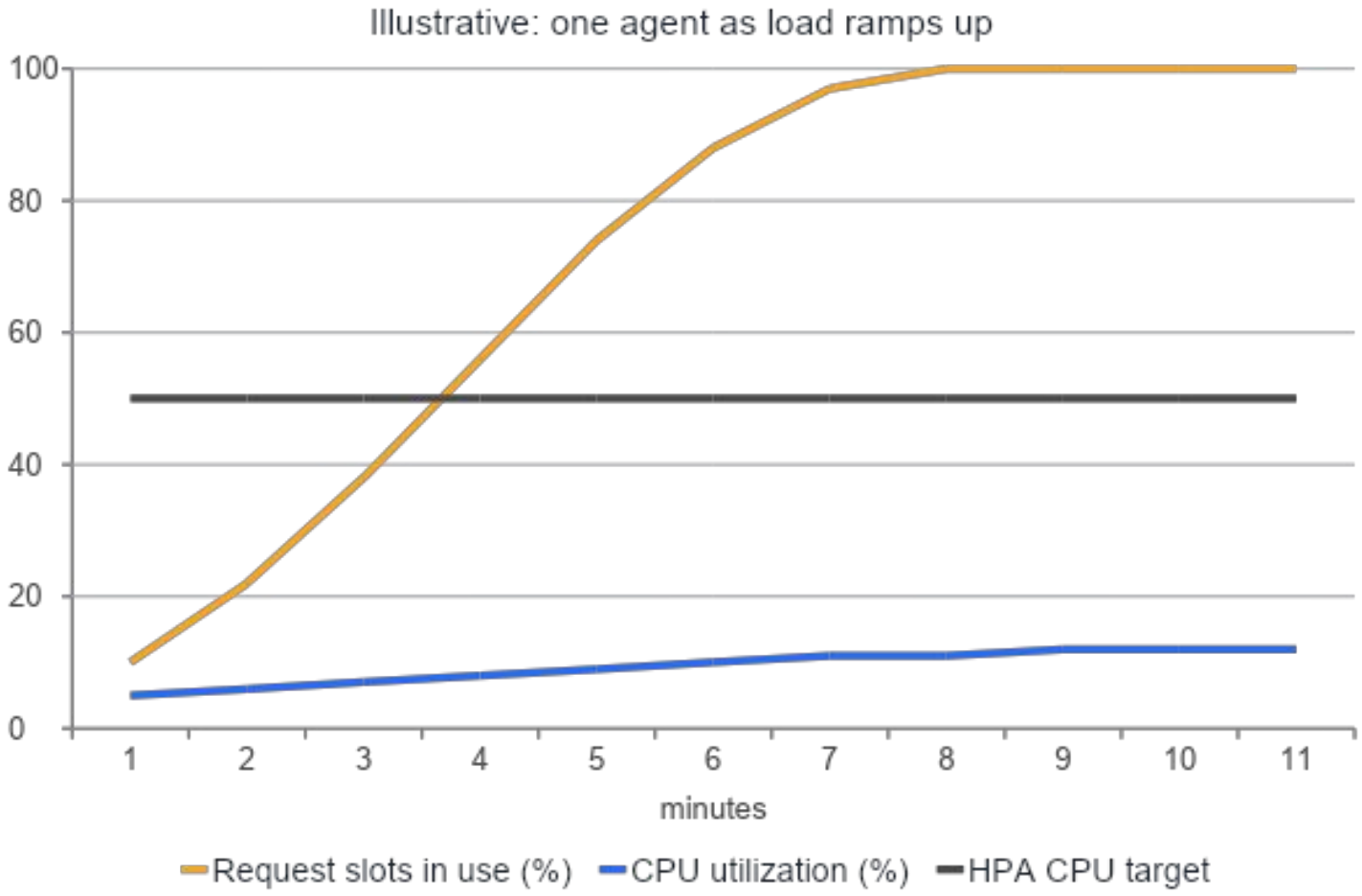
# call a peer by name: a ready A2A client
hotels = await client.peer_client("hotels")
async for event in hotels.send_message(request): ...

# or discover peers and turn them into tools
for name in client.peers():
    card = await client.resolve_agent_card(name)
    tools.append(as_tool(name, card))
```

- Get an A2A client for any peer by name
- Or get peer clients by discovery
- No URLs or Kubernetes API calls in agent code

**Under the hood:** the controller writes each agent's peers to a file in the pod, and every peer is a stable Kubernetes Service name. Pod restarts never strand a caller.

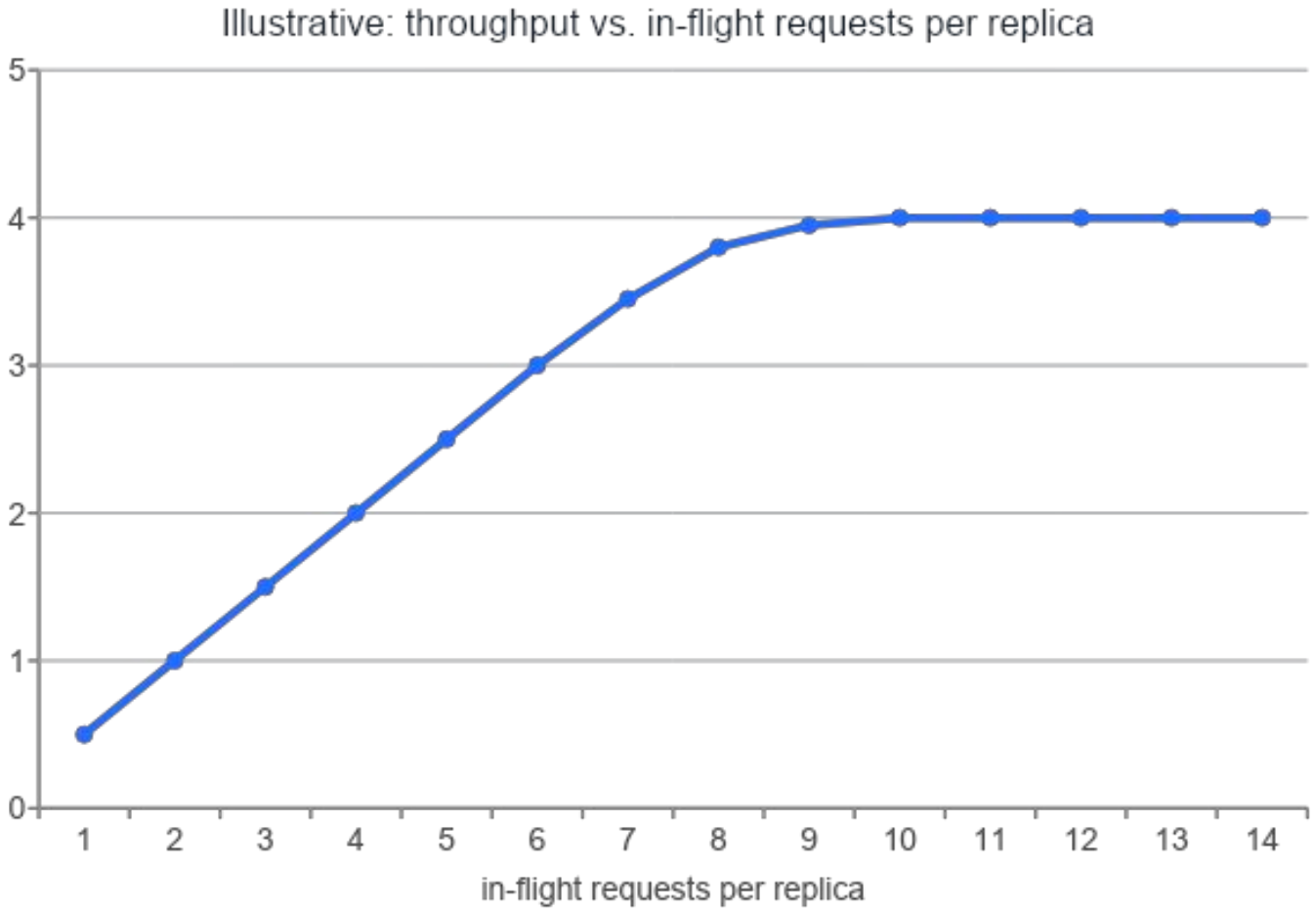
# CPU is the wrong signal for agents




An agent waiting on an LLM is idle on CPU and full on request slots.

CPU never crosses the HPA target, so the fleet never scales out while requests queue.

# Scale on in-flight requests; learn each agent's limit




## The knee

Where more concurrency stops buying throughput. Kynomesh learns it per agent from live traffic.

```
replicas = in-flight ÷
(knee × 80%)
```

No per-agent tuning. HPA and KEDA still work if you prefer.

# Protect what's downstream with rate limits

## Rate limiting




Travel: the flights agent fronts an airline API with a hard quota. Extra searches get a 429 and retry, instead of knocking the airline API over.

- Over the cap: 429, retry later
- One budget for the whole fleet
- At the cap, scale-up stops: replicas can't raise a quota

# Ship new versions without dropping calls

## Zero-downtime rollouts




- Outgoing pods finish in-flight requests first
- New pods get traffic only when ready
- Autoscaling pauses during a rollout

Travel: a booking call is mid-flight when we ship a new planner image. The old pod finishes the booking; new pods only join once ready.

# Demo

## 1 Deploy

Apply one AgentSet. The planner finds its flights and hotels workers.

## 2 Add

Add a weather worker. The planner picks it up with no code change.

## 3 Drift

Give hotels a new skill. Come back to check the result.

Demo resource - https://kyno.sh/stable/tutorials/langgraph/3-dynamic-peers/

# When a peer changes, restart only the callers that need it

A2A clients cache a peer's AgentCard. When the peer gains a skill, callers keep using the old card.

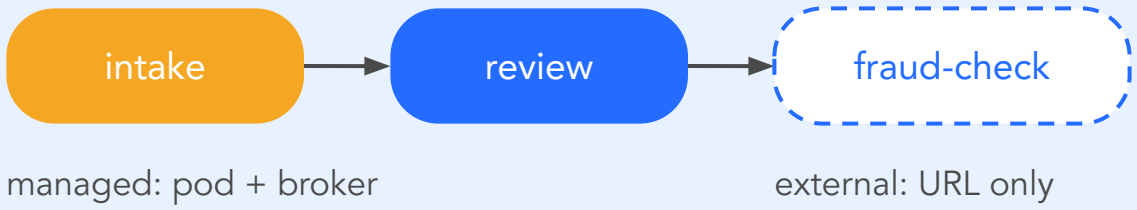



```
driftReload:
  enabled: true
```

- Opt-in, because restarting pods is a real behavior change
- Waits for a peer's rollout to settle before acting
- Respects the same rollout budget as a deploy

# Mix in-cluster and external agents

Sequential with an external last hop




intake → fraud-check → review is rejected at admission.

- **No pod, Service or broker.** An external agent is a name and a URL in the peer list.
- **Can’t be** the entry, and in Sequential it can only be the single last hop.
- **Cross-AgentSet calls** use the other set's Service DNS name, so teams share agents without redeploying them.

**Why the rule:** Kynomesh controls managed agents' code paths, not external ones. Constraining where external agents sit keeps every topology predictable.

# One resource declares the whole agent set

```yaml
apiVersion: kynomesh.kyno.sh/v1alpha1
kind: AgentSet
metadata:
  name: research-assistant
spec:
  pattern: Supervisor
  entry: coordinator
  driftReload:
    enabled: true
  agents:
    - name: coordinator        # LangGraph
      container:
        image: example/coordinator:1.4
      scale: { min: 2, max: 10 }
    - name: searcher           # Google ADK
      container:
        image: example/searcher:2.0
      rateLimit: { maxInFlight: 20 }
  externalAgents:
    - name: fraud-check
      url: https://fraud-check.example.com
```

- **`pattern + entry`**

  The controller derives every peer list from these two fields.

- **`agents[ ]`**

  Each becomes its own AgentDeploy: own pods, broker, Service and scaling.

- **`scale, rateLimit`**

  Set per agent. Autoscaling is on by default; limits are opt-in.

- **`externalAgents`**

  A name and a URL. Joins the peer list without a pod.

- **`driftReload`**

  Fleet-wide default, overridable per agent.

TAKEAWAYS

# Three patterns to take home

1

## Model coordination as topology

Put the pattern and entry in a CRD. Let a controller derive and enforce who may call whom.

In Kynomesh: AgentSet with Supervisor, Handoff or Sequential, any framework or language.

2

## Discover peers by name, and keep them fresh

Agents ask for peers by name; the controller derives who they may see.

In Kynomesh: Stable Service names for peers, plus drift reload to restart only stale callers.

3

## Scale on slots, not CPU, and ship safely

Autoscale on in-flight load, cap what sits downstream, drain before replacing pods.

In Kynomesh: Learned-knee autoscaling, maxInFlight rate limits and zero-downtime rollouts.

Your agents deserve a mesh, not a monolith.

# Thank you

https://github.com/kynoproj/kynomesh

https://kyno.sh · Apache 2.0

SDKs

```
Python: pip install kynomesh
Golang: go get github.com/kynoproj/kynomesh-go
```

Derek Wang · Intuit

Questions, issues and contributions welcome.