KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Who Ran Up the Inference Bill?

AI Traffic Control with KServe and Gateway API Inference Extension

Lasse Vierow Cloud Engineer



Moin, I'm Lasse



Lasse Vierow

Cloud Engineer, Independent
Germany

linkedin.com/in/lasse-vierow

github.com/Lasse4

Agenda

1

Problem

One GPU bill, no owner, and why tokens drive the cost

2

Serving stack

KServe, Gateway API and the Inference Extension

3

AI gateway

Prices, guardrails, token budgets and failover per team

4

Live demo

Four teams share one model on a kind cluster

5

Takeaways

What to bring back to your own platform

KubeCon CloudNativeCon
North America 2026

Why did my inference bill just explode?



One GPU bill, and no owner for the tokens

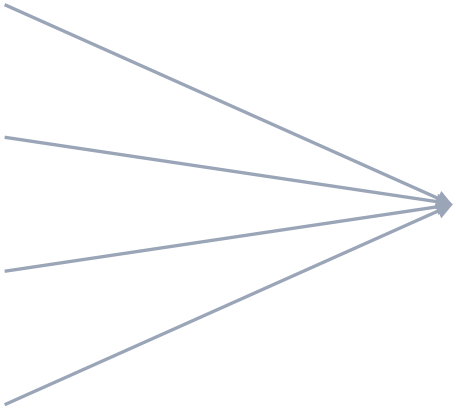
Many teams, one shared model

Support chatbot
Team: support

Code review agent
Team: platform

Nightly batch summaries
Team: data

Prompt experiments
Team: ML research



What the cluster bill shows

GPU cost, November

Namespace	kserve-test
Workload	mock-llm
Total due	**$17,280**

Not on the bill:

Which team?
Which service?
How many tokens?

All teams call one model, so its GPU cost lands on one namespace, not on the callers.

Requests lie. Tokens don't.

Team ? `POST /v1/chat/completions`

1 request 70 words → **~145 tokens**

Team ? `POST /v1/chat/completions`

1 request 7,000 words → **~14,000 tokens**

Tokens counted in + out: the demo model echoes the prompt back.

~100×

more tokens for the same single request.

Same endpoint, same 200 OK. Request counts hide where the money goes.

Which of the four teams sends the big one? The demo will tell.

An LLM request is not a normal HTTP request

	Classic API traffic	LLM inference traffic
Duration	Milliseconds, one response	Seconds to minutes, streamed token by token
Cost unit	One request is roughly one unit of work	Cost follows tokens: one request can be 20 tokens or 200,000
Routing signal	Host, path and headers	The JSON body: model, prompt, stream flag
Backends	Stateless, interchangeable replicas	GPU pods with queues, KV caches and loaded LoRA adapters

Requests-per-second limits and round-robin balancing see none of this.

KubeCon CloudNativeCon
North America 2026

Part 2 · Serving stack

How does a request reach the model?



KServe: one resource, a full serving stack

01-model.yaml (excerpt)

```yaml
apiVersion: serving.kserve.io/v1alpha2
kind: LLMInferenceService
metadata:
  name: mock-llm
spec:
  model:
    name: mock-llm
    uri: hf://mock/mock-llm
  replicas: 1
  router:
    route: {}      # HTTPRoute
    scheduler: {} # pool + picker
```

kubectl apply creates:

HTTPRoute Gateway API
attaches to the shared Gateway

InferencePool Inference Extension
the set of model server pods

Endpoint picker Inference Extension
picks a pod by queue and KV cache

Model pods KServe
the model server itself

More callers, more copies, one bigger bill

KServe

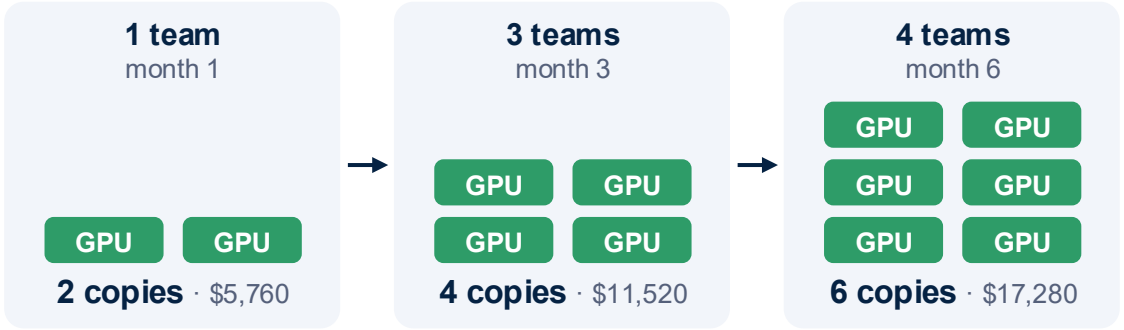
Scales on load, not on callers

Traffic grows, KServe adds copies.

Every copy holds its own GPU: $4 an hour, $2,880 a month.

KServe sees the load, but not which team sent it.

Same model, more callers, more copies

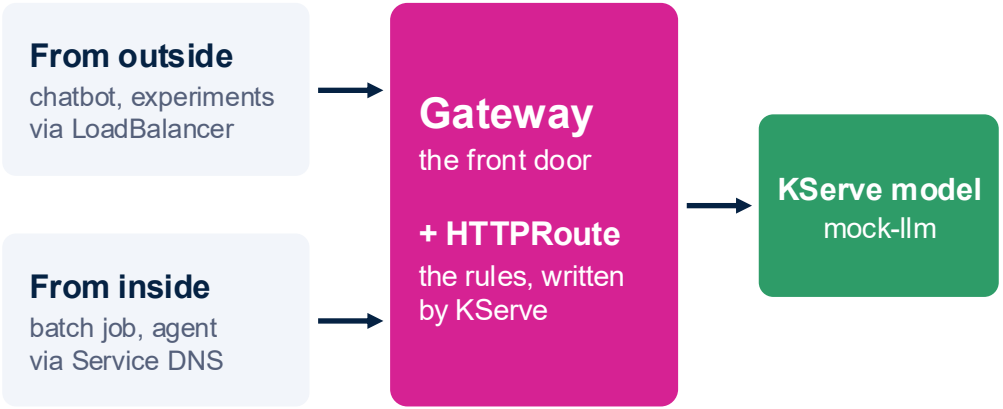


More tokens → More copies → More GPU nodes → **Bigger bill**

Request path

Who is calling? → **KServe model**

Gateway API: one front door for every request



1 **One address for everyone**
Outside via LoadBalancer, inside via Service DNS. Same door, same rules.

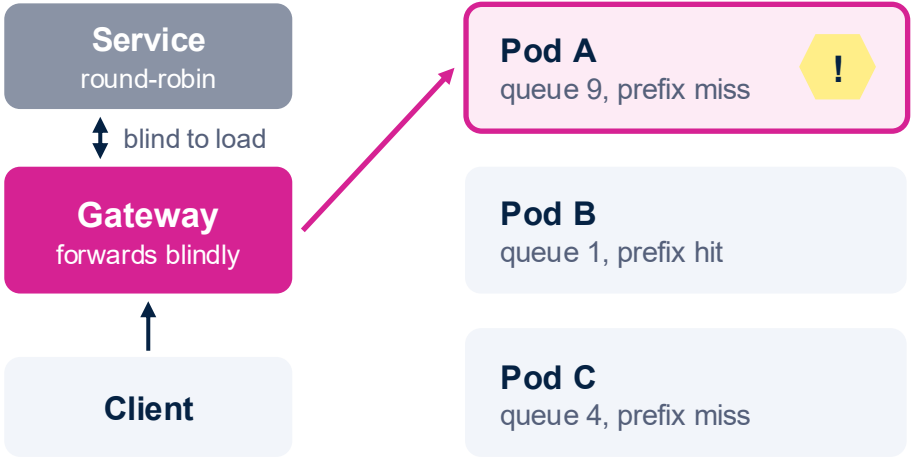
2 **KServe writes the route**
The HTTPRoute tells the gateway where the model is. You don't write it.

3 **Pick your gateway**
Gateway API is a standard. Any conformant gateway works. We use agentgateway.

Request path

Clients → **Gateway** → KServe model

Before: round-robin picks the busiest pod



1 Blind to load

Round-robin ignores queues and cache.

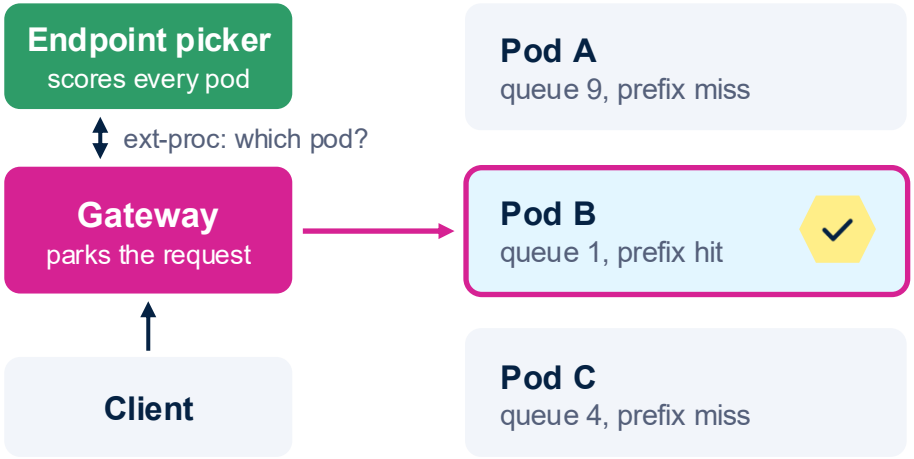
2 Pays twice

Cache misses recompute the same prompt.

Request path

Clients → Gateway → **Service** → KServe model

Inference Extension: ask before you route



1

Endpoint picker

Scores pods on queue depth, KV-cache use, prefix-cache hits and loaded LoRA adapters.

2

Cheaper tokens

More cache hits, more tokens per GPU.

Request path

Clients → Gateway → **InferencePool** → KServe model

KubeCon CloudNativeCon North America 2026

Part 3 · AI gateway

How do we put a name on every token?



AI gateway: a front door that reads tokens

agentgateway

A Gateway API gateway that speaks LLM

Open source · Linux Foundation

What the GPU bill can't tell you

Which team?	JWT claim → team label
How many tokens?	Read from every response
What did it cost?	Tokens × price list
Who caps the spend?	Token budget per team

Request path Clients → **agentgateway** → **AgentgatewayBackend** → InferencePool → KServe model

Self-hosted tokens need a price, too

$4.00
per GPU-hour

÷

10 M
tokens per GPU-hour

=

$0.40
per 1M tokens

Your GPUs have no list price. So set one: what the node costs, divided by what it really serves.

Use your own numbers — these are the demo's.

```
model-catalog ConfigMap · USD per 1M tokens

model       input    output
mock-llm    0.40     0.40
cloud-llm   2.50     10.00
```

Token budgets: cap the spend per team

jwt.team
the budget key

→

60,000
tokens per team per hour

→

429
when it is spent

Count tokens, not requests. Checked before each request, charged after it: the request that crosses the line still completes, the next one gets 429.

A rate limit service keeps the count, so every gateway replica shares one budget.

04-controls.yaml · team-token-budget (excerpt)

```yaml
rateLimit:
  global:
    domain: team-budgets
    descriptors:
    - entries:
      - name: team
        expression: jwt.team
      unit: Tokens  # not dollars
```

Failover without a surprise bill

1 · self-hosted `mock-llm`

KServe **$0.40** per 1M tokens

2 · cloud `cloud-llm`

Paid API **$2.50** in · **$10** out

Two priority groups in one AgentgatewayBackend (05-failover.yaml).
One 503 evicts our model until it recovers.

6–25×

the price per token, while the cloud answers.

Budgets count tokens, not dollars. A team's 60,000 tokens now cost up to 25× more. The log names the model that answered.

Guardrails: PII never reaches the model

04-controls.yaml · pii-guard (excerpt)

```yaml
kind: AgentgatewayPolicy
metadata: { name: pii-guard }
spec:
  backend:
    ai:
      promptGuard:
        request:
        - regex:
            builtins:
            - CreditCard
            - Ssn
            - Email
            action: Reject  # 403
```

Checked at the gateway, for every team:

Before the model request
Card numbers, SSNs, emails: 403

No tokens, no cost GPU
A rejected prompt never reaches a model pod

After the model response
Card numbers in answers are masked

One policy, not one per app HTTPRoute
Attached to the route KServe already wrote

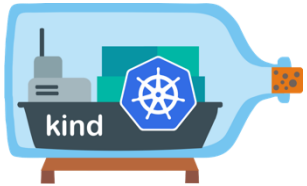
KubeCon CloudNativeCon
North America 2026

Part 4 · Live demo

Can we split the bill by team, live?



Tools used in the demo



kind

Kubernetes in Docker

agentgateway

agentgateway

AI-aware traffic routing



KServe

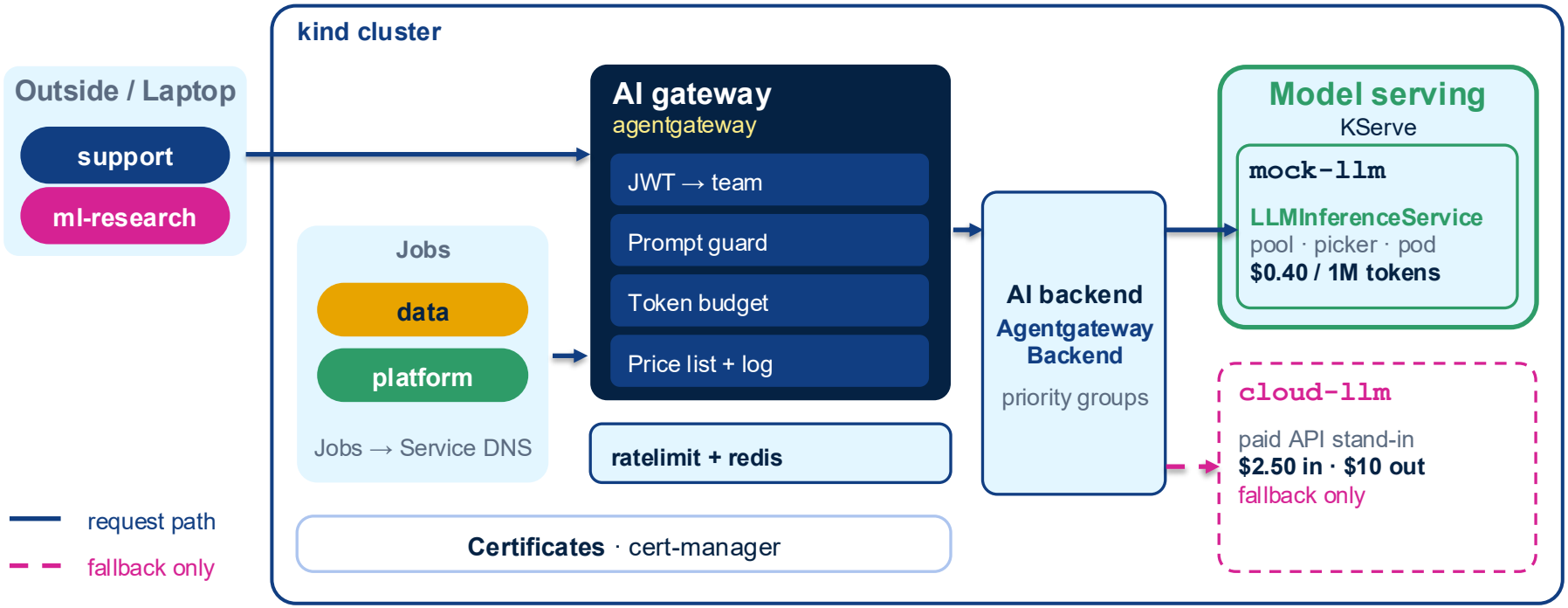
LLM inference serving



cert-manager

Required by KServe

Demo setup: what runs where



Follow along: the demo repo



github.com/Lasse4/
KubeConNorthAmerica2026

KubeCon CloudNativeCon
North America 2026

DEMO



Same bill, now with owners

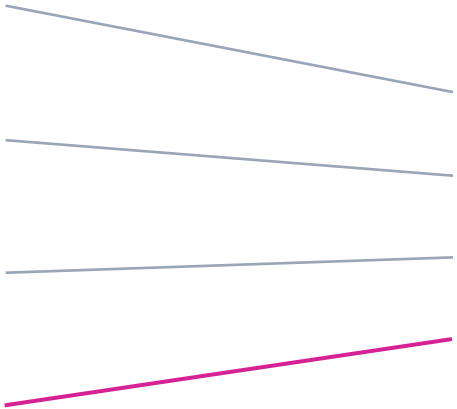
Same four teams

Support chatbot
24M requests · 5.7B tokens

Code review agent
7.6M requests · 7.5B tokens

Nightly batch summaries
61M requests · 8.8B tokens

Prompt experiments
1.5M requests · 21.2B tokens



What the bill shows now (same mix as the demo)

GPU cost, November, by team

support	13%	**$2,270**
platform	17%	**$3,010**
data	20%	**$3,510**
ML research	49%	**$8,490**

Total $17,280 = 43.2B tokens × $0.40 per 1M
Share of tokens = share of GPU cost

The batch job sent 65% of the requests. ML research sent 1.6% and used 49% of the tokens.

Four things to take home

1

Meter tokens, not requests.

The batch job looked guilty. The tokens told the truth.

2

Put a name on every token.

Team identity at the gateway turns one GPU bill into a line per team.

3

Control the spend at the gateway.

Token budgets per team, PII guardrails, and failover that names the model it used.

4

Build on standards, keep the policy thin.

KServe, Gateway API and the Inference Extension do the plumbing. AI policy sits at the gateway.

KubeCon CloudNativeCon
North America 2026

Thank you!



agentgateway Discord
discord.gg/y9efgEmppm

KServe on CNCF Slack
#kserve · slack.cncf.io

Guide: KServe + agentgateway
agentgateway.dev › KServe integration guide

Lasse4 lasse-vierow

Feedback