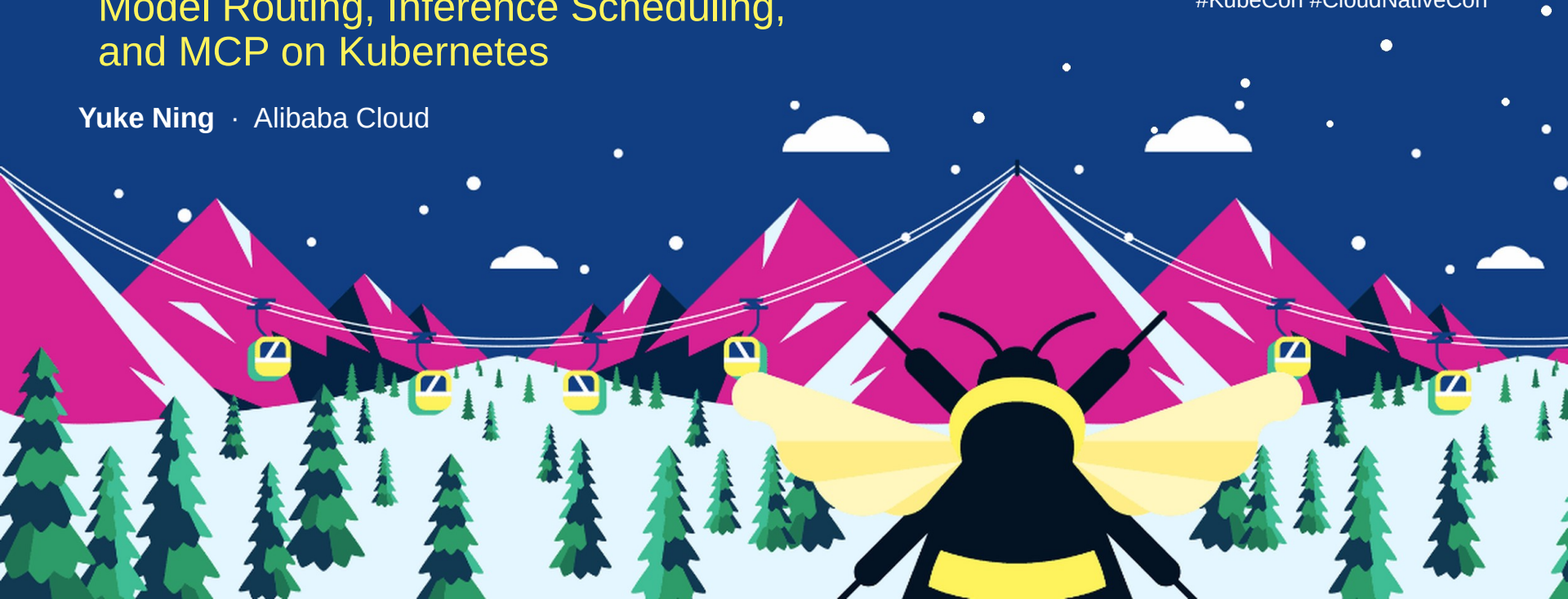
AI Gateways in Practice

Model Routing, Inference Scheduling, and MCP on Kubernetes

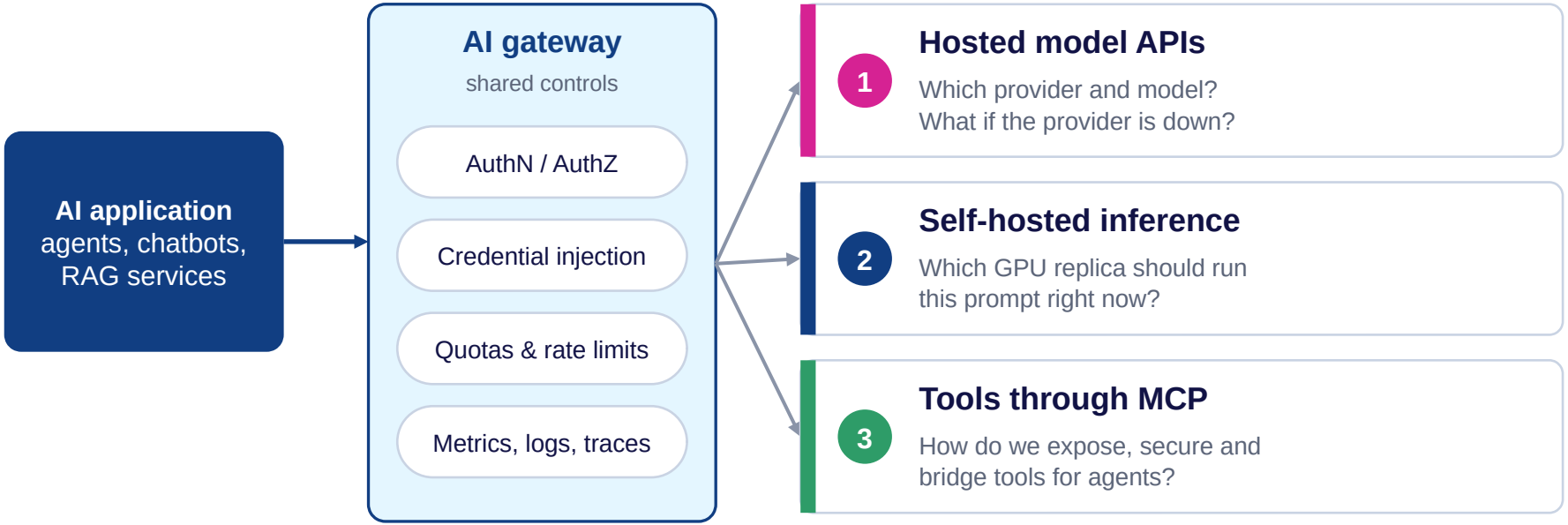
Yuke Ning · Alibaba Cloud

KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon



AI apps on Kubernetes need more than a model API call



One entry point, three different routing decisions, and a different failure model for each.

Agenda

01 **Model routing across providers**

one API surface, model-based routing, fallback

02 **Self-hosted models with Gateway API**

HTTPRoute → InferencePool → Endpoint Picker

03 **Inference scheduling**

embedded picker vs external EPP (llm-d), failures, conformance

04 **MCP through the gateway**

REST → MCP tools, MCP proxy, transport bridging

IMPLEMENTATION USED IN THIS TALK

Higress

- CNCF Sandbox project
- Built on Envoy and Istio
- Gateway API, Ingress and AI routing in one data plane
- Extensible with Wasm plugins and a Go filter

github.com/higress-group/higress

KubeCon CloudNativeCon North America 2026

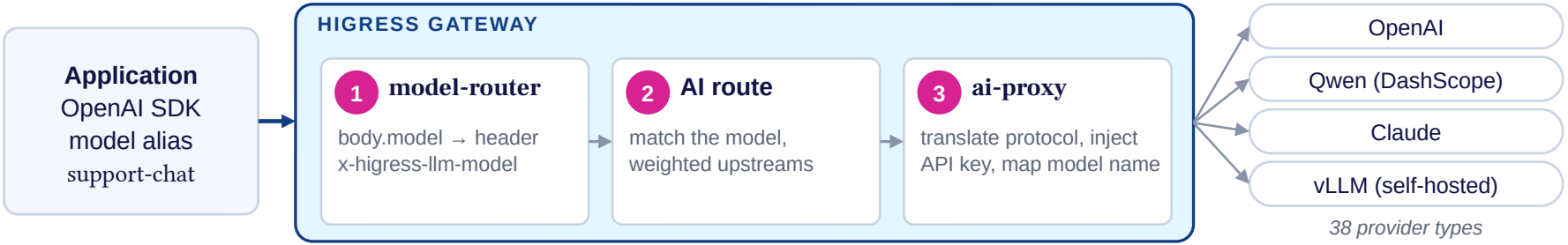
PART 01

Model routing across providers

One API surface for many model providers, and what happens when one of them fails



One OpenAI-compatible API in front of many providers



model-router

```
modelToHeader: x-higress-llm-model
addProviderHeader: x-higress-llm-provider
```

dashscope/qwen-long → provider header dashscope, body model qwen-long

ai-proxy (excerpt)

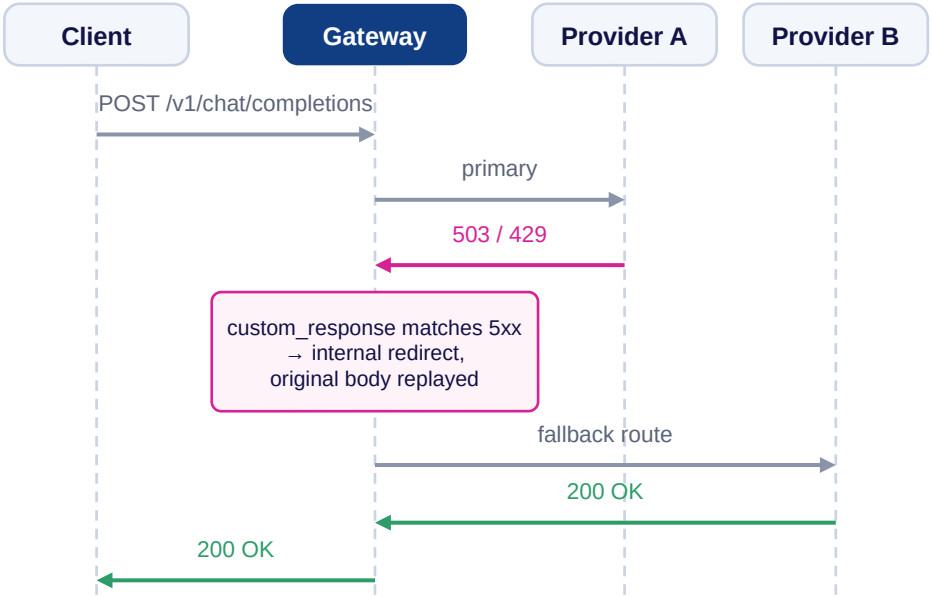
```
provider:
  type: qwen
  apiTokens:
  - "YOUR_QWEN_API_TOKEN"
  modelMapping:
    'gpt-4-*': 'qwen-max'
    '*': 'qwen-turbo'
```

- Apps keep **one SDK and a stable model alias**
- Provider **credentials stay in the gateway**
- Exact, prefix (*) and catch-all **model mapping**
- Token and latency **metrics per route, model and consumer**

Fallback when a provider becomes unavailable

AI route (Higress console / admin API)

```
name: support-chat
modelPredicates:
- matchType: EQUAL
  matchValue: support-chat
upstreams:
- provider: openai-prod
  weight: 100
  modelMapping: {support-chat: gpt-4.1-mini}
fallbackConfig:
  enabled: true
  responseCodes: ['4xx', '5xx']
  fallbackStrategy: SEQ
  upstreams:
  - provider: qwen-prod
    weight: 100
    modelMapping: {support-chat: qwen-plus}
```



The response carries `x-higress-fallback-from`, so fallback is visible.
Per request and stateless: the next request tries the primary again.

Two layers of failure handling

	API-key failover (ai-proxy failover)	Provider fallback (AI route fallbackConfig)
Scope	rotate between API keys of **one provider**	switch to **another provider** and model
Trigger	failoverOnStatus (default 4xx, 5xx) after failureThreshold failures	responseCodes class (4xx, 5xx) on the response
Recovery	health check with healthCheckModel, or cooldownDuration	none needed: each request starts at the primary
State	per gateway Pod	none: an Envoy internal redirect

Decide explicitly

429 ≠ 503
quota exhaustion and outages usually need different reactions

Streaming
fallback is decided on the status code; a stream that already returned 200 cannot be redirected

Model parity
the fallback model may differ in context window, tool calling and quality

Visibility
x-higress-fallback-from on the response, plus metrics per route and model

KubeCon CloudNativeCon North America 2026

PART 02

Self-hosted models with Gateway API

Connecting application traffic to an InferencePool through the Inference Extension



Model servers are not interchangeable endpoints

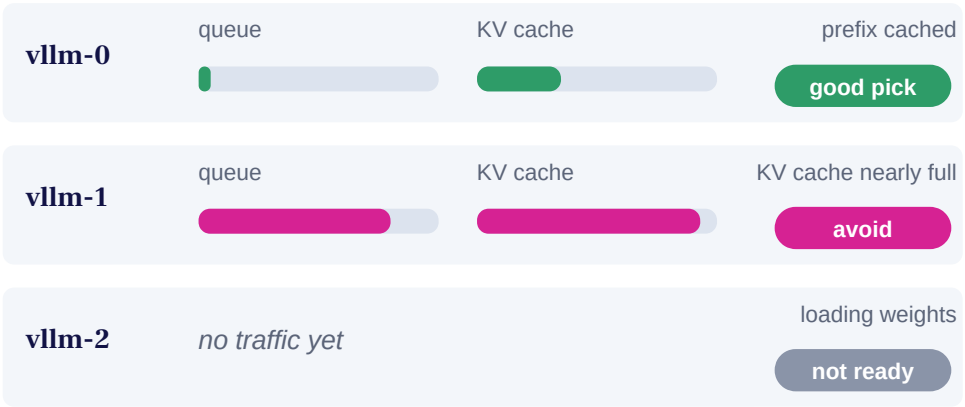
Classic Service backend

round robin / least request

- pod-1 · ~same cost per request
- pod-2 · ~same cost per request
- pod-3 · ~same cost per request

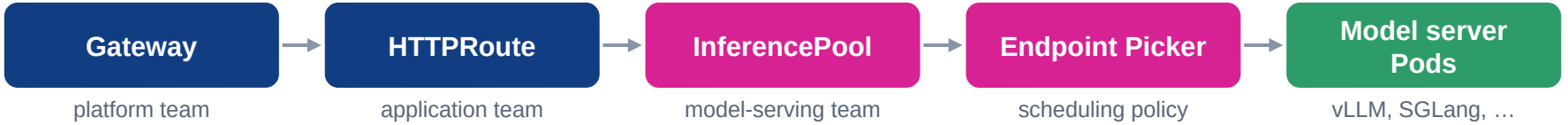
Model server replicas (same model)

state that connection metrics cannot see



Useful signals: queue depth, KV-cache utilization, prefix-cache locality, loaded LoRA adapters.

Gateway API Inference Extension: one more backend kind



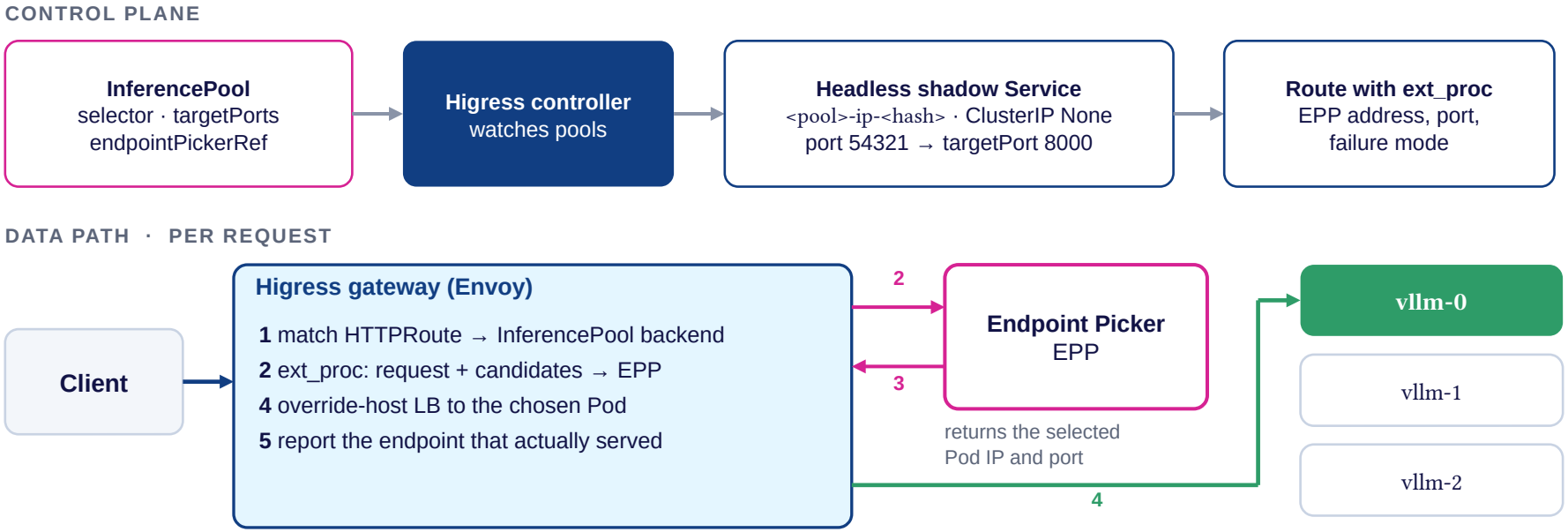
httproute.yaml

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: llm-route
spec:
  parentRefs:
  - name: inference-gateway
  rules:
  - backendRefs:
    - group: inference.networking.k8s.io
      kind: InferencePool
      name: vllm-pool
```

inferencepool.yaml

```yaml
apiVersion: inference.networking.k8s.io/v1
kind: InferencePool
metadata:
  name: vllm-pool
spec:
  selector:
    matchLabels:
      app: vllm-llama3-8b
  targetPorts:
  - number: 8000
  endpointPickerRef:
    name: vllm-pool-epp
    port: {number: 9002}
    failureMode: FailClose
```

Inside Higress: from InferencePool to a selected Pod



Opt-in: Helm global.enableInferenceExtension: true + Inference Extension CRDs + an EPP deployment

KubeCon CloudNativeCon North America 2026

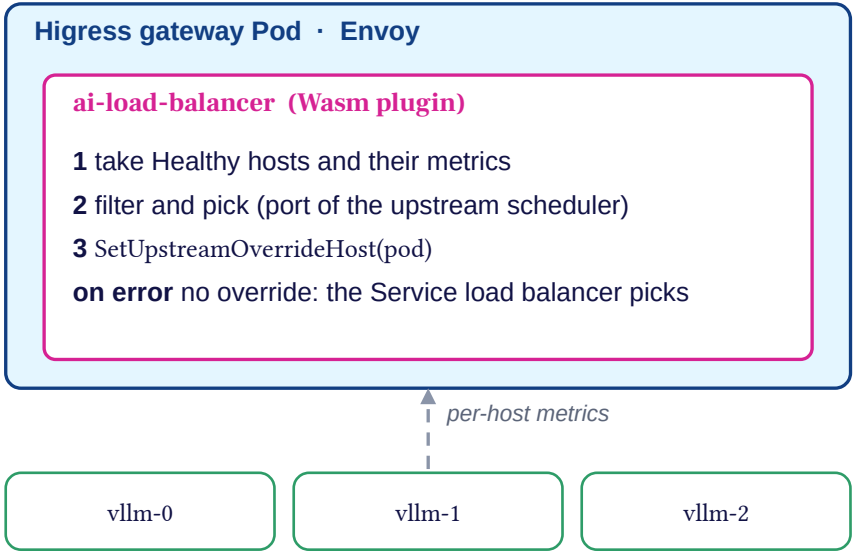
PART 03

Inference scheduling

Embedded endpoint picker vs. external Endpoint Picker (EPP): trade-offs, failures, conformance



Option A: an endpoint picker inside the gateway

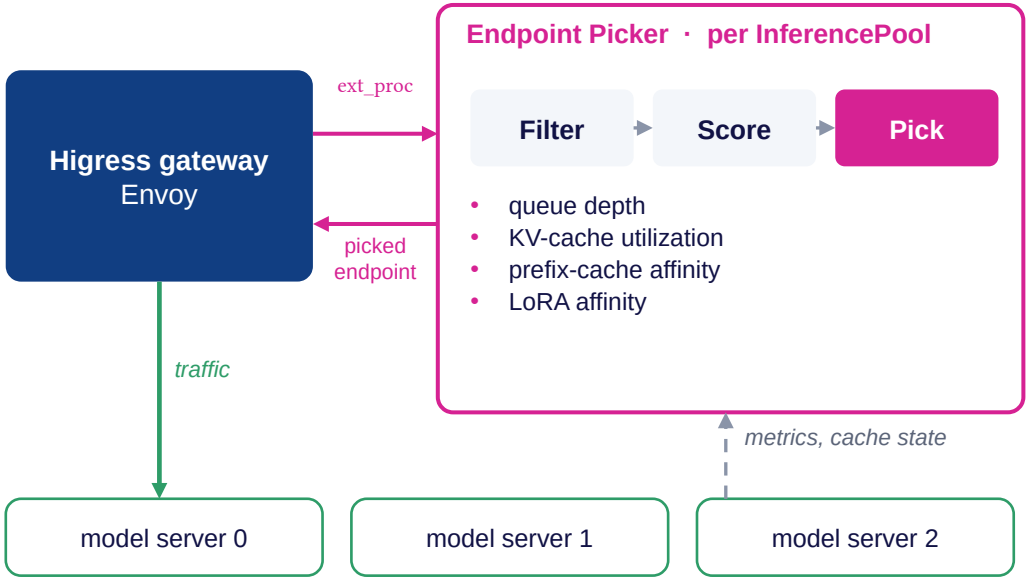


Policy	Signal	State
endpoint_metrics	queue, KV cache, LoRA	local
prefix_cache	prompt prefix → same Pod	Redis
global_least_request	in-flight across gateways	Redis
cluster_metrics	busy, latency, TTFT per cluster	local

ai-load-balancer

```
lb_type: endpoint
lb_policy: endpoint_metrics
lb_config:
  metric_policy: least
  target_metric: vllm:num_requests_waiting
  rate_limit: 0.6    # max share per Pod
```

Option B: an external Endpoint Picker, e.g. llm-d



- **Same API:** HTTPRoute → InferencePool. Change the scheduler without touching application routes.
- **Pluggable:** filters and scorers evolve with the upstream project.
- **llm-d** adds precise prefix-cache routing, prefill/decode disaggregation and latency prediction.
- **Cost:** one more component per pool and one gRPC round trip per request.

The EPP decides; the gateway still owns the data path and its policies.

Embedded or external: the trade-offs

	Embedded picker (ai-load-balancer)	External EPP (e.g. llm-d)
Runs	inside every gateway Pod	separate Deployment per InferencePool
Per-request cost	no extra hop	one ext_proc gRPC round trip
Signals	per-host metrics in the gateway; Redis for shared state	model-server state, prefix cache, P/D roles
Policy changes	fixed set of policies in plugin config	pluggable filters and scorers
API	Higress plugin configuration	standard InferencePool + EPP protocol
If the picker fails	scheduling is skipped; Service load balancing applies	failureMode: FailClose (default) rejects, FailOpen lets the gateway pick
Good fit	small pools, few moving parts	large pools, cache-heavy workloads, portability

Both can run in the same gateway, so the choice is per route.

When inference instances become unavailable

Situation	Embedded picker	External EPP
Pod NotReady or deleted	only Healthy hosts are candidates	leaves the pool; the EPP stops picking it
Pod alive but saturated	KV-cache and queue thresholds filter it out	scorers steer away from it
Every Pod saturated	no picker can add capacity: queue, rate-limit or scale out	same
Picker unavailable	nothing extra to fail; Service load balancing applies	FailClose (default): request fails FailOpen: gateway picks an endpoint
Retry lands on another Pod	the route's retry policy applies	Higress reports the endpoint that actually served

Set failureMode **deliberately.** FailOpen favors availability. FailClose keeps placement guarantees (prefill/decode roles, adapters loaded on some Pods) at the cost of errors.

Conformance: which behaviors match the upstream specs

Gateway API

v1.6.0 · GATEWAY-HTTP profile

37 / 37

core tests passed
no failures, no skips

Report accepted into the upstream kubernetes-sigs/gateway-api repository

Gateway API Inference Extension

v1.4.0 · Gateway profile

12 / 12

core tests passed
no failures, no skips

Reproducible report published by the Higress project (not an upstream badge)

How to read these results

Scoped
to the named core profile, not every extended or implementation-specific feature

Contract, not quality
it verifies API behavior, not which picker gives better latency

Reproducible
reports record Kubernetes version, CRDs, source commit and image digests

KubeCon CloudNativeCon North America 2026

PART 04

MCP through the gateway

Exposing REST APIs as tools, proxying MCP servers, and bridging protocol versions



Three jobs for a gateway in front of MCP tools

1 Expose REST APIs as tools

`mcp-server · type: rest`

- describe args and request / response templates
- the gateway answers `tools/list` and `tools/call`
- no new service to write

2 Proxy existing MCP servers

`mcp-server · type: mcp-proxy`

- authentication and credential policy
- rate limits, audit logs, metrics
- `mcp-router` routes `server/tool` names

3 Bridge transports and versions

explicit, per upstream

- 2026-07-28 clients → legacy servers
- legacy SSE clients → REST tools (state in Redis)
- no silent protocol switching

Shared with model traffic: one place for **authentication, quotas, metrics and logs**.

A REST API becomes an MCP tool through configuration

mcp-server config · samples/mcp/protocol/2026-07-28/02-rest-to-mcp

```
server:
  name: weather-demo
  type: rest
  version: 1.0.0
tools:
- name: get_weather
  description: Query deterministic weather
  args:
  - name: location
    description: City name
    type: string
    required: true
  requestTemplate:
    url: http://observable-weather.mcp-demo.svc.cluster.local:8080/weather
    method: GET
    argsToUrlParam: true
```

tools/list

returns get_weather with an input schema built from args

↓

tools/call get_weather {'location': 'Seattle'}

GET .../weather?location=Seattle (argsToUrlParam)

↓

REST response

becomes the MCP result, optionally via responseTemplate

The sample checks: one tools/call → exactly one REST call.

MCP's HTTP transport has changed shape, and gateways feel it

2024-11-05

HTTP + SSE

- GET /sse stream + POST endpoint
- connection-scoped session
- deprecated since 2025-03-26

gateway keeps long-lived streams sticky

2025-03-26 … 2025-11-25

Streamable HTTP with sessions

- single MCP endpoint
- optional Mcp-Session-Id
- server requests over SSE

gateway needs session state or affinity

2026-07-28

Stateless Streamable HTTP

- every request is self-contained
- no sessions, no GET stream
- metadata mirrored into headers

any replica can serve any request

2026-07-28 request: routable from headers

```
POST /mcp HTTP/1.1
Content-Type: application/json
MCP-Protocol-Version: 2026-07-28
Mcp-Method: tools/call
Mcp-Name: get_weather
```

- **Route, authorize and rate-limit per tool** from headers, without parsing JSON-RPC
- **Body stays the source of truth:** servers reject header/body mismatches (HeaderMismatch)

Bridging clients and servers, explicitly

Client	Upstream	Behavior
2026-07-28	REST or registered tools	supported
2026-07-28	MCP server with protocolStrategy: modern	one stateless request per exchange
2026-07-28	MCP server with protocolStrategy: legacy	isolated legacy handshake per request
legacy	legacy MCP server	existing session behavior, unchanged
legacy	modern-only MCP server	not supported yet

03-modern-to-legacy (excerpt)

```
server:
  name: legacy-bridge-demo
  type: mcp-proxy
  transport: http
  protocolStrategy: legacy
  mcpServerURL: http://legacy-mcp.mcp-demo.svc.cluster.local:8080/legacy
```

No auto-detection, no silent fallback. mcp-proxy defaults to legacy.

Isolated handshake: the upstream session ID never reaches the client.

Takeaways

1. **Name the decision**

   provider choice, endpoint choice and tool transport are different problems with different failure models

2. **Separate key failover from provider fallback**

   decide status codes, streaming behavior and model parity up front

3. **Use the standard boundary for self-hosted models**

   HTTPRoute → InferencePool; pick an embedded or external picker per pool; set `failureMode` on purpose

4. **Be explicit about MCP versions**

   declare each upstream's profile, validate at the edge, forward only what the tool needs

RESOURCES

Try it and tell us what breaks

Model routing	plugins/wasm-go/extensions/ai-proxy, .../model-router
Embedded picker	plugins/wasm-go/extensions/ai-load-balancer
Inference Extension	Helm global.enableInferenceExtension: true
MCP tools	plugins/wasm-go/extensions/mcp-server
MCP samples	samples/mcp/protocol/2026-07-28
Conformance	test/gateway/reports/v1.6.0, test/inference-extension/reports/v1.4.0

Feedback we are looking for

Gateway API ownership boundaries · EPP behavior under model load · MCP compatibility during migration

github.com/higress-group/higress

Higress


KubeCon
CloudNativeCon
North America 2026
Thank you!
Questions?
Yuke Ning · Alibaba Cloud