Why Kubernetes Load Balancing Fails with HTTP/2 Traffic

A real production incident caused by HTTP/2 multiplexing

Quick Note About Me !

I am Mariem ;) Site Reliability Engineer

I spend most of my time running and scaling Kubernetes platforms in production !

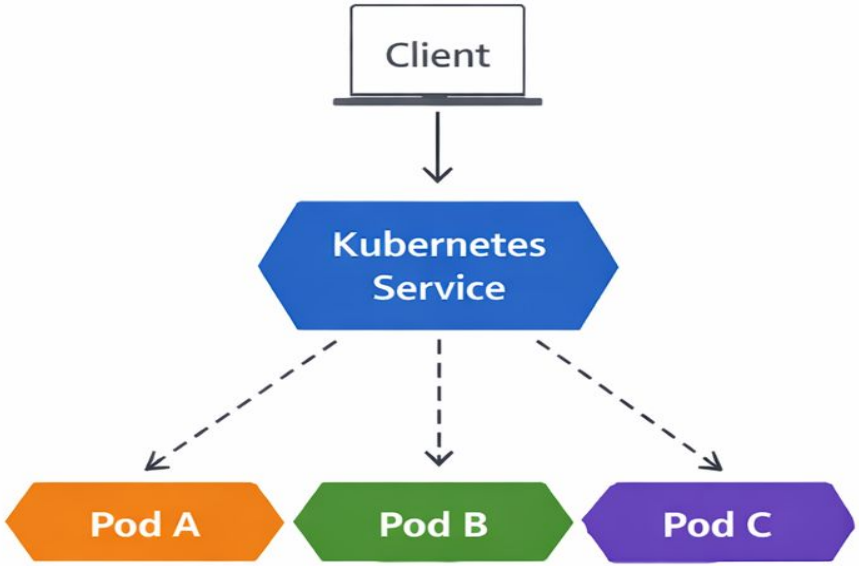
Agenda

- The incident we encountered.
- Why Kubernetes load balancing behaved unexpectedly.
- How HTTP/2 changed traffic distribution.
- Why kube-proxy caused a bottleneck.
- Temporary adjustments, long-term solution and lessons learned.

The Incident

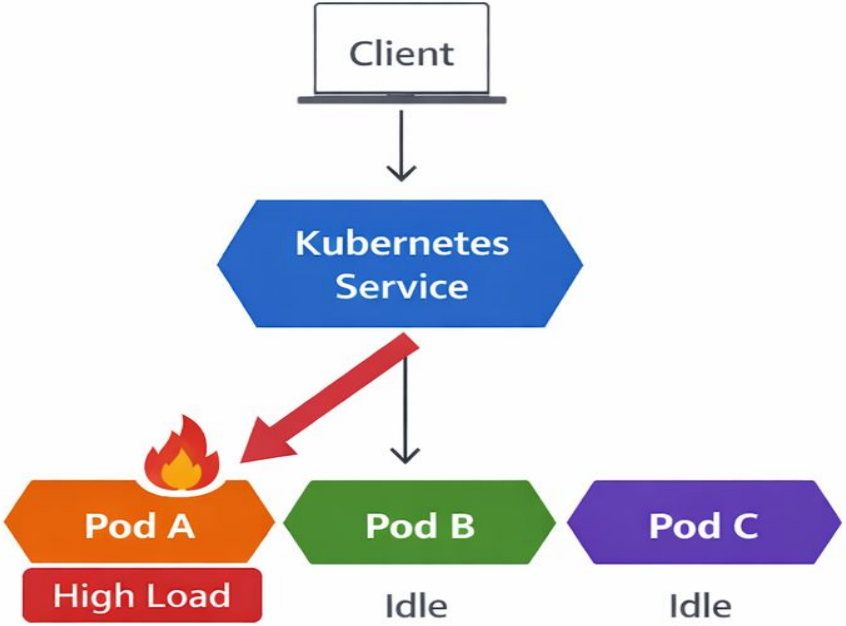
- HTTP/2 traffic for critical Kubernetes services in production.
- User experience impact: Latency
- Unusual load pattern was observed in HTTP/2 traffic distribution across backend pods.
- Traffic was being forwarded to one single pod.

Expected: Even Traffic Distribution



Traffic distributed evenly across pods

Reality: Imbalanced Traffic



Most requests sent to one overloaded pod

What Made This Incident Confusing

- Pods and nodes metrics showed plenty of idle capacity.
- Other backends pods for the same service remained mostly idle.
- HPA was working normally and not affected by the traffic pattern.
- No traffic redistribution even after restarting the single serving pod.

Our Initial Assumption

We expected HTTP/2 traffic to behave like HTTP/1.x traffic

Assumption:
- Requests evenly distributed across backend pods.
- Kubernetes service would balance traffic equally.
- Scaling worked normally as traffic increased.

Reality:
- Traffic became highly uneven.

How Kubernetes Services Load Balance Traffic

By default, Kubernetes Service performs Layer-4 (TCP) load balancing.

Key behavior:

- kube-proxy selects a backend pod when a TCP connection is created
- The selection uses a round-robin algorithm
- The chosen pod handles all requests on that connection

-> Load balancing happens at the connection level, not the request level

Why HTTP/1.x works fine with this model

Typical behavior of HTTP/1.x clients:

- Clients open multiple short-lived TCP connections
- Each new connection may be routed to a different backend pod
- Requests are therefore distributed across pods

-> Load balancing appears even and stable.

Why HTTP/2 caused traffic imbalance

HTTP/2 introduces **request multiplexing** over a single connection.
Key differences:

- A client can send many requests over one TCP connection
- The connection stays long-lived
- Kubernetes still balances only at the connection level
- As a result, all requests can be routed to the same pod

What Actually Happened

After introducing HTTP/2 traffic:

- One pod received **most of the requests**
- CPU usage spiked on that pod
- Request latency increased significantly
- Other pods remained **mostly idle**

Temporary Mitigation

Short-term adjustments:

- Tuned **application keep-alive settings**
- Reduced **connection reuse** from HTTP/2 clients
- Increased the number of **active connections**

-> This improved **traffic distribution temporarily**

```
http:
  keep_alive: true
  max_idle_connections: 20
  max_connections_per_host: 5
  idle_connection_timeout: 30s
```

```
http2:
  max_concurrent_streams: 50
  connection_pool_size: 5
```

```
connection_pool:
  min_connections: 3
  max_connections: 10
```

“How to permanently fix the issue ?”

Key Question !

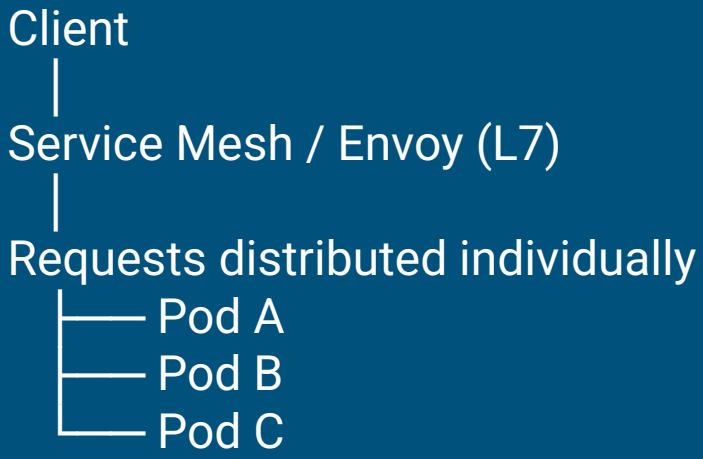
Long-Term solution: Layer-7 routing / service mesh

- Service Mesh (Istio, Linkerd)
- HTTP-aware proxy (Envoy)
- API Gateway / Ingress with request-level routing

- Added an **HTTP-aware routing layer: service mesh**
- Requests are balanced **at the request level**, not the connection level
- HTTP/2 multiplexed requests are **distributed across pods**
- Traffic is now **evenly balanced between backend pods**

How It Solved the Problem

Instead of routing connections, the proxy routes requests:

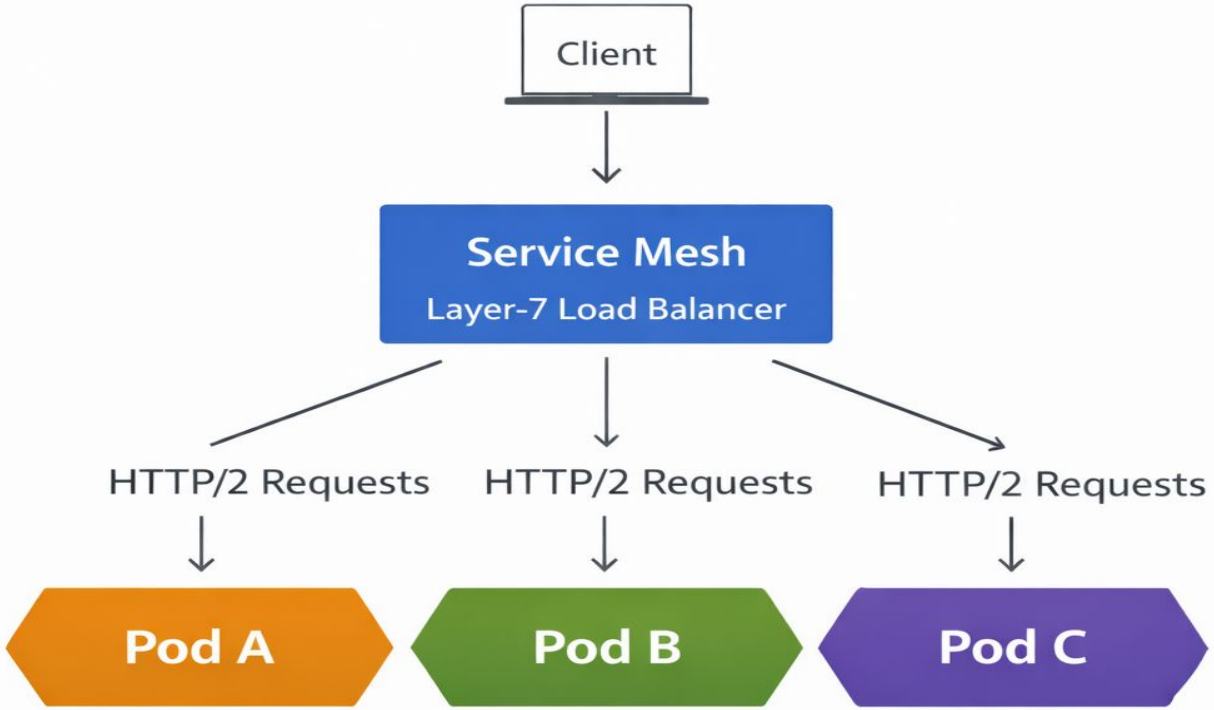


What the Proxy Understands

The proxy can inspect HTTP traffic and make better decisions:

- Request paths
- Headers
- Latency / health
- Routing rules

Service Mesh Routes Traffic at the Request Level



Traffic balanced among individual backend pods

Key Learnings

- Kubernetes Services perform **connection-level (Layer-4) load balancing**
- HTTP/2 multiplexing can cause **traffic imbalance across pods**
- A single long-lived connection can **overload one backend pod**
- **Layer-7 routing** is required for HTTP-aware traffic distribution

Thanks!