

gRPConf

INDIA 2026

gRPConf INDIA 2026

gRPC Probes in Kubernetes: Evolution, Use Cases, Performance, and Security

Neeraj Krishna Gopalakrishna (Senior Principal Software Engineer @ Red Hat)

Amritansh (Associate Software Engineer @ Red Hat)

The port is open. Are we ready?

gRPConf INDIA 2026

Scenario 1 • Traffic Routing

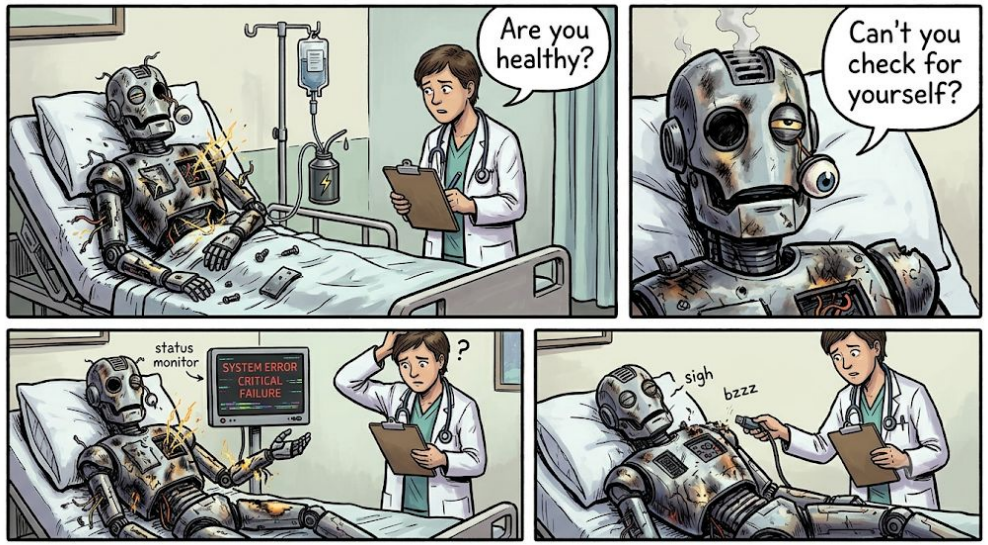
Can a gRPC server be reachable and alive, yet still unable to serve users?

➜ **Should Kubernetes send traffic to it?**

Scenario 2 • Failure Recovery

The service finished starting, but later gets stuck. The process exists, yet cannot work.

➜ **Should Kubernetes keep waiting or restart it?**



Why "Is it healthy?" is too broad a question

We must break health probing down into three explicit questions:

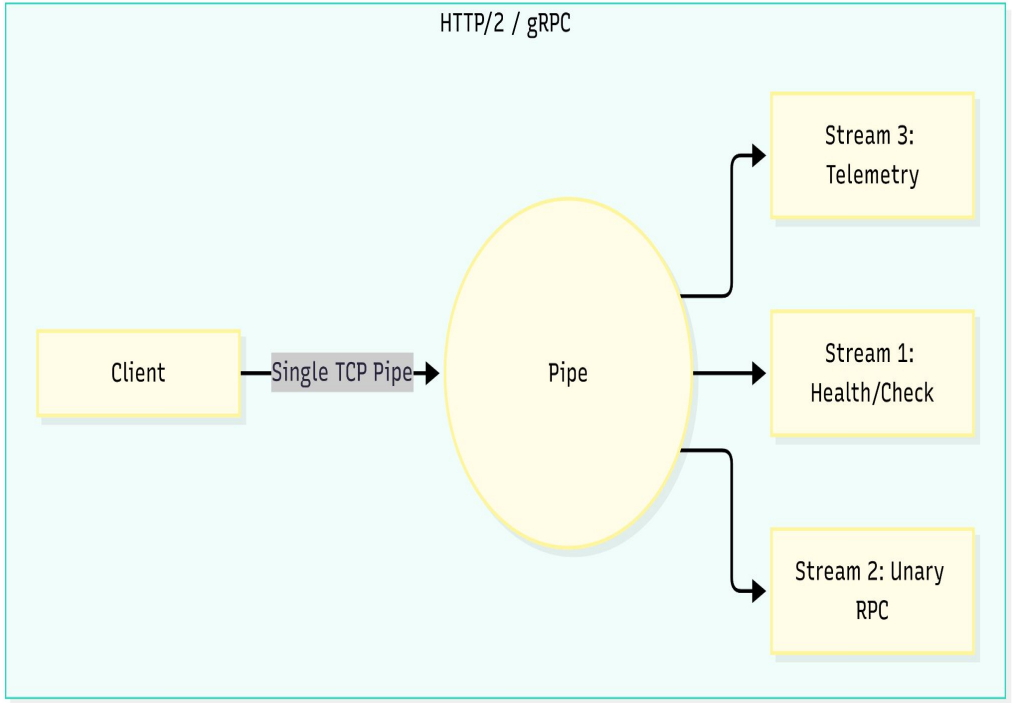
01
Has it started?
Startup Probe

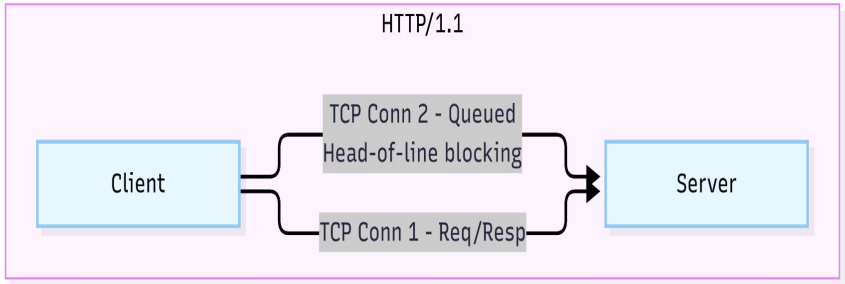
02
Is it ready to serve?
Readiness Probe

03
Does it need recovery?
Liveness Probe

Protocol Foundations:- HTTP/1.1 vs HTTP/2 vs gRPC in Health Probing

Feature	HTTP/1.1	HTTP/2	gRPC
Serialization	Text-based	Text/Binary framing	Protocol Buffers
Payload Size	Large wire overhead	Medium	Ultra-compact
Connection Model	1 request per TCP connection	Multiplexed streams over a single TCP connection	Multiplexed HTTP/2 streams natively
Head-of-Line Blocking	Application + transport layer blocking	Transport-level only; no request head-of-line blocking	Non-blocking parallel bidirectional streaming
Probe Footprint on Kubelet	High socket churn, TCP connection tear-down overhead	Low socket churn via connection reuse	Minimal socket pressure, standardized RPC schema





grpc Probe Options

Option	Default	Purpose
`initialDelaySeconds`	0	Initial wait before probing.
`periodSeconds`	10	How often to run the check.
`timeoutSeconds`	1	Time allowed for a check to complete.
`failureThreshold`	3	Consecutive failures before taking action.
`successThreshold`	1	Consecutive successes required after failure. Must be 1 for startup and liveness.
`terminationGracePeriodSeconds`	Inherits Pod setting	Shutdown grace after startup/liveness failure; unavailable for readiness. Pod default is 30 seconds.

Evolution

- **2016** - The early debate was about Kubernetes' scope - weather supporting gRPC would create pressure to embed every RPC protocol

- **2018** - The external tool - Package grpc_health_probe in the container and invoke it through an exec probe.

- **2021** - Kubelet gains built-in gRPC probes for startup, readiness, and liveness; disabled by default.

- **2023** - Native gRPC probes graduate to GA.

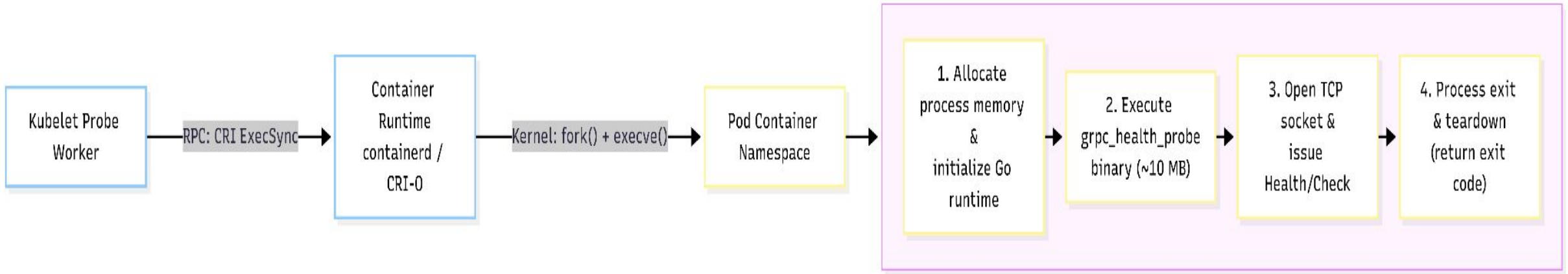
- **2026** - TLS enhancements

The Pre-Native Era:- Workarounds, Exec Spawns, and Bottlenecks

The Workarounds Before Native Support:

- **The Exec Workaround (grpc_health_probe):**
  - Bundled a ~10 MB static Go binary into every application container.
  - Over 10.5M+ GitHub downloads; widespread across production, including Google.
  - Invoked via exec probe: forks a brand-new OS process inside the container every probe interval.

- **The Dual Listener Hack:**
  - Applications runs a second HTTP/1.1 listener solely to satisfy httpget probes.
  - Introduces Extra attack surface, duplicate port management, security hardening requirements and memory overhead.



KEP-4939:- Native gRPC Probes with TLS Support

Context: Native plaintext gRPC probes were introduced via KEP-2727 (GA in v1.27), but lacked TLS transport support.

The Solution: KEP-4939 (Currently targeting beta in 1.38 release)

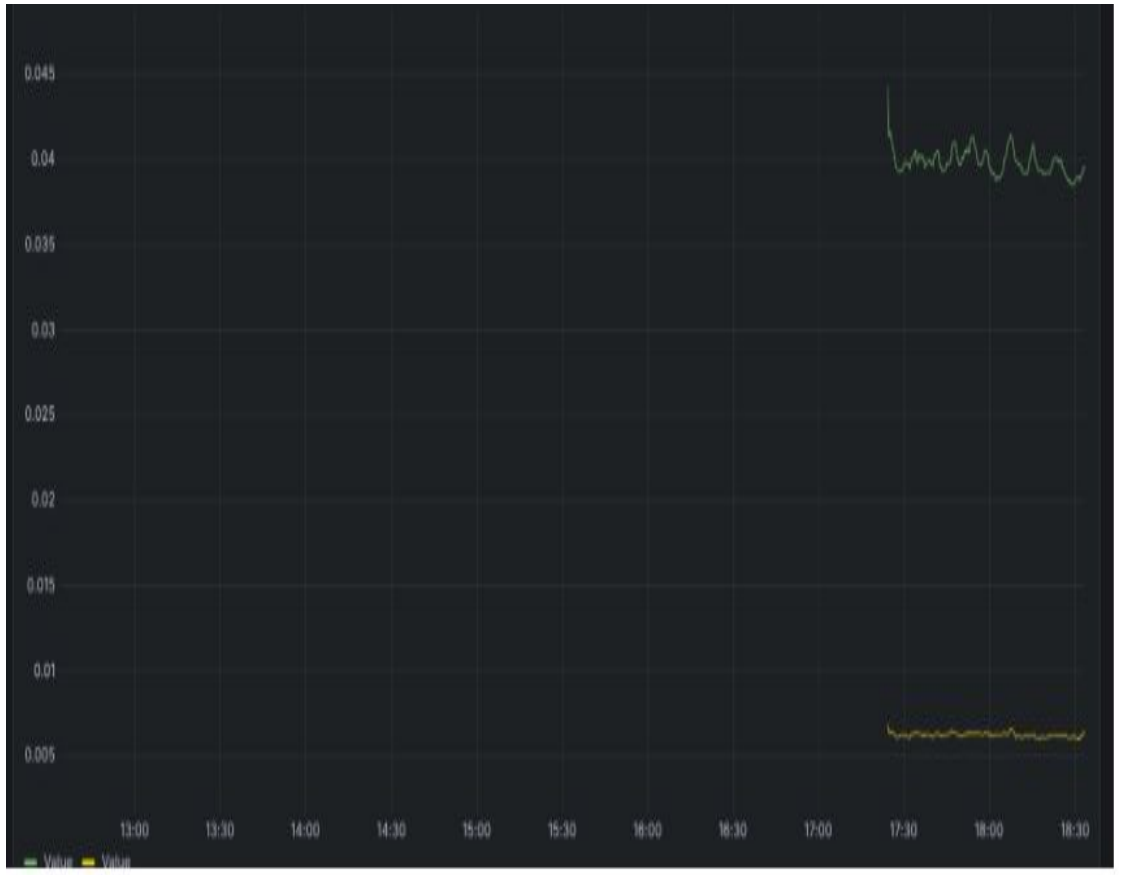
- **Declarative API:** Adds an optional mode field to the grpc probe spec (Plaintext | TLS, defaulting to Plaintext).
- **In-Process TLS Handshake:** Kubelet dials directly over HTTP/2 with InsecureSkipVerify: true, securing the wire without failing on ephemeral pod IPs.
- **Direct RPC Execution:** Calls grpc.health.v1.Health/Check in-process:- zero image bloat, zero CRI involvement, and no process forks.



Production Benchmarks:- Exec Probe vs. Native gRPC TLS

- **~6.7x Faster Response:** Native TLS probes drop average latency from **~40ms** (exec) down to **~6ms**.
- **Zero Process Overhead:** Eliminates recurring Linux kernel `fork()`/`execve()` calls and Go runtime spin-up.
- **Direct In-Process Dial:** Kubelet worker threads dial over HTTP/2 TLS directly, completely bypassing the CRI bridge.
- **Flatline Stability:** Delivers predictable, low-jitter execution, eliminating the latency spikes and CPU throttling seen with exec probes

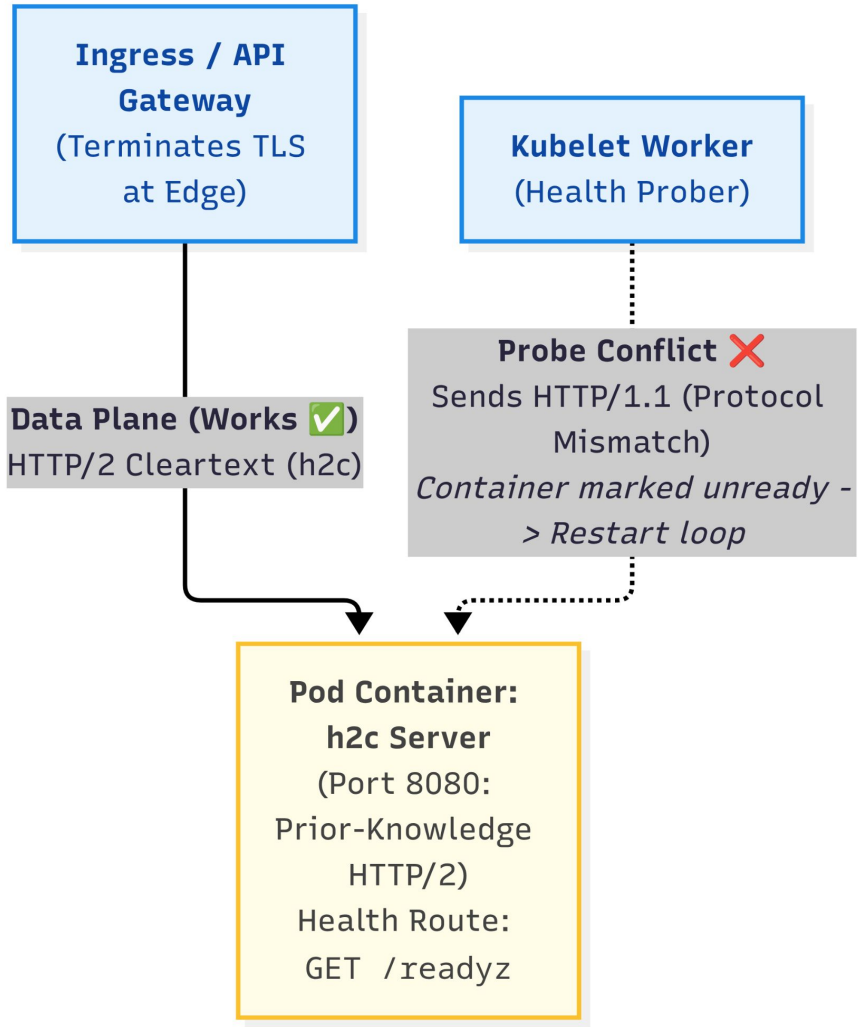
Latency Profile Comparison



- Green line (~0.040s / 40ms): - Exec Probe
- Yellow line (~0.006s / 6ms): - Native gRPC TLS Probe

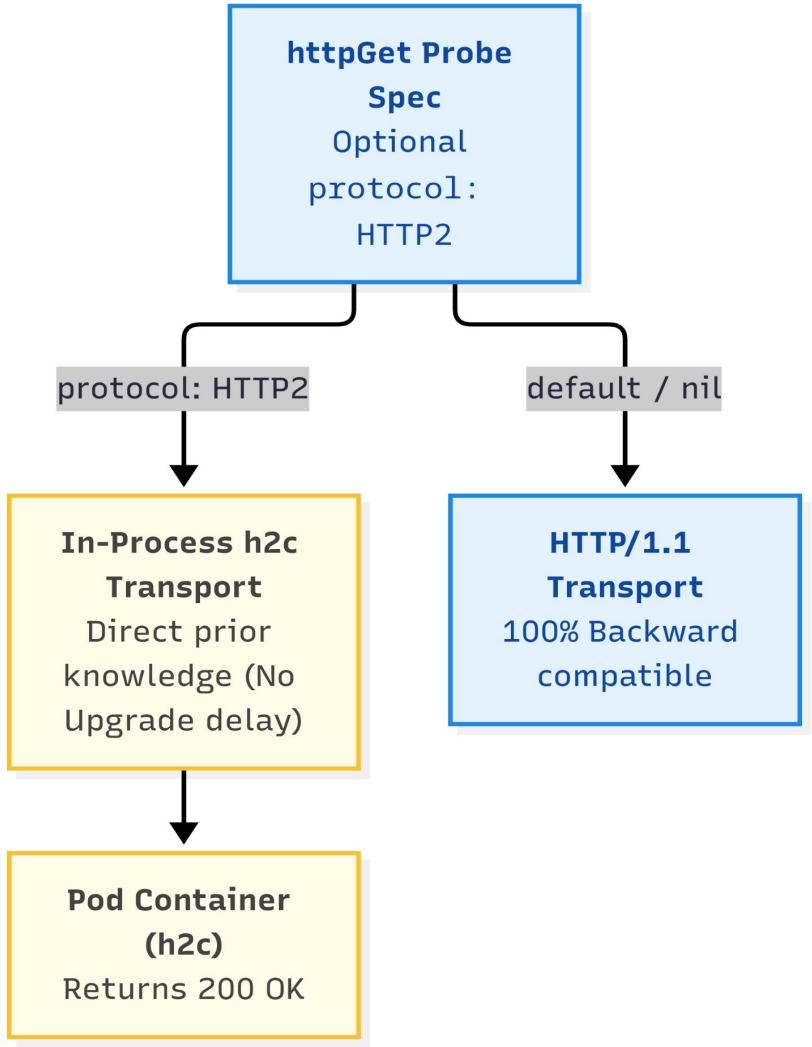
Beyond End-to-End TLS:- The Cleartext HTTP/2 (h2c) Dilemma

- **In-Cluster Reality:** Ingress terminates TLS at the edge, routing cleartext HTTP/2 (h2c) internally where application data traffic thrives.
- **Probe Deadlock:** httpGet is locked to HTTP/1.1 and grpc probes mandate Protobuf schemas, leaving internal h2c endpoints unprobeable.
- **Unacceptable Workarounds:** Forces a compromise between blind tcpSocket checks and memory-heavy dual listeners.



KEP-5999:- Completing the Kubernetes HTTP/2 Probing Matrix

- **API Enhancement:** Extends httpGet with an optional protocol: HTTP2 field without altering URI schemes or creating new probe handlers.
- **In-Process Prior Knowledge:** Uses Go's net/http configured with Protocols.SetUnencryptedHTTP2(true) to initiate direct HTTP/2 communication—bypassing slow HTTP/1.1 Upgrade negotiations.
- **100% Non-Breaking:** Defaults to HTTP1 when omitted, guaranteeing full backward compatibility for existing workloads.



gRPConf INDIA 2026

Thank You

Questions?

Neeraj Krishna Gopalakrishna

https://www.linkedin.com/in/neeraj-krishna/

Amritansh

https://www.linkedin.com/in/amrit1502/

