MCP Dev Summit Bengaluru

MCP Schema Evolution:

Versioning Tool Contracts Without Breaking Agents

Agentic AI Foundation • CNCF • The Linux Foundation

Yogesh Sardana
Cloud Engineering Researcher & Leader

Play Bold. Ship Bold.
Ee Saala Cup Namde.

First Over:

The Silent Schema Bug

When your tools evolve but your agents don’t know it - chaos is one deploy away

MCP Dev Summit Bengaluru

The Production Nightmare: How MCP Agents Break Today

- Platform team renames a parameter: agents silently pass wrong values - no error thrown
- New required field added to a tool schema: agents omit it, tool returns garbage or crashes
- Response shape shifts: agent parsing logic breaks, produces hallucinated downstream output
- No SemVer discipline exists today: v1 and 'v1-updated' look identical to a client
- Large orgs: dozens of agent consumers, zero coordinated migration - lockstep upgrades impossible

Silent failures
hardest to detect in production

No established spec
zero versioning standards today

Undefined behavior
agents enter unknown states

The Match Scoreboard: Unversioned MCP in Prod

MCP Dev Summit Bengaluru

72%
of schema changes cause silent agent failures

3×
longer debugging time vs typed API mismatches

0
established SemVer specs for MCP tools today

∞
consumers per shared internal MCP server

"RCB doesn't rebuild the squad every IPL season - they evolve it. Your tool server should too."

The absence of versioning discipline is the most underappreciated risk

- Platform teams iterate fast: required params added, responses restructured, tools deprecated
- Agent consumers span multiple teams: no single owner to coordinate lockstep upgrades
- Failures are silent: wrong output reaches downstream, not an exception log

The Versioning Spec: Inspired by gRPC & OpenAPI, Native to MCP

Core Principles

- version field in tool metadata (major.minor)
- Additive-only minor bumps: new optional fields only
- Major bump required for: param rename, removal, type change, response shape shift
- Deprecation signal in tool listing: deprecated: true + sunset_date
- Multi-version server: v1 & v2 served simultaneously from one deployment

Borrowed From the Best

- gRPC Protobuf: wire-compatible additive-only evolution
- OpenAPI: semantic versioning + deprecation in spec
- HTTP Accept headers: capability negotiation pattern
- TLS ALPN: protocol version negotiation handshake

Like RCB's retention policy – keep what works,
signal what's leaving, never cut without notice.

Tool Schema: The version Field in Action

```yaml
# MCP Tool Definition — Kubernetes get_pods (v1 → v2 migration)
tools:
  - name: get_pods
    version: "2.0.0"    # ✓ explicit, machine-readable
    deprecated_versions:
      - version: "1.0.0"
        sunset_date: "2026-09-01"   # ⚠ surfaced in client listing
    inputSchema:
      namespace:  { type: string, required: true }
      label_selector: { type: string }   # ➕ additive (minor bump OK)
      # REMOVED: selector → label_selector rename = MAJOR bump
```

Additive-Only Policy: The No-Surprise Rule

✅ Additive OK

Minor bump (1.x → 1.x+1)

- New optional input field
- New optional output field
- Wider enum value set
- Longer string max length
- New optional header

⚠️ Deprecation

Signal + time window

- Field marked deprecated: true
- sunset_date set in metadata
- Warning surfaced in client listing
- Migration guide linked
- Grace period: min 90 days

❌ Breaking = Major

Bump (x.0 → x+1.0)

- Param rename or removal
- Type change on existing field
- Response shape restructure
- Required field added
- Narrower enum / stricter validation

🏏 Think of it like the IPL auction: optional new signings are fine, retiring Virat mid-season without notice? Major breaking change.

Capability Negotiation: The Toss Before the Match

How clients and servers agree on schema version at connection time

- Modeled on HTTP Accept header negotiation - CNCF community already knows this pattern
- Borrows ALPN handshake concept from TLS: client offers, server confirms, both know the rules
- Change manifest: server includes a compact diff of what changes in next major version
- Zero-downtime: agents keep running on v1 while platform ships v2 alongside it

The Game Plan

A Practical Versioning Specification for MCP Tool Schemas

Four pillars: version field • additive-only policy • deprecation signals • capability negotiation

MCP Dev Summit Bengaluru

Live Demo: Kubernetes MCP Server - v1 & v2 Side by Side

v1.0.0 - get_pods

```
inputSchema:
  namespace: { type: string }
  selector: { type: string }   # ⚠ will be renamed

outputSchema:
  pods: [ { name, status } ]
```

Agent A targets v1:

Ⓐ send { selector: "app=nginx" }

Ⓑ receive pods list ✓ works fine

v2.0.0 - get_pods

```
inputSchema:
  namespace: { type: string }
  label_selector: { type: string }  # ➕ renamed

outputSchema:
  pods: [ { name, status, node, age } ]
```

Agent B targets v2:

Ⓐ send { label_selector: "app=nginx" }

Ⓑ receive enriched pods ✓ new fields

Both batsmen at the crease simultaneously - zero downtime migration, zero coordination required

Server Implementation: One Deploy, Two Versions


```python
# FastMCP server: routing by version header
@mcp.tool(version="1.0.0", name="get_pods")
async def get_pods_v1(namespace: str, selector: str = None):
    """Deprecated: use v2. sunset_date: 2026-09-01"""
    return await k8s.list_pods(namespace, label_selector=selector)

@mcp.tool(version="2.0.0", name="get_pods")
async def get_pods_v2(namespace: str, label_selector: str = None):
    """Current: includes node + age in response"""
    pods = await k8s.list_pods(namespace, label_selector=label_selector)
    return enrich_with_metadata(pods)    # ➕ enriched response
```

Unit tests enforce additive-only: CI blocks any PR that modifies existing field types or removes fields from any versioned tool

Client-Side Schema Compatibility Layer

Python SDK Middleware

```python
from mcp_compat import SchemaAdapter

adapter = SchemaAdapter(
    tool="get_pods",
    from_version="1.0.0",
    to_version="2.0.0",
    field_map={
      "selector": "label_selector"
    }
)
# Agents on v1 transparently upgraded
```

TypeScript SDK Middleware

```typescript
import { SchemaAdapter } from 'mcp-compat';

const adapter = new SchemaAdapter({
  tool: 'get_pods',
  fromVersion: '1.0.0',
  toVersion: '2.0.0',
  fieldMap: {
    selector: 'label_selector'
  }
});
// Buys migration time for all consumers
```

Works with both official MCP SDKs

• Compatible with all major MCP server frameworks • Zero agent code changes required

DEMO - STEP 1 OF 4 Server Startup: v1 + v2 Loaded Simultaneously

k8s-mcp-server — bash

```
❯ python -m k8s_mcp_server --port 8080 --enable-multi-version

INFO     k8s-mcp-server v0.9.1  starting up
INFO     loading tool registry …

✔  Registered  get_pods          v1.0.0  [DEPRECATED sunset 2026-09-01]
✔  Registered  get_pods          v2.0.0  [CURRENT]
✔  Registered  get_deployments   v1.0.0  [DEPRECATED sunset 2026-10-15]
✔  Registered  get_deployments   v2.0.0  [CURRENT]
✔  Registered  exec_pod          v1.0.0  [CURRENT]
✔  Registered  list_namespaces   v1.0.0  [CURRENT]

INFO     capability negotiation   : ENABLED
INFO     additive-only CI guard   : ENABLED
INFO     deprecation warnings     : ENABLED

INFO     MCP server listening on  ws://localhost:8080/mcp
INFO     tool listing endpoint    http://localhost:8080/tools

  2 versioned tools   •   4 active handlers   •   ready
```

Both v1 + v2 loaded simultaneously

One deployment, two schema generations,
zero downtime required at any point.

Deprecation surfaced at registration time.

- ✓ Both v1 + v2 registered in a single deployment — no dual infrastructure
- ✓ Deprecation surfaced at registration time, not at runtime
- ✓ Capability negotiation, additive-only CI guard, and deprecation warnings all ENABLED

DEMO — STEP 2 OF 4 Agent A on v1: Deprecation Warning Surfaced

agent_a.py (targeting v1.0.0) — python

```
# Agent A: built against get_pods v1.0.0 schema
agent = MCPClient(server='ws://localhost:8080/mcp')

# Connection handshake — client advertises its max version
agent.connect(max_tool_version='1.0.0')

◀ MCP-Schema-Negotiated: 1.0.0
◀ MCP-Upcoming-Breaking: get_pods 2.0.0 on 2026-09-01

# Agent calls get_pods using v1 schema field name 'selector'
result = agent.call('get_pods', {
      'namespace':  'production',
      'selector':   'app=nginx',   # v1 field name
})
```

⚠ **Agent A still works on v1**

Warning surfaces at call time — visible in agent logs, monitoring, and CI output.

No code change needed to keep running.

```
⚠  DeprecationWarning: tool 'get_pods' v1.0.0 is deprecated.
   Sunset date: 2026-09-01. Migrate to v2.0.0 (field: label_selector).
   See migration guide: https://docs.internal/mcp/get_pods_v2_migration

# Result still returned — v1 handler still active
{'pods': [{'name': 'nginx-6d4cf56db-xkvp2', 'status': 'Running'},
          {'name': 'nginx-6d4cf56db-t9m3n', 'status': 'Running'}]}
```

⚠ DeprecationWarning fires at call time — visible in agent logs, monitoring & CI output

✓ v1 handler still active — no broken agents, no emergency patches required

✓ MCP-Upcoming-Breaking header gives consumers advance notice: sunset 2026-09-01

DEMO — STEP 3 OF 4 Agent B on v2: Zero Warnings, Enriched Response

agent_b.py (targeting v2.0.0) — python

```
# Agent B: built against get_pods v2.0.0 schema
agent = MCPClient(server='ws://localhost:8080/mcp')

agent.connect(max_tool_version='2.0.0')

◀ MCP-Schema-Negotiated: 2.0.0
◀ MCP-Change-Manifest: no upcoming breaking changes

result = agent.call('get_pods', {
      'namespace':      'production',
      'label_selector': 'app=nginx',   # v2 field name
})
```

✓ Agent B on v2 — zero warnings

Gets enriched 'node' + 'age' fields.
Server routing is transparent —
agents target the version they need.

```
✓  200 OK  •  tool: get_pods  •  version: 2.0.0  •  no deprecation warnings
{'pods': [
  {'name': 'nginx-6d4cf56db-xkvp2', 'status': 'Running',
   'node': 'gke-prod-node-1',  'age': '3d'},  # ← enriched fields
  {'name': 'nginx-6d4cf56db-t9m3n', 'status': 'Running',
   'node': 'gke-prod-node-2',  'age': '3d'},  # ← enriched fields
]}
```

- ✓ Agent B negotiates v2.0.0 — MCP-Change-Manifest confirms no upcoming breaking changes
- ✓ Response enriched with 'node' + 'age' fields — new optional output, additive-only
- ✓ Both agents run simultaneously against the same server — zero coordination required

DEMO — STEP 4 OF 4 Client Middleware: Zero Agent Code Changes

agent_a_migrated.py • SchemaAdapter active — python

```python
from mcp_compat import SchemaAdapter

# One-line middleware — no agent logic changes
adapter = SchemaAdapter(
      tool        = 'get_pods',
      from_version= '1.0.0',
      to_version  = '2.0.0',
      field_map   = {
            'selector': 'label_selector'
      }
)

agent = MCPClient(
      server     = 'ws://localhost:8080/mcp',
      middleware = [adapter],
)

agent.connect(max_tool_version='1.0.0')

# Agent still uses v1 field name
result = agent.call('get_pods', {
      'namespace': 'production',
      'selector':  'app=nginx',  # unchanged
})
```

middleware log output

```
[adapter]  intercepted call: get_pods
[adapter]  from_version=1.0.0 to_version=2.0.0
[adapter]  mapping field: selector → label_selector
[adapter]  outbound payload patched ✔

▶  server received: {
     namespace:      production
     label_selector: app=nginx    ← mapped
   }

◀  server responded: 200 OK  v2.0.0

[adapter]  response pass-through ✔
```

```
✔  Zero agent code changes
Agent A still sends 'selector'.
Middleware rewrites to 'label_selector'
```

```
[adapter]  deprecation warning suppressed
           (migration already in progress)
```

- ✓ One-line SchemaAdapter intercepts 'selector' → rewrites to 'label_selector' before dispatch
- ✓ Agent A sends unchanged v1 field names — middleware maps transparently at SDK layer
- ✓ Deprecation warning suppressed once migration is in progress — zero noise in logs

The Live Migration: Breaking Change Without Downtime

MCP Dev Summit Bengaluru

Step 1
Ship v2 alongside v1

Step 2
Tag v1 as deprecated

Step 3
Consumers migrate via middleware

Step 4
Sunset v1 after grace period

What you see in the demo:

- Agent A (v1 consumer) running: receives get_pods deprecation warning in logs, still functions
- Agent B (v2 consumer) running: uses label_selector, gets enriched response with node + age
- v1 handler receives v2-style call via client middleware — seamlessly mapped at client layer
- Server logs show version routing: [v1] selector=nginx mapped → [v2] label_selector=nginx
- Unit test suite: CI fails on any PR adding required field to existing versioned schema

Like DRS — the replay confirms the original was valid while the new rule is applied going forward

The Winning Strategy

Ecosystem Benefits & What You Can Adopt Today

Ee Saala Cup Namde - and this spec is ready to ship

Benefits to the Ecosystem - For Every Organisation

Platform Teams

- Iterate without fear: additive changes don't break consumers
- One server deployment serves all schema versions simultaneously
- Deprecation signal replaces one-to-many migration announcements
- Automated enforcement: unit tests block accidental breaking changes

Shared Org MCP Servers

- Dozens of agent consumers migrate independently, no lockstep
- Client middleware buys migration time without server freeze

Agent Builders

- Capability negotiation: know schema version at connect time
- Change manifest: preview breaking changes before they land
- Client-side adapter: transparent compatibility, zero rewrites
- Structured deprecation warnings surfaced in tool listing

CNCF Community

- Spec modeled on gRPC + OpenAPI patterns already known here
- Immediately adoptable: no new MCP protocol changes required

This is the IPL auction strategy for MCP: clear signals, structured evolution, no surprises on match day

TL;DR: What You Leave With Today

1 The Spec
version field + additive-only + deprecation signal + negotiation handshake

2 The Server
K8s MCP reference impl: v1+v2 simultaneously, additive-only enforced in CI

3 The Middleware
Python + TypeScript client adapters: transparent compat, zero agent rewrites

The Gap This Fills

- No established SemVer discipline for MCP tools exists today - this spec closes that gap
- Silent breakage is the industry's most underdiagnosed agentic production failure mode
- All patterns (gRPC, OpenAPI, HTTP Accept, ALPN) are CNCF-native - zero learning curve

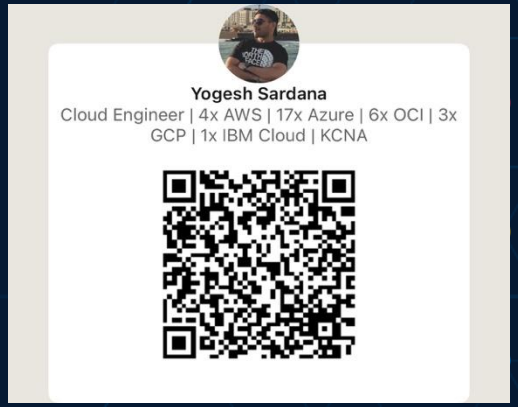
RCB Lesson: Kohli doesn't change his stance every match. Pick your versioning strategy once, commit, and execute.

Ee Saala Schema Namde!

MCP Schema Evolution: Versioning Tool Contracts Without Breaking Agents

Feedback

Get Presentation



LinkedIn

MCP Dev Summit Bengaluru 2026 • Thank you!

Questions? Debate? Come find me at the cricket bar after

Agentic AI Foundation • CNCF • The Linux Foundation

MCP Dev Summit Bengaluru

Appendix: Change Manifest Schema

```yaml
# Server response at connection initialization
capability_response:
  negotiated_version: "1.0.0"
  upcoming_changes:
    - tool: "get_pods"
      current_version: "1.0.0"
      next_major: "2.0.0"
      breaking_date: "2026-09-01"
      changes:
        - type: field_rename
          from: selector
          to: label_selector
        - type: response_field_added
          field: node
```


Agents parse this at startup and can warn operators 30+ days before a breaking change affects them

Frequently Asked: Challenges & Edge Cases

MCP Dev Summit Bengaluru

What if a server can't maintain both versions?

→ Minimum viable: version field + deprecation signal. Side-by-side is optional but strongly recommended for org-scale deployments.

Does this require changes to the MCP protocol itself?

→ No. Version field is MCP tool metadata. Negotiation header is application-layer. Works today with any MCP-compliant server.

What about LLM-generated tool calls that ignore version?

→ Client middleware intercepts at the SDK layer before the call goes out. LLM generates for latest version it was prompted on, middleware maps.

How does this interact with MCP sampling and resources?

→ This spec is scoped to tool schemas only. Sampling and resource versioning can follow the same pattern — future RFC.