Building an MCP Marketplace

Lessons in Discovery, Versioning, and Trust

Hello!

- **Google Developer Expert**
  GCP
- **DevOps Ambassador**
  Institute Ambassador and Organizer
- **Community Leader**
  Graph DB Pune & Kong Champion Organizer
- **Multi-Cloud Advocate**
  Calico Big Cats, CDF & open-appsec AMB

Saurabh Mishra

Connect with me

linkedin.com/connectsaurabhmishra
medium.com/@connectsaurabhmishra

Agenda

Marketplace Foundations
MCP marketplace fundamentals
Discovery challenges & Economics

Lifecycle & Evolution
Versioning strategy
Future directions

Governance
Trust and governance mechanisms

Technical Deep Dive
Reference architectures
Integration patterns

Key Takeaways
Lessons in trust & discovery

Closing
Open Q&A Session

What is MCP?

Model Context Protocol (MCP)

Definition

An open protocol for connecting AI models with external tools, data sources, and applications, standardizing communication between AI clients and servers.

Key Benefits

- Interoperability & Reusability
- Tool discoverability
- Standardized messaging
- Scalable integrations

Typical Components

• MCP Client & Server
• Tools, Resources & Transport Layer

The Rise of MCP

Unified Connections

The Model Context Protocol establishes an open-source standard enabling AI assistants to securely connect directly to data, tools, and custom environments.

It solves the costly `N x M` integration problem, operating essentially like the **'USB-C port'** for AI models.

Production-Ready

Designed for scale over one-off runs. Enables:

- Persistent connections & state management
- Standardized error-handling schemas
- Cross-framework composability

An ecosystem is now scaling incredibly fast, demanding robust marketplaces to govern these connections.

Why MCP Marketplaces Matter

AI agents are evolving from isolated copilots into interconnected ecosystems

Agents increasingly need:

- Tools
- APIs
- Memory systems
- Data connectors
- Workflows

Standardized Interoperability

MCP enables standardized interoperability across the ecosystem.

Marketplace Layer

The next layer: discovery and management of MCP servers.

The Problem We're Solving

Challenges Without a Marketplace

- Discovery is fragmented
- Trust is unclear
- Version compatibility breaks workflows
- Enterprises struggle with governance
- Developers duplicate integrations repeatedly

Key Question

How do we build an ecosystem where agents can safely discover, trust, and compose capabilities?

What is an MCP Marketplace?

Defining the Marketplace

An MCP Marketplace is:

- A registry of MCP-compatible services/tools
- A discovery layer for agents and developers
- A trust and governance framework
- Versioning and compatibility management
- A monetization and ecosystem layer

Core Marketplace Components

Registry Layer
Tool metadata, Schemas, Capabilities, Auth, Permissions

Discovery Layer
Semantic search, Capability matching, Recommendations, Optimization

Runtime Layer
Invocation, Authentication, Rate limiting, Observability, Sandboxing

Governance Layer
Verification, Reputation, Security reviews, Compliance

Discovery: The First Hard Problem

Why Discovery is Difficult

Traditional app stores are built for humans. MCP marketplaces are built for:

- Humans
- AI agents
- Multi-agent orchestration systems

Discovery challenges:

- Ambiguous capabilities
- Overlapping tools
- Inconsistent metadata
- Rapid ecosystem growth
- Context-dependent relevance

Human vs. Agent Discovery

Human-Centric	Agent-Centric
UI browsing	Capability resolution
Ratings & Reviews	Reliability metrics
Natural descriptions	Structured schemas
Screenshots	Invocation contracts
Experience reviews	Execution outcomes

Capability Graphs vs. Categories

Traditional categories fail because MCP tools are composable. We propose a new model.

Proposed Model

- Capability graph
- Semantic relationships
- Intent-based discovery
- Dependency awareness

Example Execution Path

"Generate quarterly sales report"

Resolves into:

- CRM connector
- SQL query engine
- Spreadsheet tool
- Visualization server
- PDF export service

Why Graph Databases for MCP Marketplaces?

The Problem

- MCP tools are interconnected
- Dependencies are complex
- Compatibility spans multiple dimensions
- Trust signals are relationship-based

Why Neo4j

- Models capabilities as graphs
- Traverses relationships in real time
- Enables intent-based discovery
- Supports agent-driven tool composition

Key Message: MCP Marketplaces are graphs, not catalogs.

Metadata is the Product

Key Dimensions

- Input/output schema
- Latency profile
- Cost model
- Security scope
- Reliability score
- Supported models
- Version compatibility
- Observability hooks

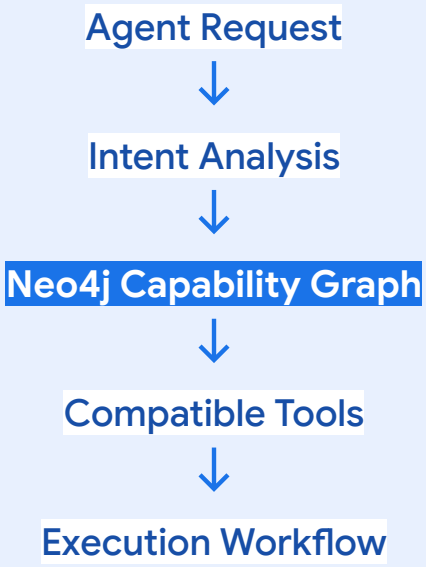
Key Insight

"Poor metadata destroys discoverability."

Lessons Learned in Discovery

- Free-form descriptions are insufficient
- Structured capabilities outperform tags
- Semantic embeddings improve search quality
- Runtime telemetry improves ranking
- Agent feedback loops matter more than star ratings

Neo4j-Powered Capability Discovery



- Semantic search finds candidate tools
- Neo4j resolves complex dependencies
- Compatibility via graph traversal
- Best execution path is recommended

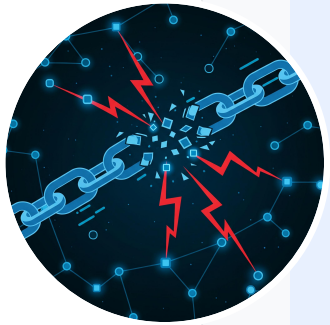
This directly supports the discovery narrative.

Why Versioning Breaks Agent Ecosystems

Traditional Software

Humans adapt to changes

Flexible, resilient, but slow to scale.



Agent Ecosystems

Agents rely on deterministic contracts

Small changes cascade through workflows

Silent failures are catastrophic

Compatibility Graphs

Relationship Model

```
Model ——COMPATIBLE_WITH—— Tool
Tool ——DEPENDS_ON—— Tool
Tool ——REQUIRES—— Policy
Tool ——SUPPORTED_BY—— Runtime
```

Key Benefits

- Impact analysis before upgrades
- Dependency visualization
- Automated compatibility checks
- Rollback planning

This is a very strong enterprise use case.

Semantic Versioning is Necessary — But Not Enough

Traditional Software

Standard semver helps manage releases:

MAJOR . MINOR . PATCH

Focuses on API stability and backward compatibility.

Agent Ecosystems Also Need:

- Capability versioning
- Behavioral guarantees
- Safety versioning
- Policy versioning
- Prompt contract versioning

Compatibility Matrices

The Marketplace must track multi-dimensional compatibility for reliable agent workflows:

Model ↔ Tool compatibility

Tool ↔ Tool interoperability

Agent ↔ Runtime compatibility

Enterprise policy constraints

Sandboxed Rollouts

The following rollout model is recommended for safe agent deployment:

- Canary release
- Agent simulation testing
- Compatibility scoring
- Gradual rollout
- Rollback support

Lessons Learned in Versioning

Backward compatibility becomes existential

Behavioral drift matters as much as schema drift

Observability is critical for debugging

Marketplace-level testing infrastructure is essential

Runtime fingerprints help diagnose failures

Trust: The Core Currency of Agentic Ecosystems

In agent ecosystems, tools have significant capabilities and risks:

- Access sensitive data
- Trigger actions
- Spend money
- Execute workflows autonomously

Trust failures are systemic failures. Trust is more important than features.

The Trust Stack

Identity

Verified publishers

Cryptographic signing

Organizational ownership

Security

Sandboxed execution

Scoped permissions

Vulnerability scanning

Secrets isolation

Reliability

Uptime metrics

Latency SLAs

Failure rates

Audit logs

Governance

Compliance certifications

Policy enforcement

Human approval workflows

Reputation Systems for Agents

Traditional ratings are insufficient.

Useful signals for evaluation:

- Successful execution rate
- Recovery behavior
- Hallucination resistance
- Policy compliance
- Cost predictability
- Observed side effects

Permissioning Models

Marketplace should support:

- Human-in-the-loop approvals
- Scoped access tokens
- Session-based permissions
- Least privilege execution
- Policy-aware invocation

Enterprise Trust Requirements

Enterprises require specialized controls for deployment:

- **Auditability**
- **Data residency**
- **Access governance**
- **Vendor verification**
- **Risk scoring**
- **Incident response visibility**

Lessons Learned in Trust

Continuous Verification
Verification must be continuous to maintain ongoing integrity.

Behavioral Merit
Runtime behavior matters more than static claims.

Machine-Readable Reputation
Reputation scores should be programmatically accessible.

Governance APIs
Enterprises need dedicated APIs for robust access governance.

Compounding Trust
Trust compounds over time through deep observability and transparency.

Types of Versioning Problems

API Contract Changes

- Schema changes
- Parameter renames
- Output structure modifications

Behavioral Changes

- Different reasoning behavior
- Different latency patterns
- New side effects

Dependency Drift

- Tool chains become incompatible
- Nested MCP dependencies break

Economics of MCP Marketplaces

Stakeholders

- Tool developers
- Agent developers
- Platform operators
- Enterprises
- End users

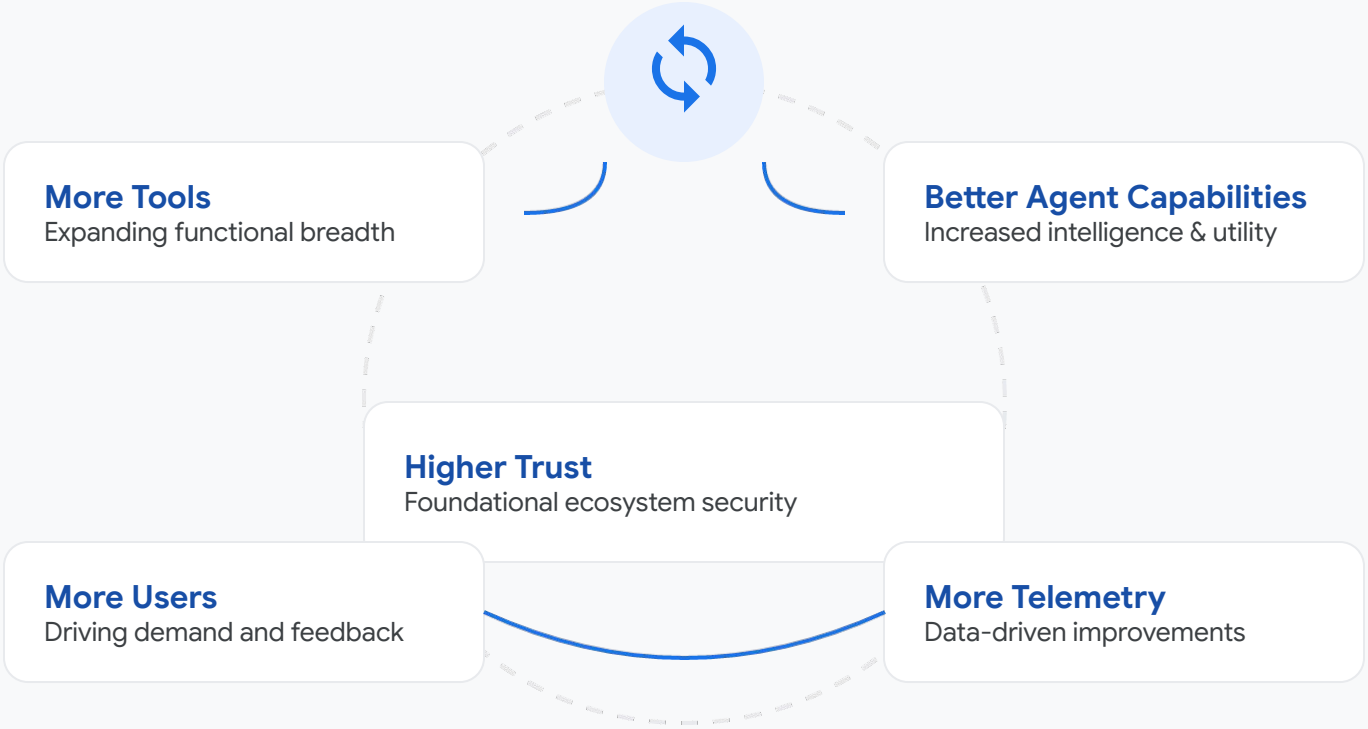
Strategic Questions

- Who captures value?
- How are capabilities monetized?
- How do we avoid ecosystem fragmentation?

Possible Monetization Models

- Usage-based billing
- Revenue sharing
- Premium verified listings
- Enterprise governance tiers
- Compute marketplace fees
- Subscription bundles

The Network Effect Flywheel



Architecture Patterns for MCP Marketplaces

- Registry service
- Metadata index
- Discovery engine
- Authentication broker
- Execution runtime
- Observability pipeline
- Policy engine
- Reputation system

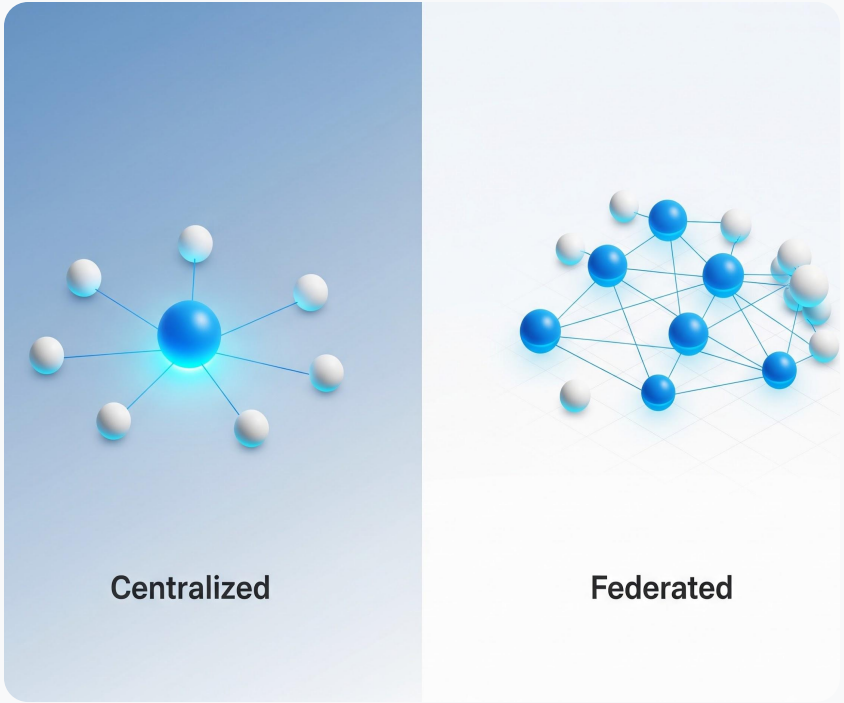
Centralized vs Federated Marketplaces

Centralized Model
Easier governance & strong consistency

Federated Model
Greater openness & more resilient

Market Dynamics
Platform control vs ecosystem diversity

Innovation Speed
Curated quality vs rapid decentralized growth



Open Standards Matter

Critical Standards

- Capability schemas
- Invocation contracts
- Trust attestations
- Policy metadata
- Audit event formats

Key Insight

Open ecosystems grow faster than closed ecosystems.

The Long-Term Vision for AI Ecosystems

The Long-Term Vision

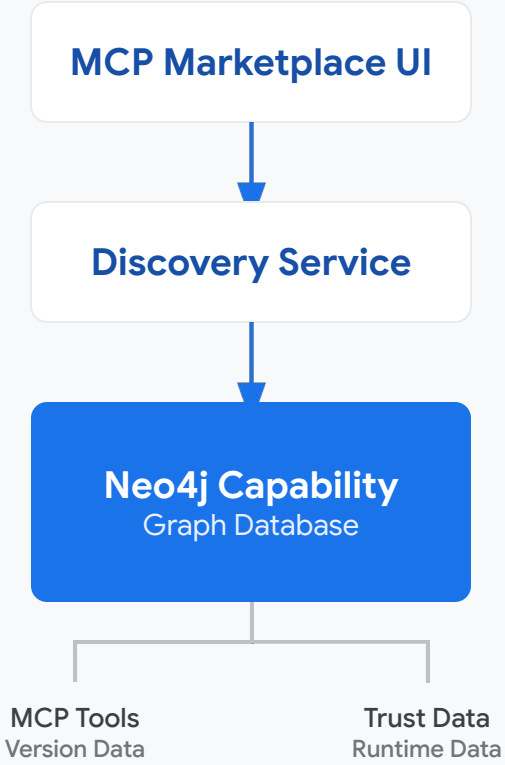
- Agents dynamically discover capabilities
- Trust is machine-verifiable
- Workflows self-optimize
- Enterprises define policy boundaries
- Marketplaces as the AI operating system

Key Takeaways & Core Lessons

Ecosystem Discovery & Trust

- **Discovery:** Metadata quality and semantic graphs drive usability.
- **Versioning:** Behavioral compatibility and observability are infrastructure.
- **Trust:** The core currency; governance must be platform-native.
- **Design:** Open standards accelerate network effects.

System Architecture: Discovery & Graph Core



Neo4j Stores

- Capabilities
- Dependencies
- Trust Relationships
- Version Compatibility
- Usage Telemetry

The Future of AI Ecosystems

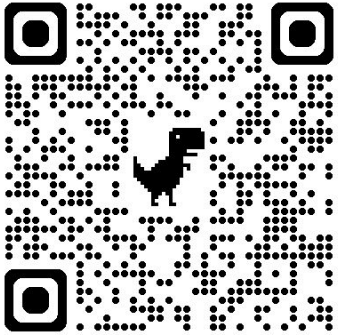
Final Thought

The future of AI is not a single model. It is an ecosystem of interoperable agents, tools, and services.

MCP marketplaces will define how those ecosystems discover, trust, and coordinate with each other.

Thank You!

Questions & Discussion



Contact: skm.jss@gmail.com