Declarative MCP Servers for Secure, Specialized AI Agents

From Tool Sprawl to Specialization: MCP Best Practices at Scale

Josh Reini • Developer Advocate | Reetika Roy • Staff Software Engineer

Snowflake

MCP Dev Summit 2026

The Problem: Tool Sprawl

- The more tools an agent can reach, the less predictable it becomes
- Every additional tool increases context window size and token cost
- Broad access = larger security blast radius when things go wrong
- Hallucination risk rises with tool count — the model has more ways to be wrong

Tool sprawl is the default failure mode for enterprise agents.

Why Sprawl Happens

- No capability boundaries — agents inherit every tool available in the environment
- Teams add tools to shared servers without considering the aggregate surface area
- Prompts are not governance — "only use these tools" is a suggestion, not a constraint
- No distinction between what a server offers and what a user should be allowed to invoke

God Server

One MCP server with every tool

Prompt Governance

"Only use tools X, Y, Z"

Shared Creds

All agents use one service account

If your only boundary is a system prompt, you don’t have a boundary.

ATTACK 50 Tools Means 50 Ways to Exfiltrate

- A sprawled agent has read access to HR, finance, infra, and customer data simultaneously
- Prompt injection can steer tool calls to any table the agent can reach
- "Read-only" is not a security boundary — the agent can still SELECT secrets, PII, credentials
- Exfiltration vector: tool output gets summarized back into an attacker-controlled context

```
-- Agent has access to everything
-- Injected via prompt:
SELECT ssn, salary, email
FROM hr.employees
WHERE department = 'Executive';

SELECT card_number, cvv
FROM billing.payment_methods;
```

Broad access is the root cause. The agent queries everything because it can.

DEFENSE Make the Agent a First-Class Identity

- Assign the agent its own identity — not the user's, not a shared service account
- Data movement and security policies scoped to the agent's role
- Least privilege: the agent should only see what the current task requires
- Audit every action against the agent's identity, not a generic account

Agent Identity

Distinct principal with auditable actions

Column Policies

Mask SSN, salary, PII at the query layer

Row-Level Security

Limit rows to the agent's authorized scope

Role-Based Access

Permissions tied to agent role, not just user

Identity + policies = defense in depth. Neither alone is enough.

The Specialization Thesis

Scaling agentic workflows safely requires specialization at every level—agents, servers, tools, and data—not just better prompts.

Two Independent Boundaries

1 Agent Specification

Defines which servers, tools and instructions each agent has access to

2 MCP Server Definition

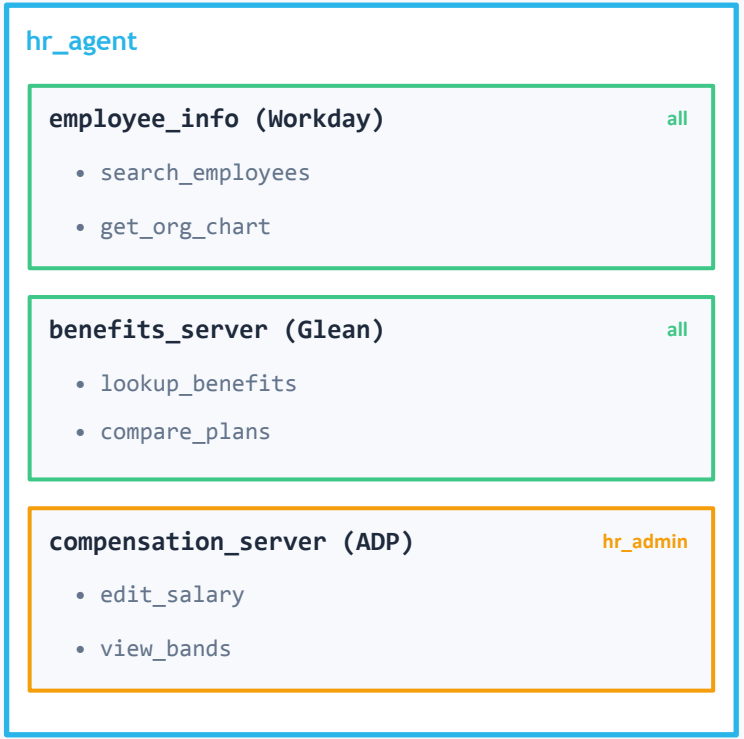
Defines which specialized tools are available through the server

RBAC is applied at every level: server, agent, tool & data.

Together, these boundaries reduce sprawl while improving operability.

Boundary 1: Agent Specification

- Each agent has its own spec: mcp servers + tools + instructions
- RBAC controls which mcp servers and tools can be invoked
- Same agent, multiple servers, each with least-privilege tool access



Each agent is a scoped contract: these tools, this instruction set, for this role.

Boundary 2: MCP Server Definition

- An MCP server specification can be even more granular than the domain boundary
- The YAML spec declares tools: Cortex Agents, Search, Analyst, SQL exec, or custom UDFs
- Servers are governed objects: versioned, auditable, with RBAC on USAGE
- Each tool requires its own privilege grant — access to the server ≠ access to tools

```
CREATE MCP SERVER hr_server
  FROM SPECIFICATION $$
    tools:
      - name: "handbook-search"
        type: CORTEX_SEARCH_SERVICE_QUERY
        identifier: hr.search.handbook_search_svc
      - name: "compensation-analyst"
        type: CORTEX_ANALYST_MESSAGE
        identifier: hr.agents.benefits_agent
  $$;
```

The server declaration is a domain boundary: these tools, for these roles, nothing else.

Both Boundaries in Action

hr_agent

employee_info_server
- search_employees
- get_org_chart

benefits_server
- lookup_benefits
- compare_plans

compensation_server
- edit_salary
- view_bands

finance_agent

reporting_server
- run_report
- get_forecast

budget_server
- view_budget
- approve_spend

RBAC at Every Level

Agent	Which agents can this role invoke?
MCP Server	Who can connect to the server at all?
Tool	Which tools within the agent are authorized?
Data	Row-level & column-level policies on query results

Two boundaries define structure. RBAC enforces access at every layer.

Coding agents can automate the RBAC setup

Demo

Benefits: Predictability & Cost

- Fewer tools per agent = fewer hallucinations — the model has a smaller decision space
- Token costs drop because each agent only carries its own tool descriptions
- Specialized agents are easier to reason about, test, and debug
- Failure blast radius is contained: a misbehaving agent can’t escape its server

Predictability

Smaller tool set = more deterministic behavior

Lower Cost

Fewer tool descriptions in every prompt

Debuggability

Scoped agents are easier to trace and test

Containment

Failures stay within the agent’s domain

Specialization isn’t just a security pattern. It’s an operability pattern.

Benefits: Governance at Scale

- RBAC at every level: server access, agent invocation, tool authorization, data policies
- Declarative definitions are reviewable, diffable, auditable — like infrastructure as code
- Teams own their servers and agent specs; admins own the RBAC policies
- Applicable across hosting models: multi-tenant, single-tenant, hybrid

Multi-Team

Teams own servers + agent specs

Auditable

Server & agent definitions = version-controlled

4-Layer RBAC

Server → agent → tool → data

Multi-Tenant

Works across hosting models and orgs

MCP servers as governed objects turns agent deployment into a manageable platform problem.

Key Takeaways

1. Tool sprawl is the default failure mode — fight it with structure
2. RBAC at every level: server, agent, tool, and data
3. Specialization improves predictability, cost, and debuggability
4. Treat MCP servers + agent specs as governed objects — infrastructure as code

Josh Reini

Developer Advocate

Reetika Roy

Staff Software Engineer

Snowflake