TECHNICAL ENABLEMENT ASSIGNMENT

MCP Integration Lab
Design for Sales Engineers

Agent calling a mock enterprise search API via Model Context Protocol

Prepared by John Yzaguirre
Role Technical Enablement Manager
Duration 60-90 minute hands-on lab

PART 1 — LAB OUTLINE

Lab overview

Title, audience, prerequisites, and what SEs will be able to do after completing this lab.

Title

MCP Integration Lab: Agent calling a mock enterprise search API via MCP

A hands-on lab where Sales Engineers wire an AI agent to a mock enterprise search endpoint using Model Context Protocol, then validate grounded, citation-backed answers.

Audience

Technical presales: SEs, SAs, and AIOMs who are comfortable with HTTP APIs and basic cloud concepts, but may be **new to MCP** and this specific integration pattern.

Prerequisites

- Basic terminal/CLI use
- JSON literacy (read and write)
- Python familiarity (read, modify, run)
- AWS account access (provided)
- Conceptual understanding of LLMs

Format

- **Duration:** 60-90 minutes
- **Delivery:** Instructor-led, hands-on
- **Environment:** AWS (Lambda + CloudShell)
- **Language:** Python 3.12
- **Tools:** MCP Python SDK, MCP Inspector

> **Why "search" not "model"?**
> Enterprise search mirrors how customers actually use Glean: retrieve authoritative context first, generate grounded answers second. This is the integration story SEs need to tell.

Learning objectives

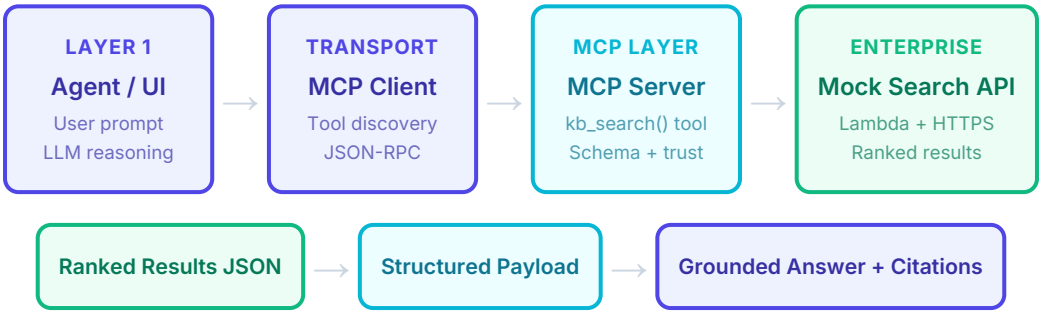
After this lab, each SE can:

1. **Explain MCP's mental model**—describe the client-server architecture, the three capability primitives (tools, resources, prompts), and where schemas live.
2. **Deploy a mock enterprise search API**—stand up a serverless endpoint on AWS Lambda with a realistic /search contract and structured metadata.
3. **Wire an MCP server tool**—implement a `kb_search(query, top_k)` tool using the official Python SDK that calls the endpoint and returns structured results.
4. **Validate grounded answers**—confirm the agent's response aligns with returned documents, includes citations, and respects permission boundaries.
5. **Troubleshoot integration failures**—use MCP Inspector, server logs, and CloudWatch to isolate faults across client, server, and endpoint layers.

ARCHITECTURE

Integration pattern overview

The three-layer architecture SEs will build and learn to explain to customers.



Response flow: enterprise data grounds the agent's answer with verifiable sources

Why this pattern matters for customers

Standard protocol

MCP replaces bespoke API integrations with a universal connector surface. One protocol, many enterprise systems.

Tool discovery

Agents discover available tools dynamically via MCP schema. No hardcoded integrations, just declare and connect.

Grounded answers

RAG pattern: retrieve authoritative context, then generate. Reduces hallucination, increases trust.

Enterprise governance

Permission-aware results, audit trails, OAuth-aligned auth. The agent only sees what the user is allowed to see.

MCP primitives covered in this lab

Primitive	What it does	Lab usage
Tools	Callable functions for API calls and side effects	`kb_search(query, top_k)`
Resources	Read-only data artifacts (file-like context)	Discussed, not implemented
Prompts	Reusable prompt templates exposed by server	Discussed, not implemented

The lab focuses on Tools because they are the most common primitive in customer integration conversations and map directly to "agent calls enterprise system."

LAB FLOW

High-level lab flow

Seven sections from customer context through hands-on wiring to a repeatable demo narrative.

1. **Intro and customer story** ~8 min

   Establish the narrative: "agents call enterprise systems via a standard protocol." Learners sketch the four-box architecture and understand where schemas and trust boundaries live. This sets up every customer conversation they will have.

2. **Cloud environment setup** ~10 min

   Open AWS CloudShell, pick a region, create an IAM role for Lambda, confirm logging works. Minimal setup, maximum time for the interesting parts. Done: learner can invoke a Lambda and find its CloudWatch log group.

3. **Deploy mock enterprise search API** ~15 min

   Implement the `/search` contract on Lambda with Function URL. Query in, ranked results out, with titles, snippets, relevance scores, and ACL groups. Deterministic data for reproducible validation. Done: `curl` returns structured JSON for sample queries.

4. **MCP server wiring** ~15 min

   Build an MCP server using the Python SDK (FastMCP). Define `kb_search(query, top_k)` as a tool, wire it to the mock endpoint, use stdio transport. Done: MCP Inspector discovers the tool and shows input schema plus output shape.

5. **End-to-end query validation** ~15 min

   Run a realistic SE scenario query ("What is our Q4 renewal process?"). Show the agent using the MCP tool, receiving results, producing a grounded answer with citations. Validate relevance, grounding, and permissions. Done: learner can explain where grounding happens.

6. **Troubleshooting and debugging** ~12 min

   Deliberately break one layer (wrong URL, schema mismatch, stdout corruption), then fix it. Teach the "client, server, endpoint" fault isolation pattern. Done: learner can isolate the fault domain in under five minutes.

7. **Wrap-up and customer positioning** ~10 min

   Convert lab experience into a repeatable talk track: "standard protocol," "tool discovery," "governance," "permissions." Done: learner can deliver a two-minute positioning narrative and a one-minute "how it works" explanation.

SECTIONS 1-2 DETAIL

Intro, customer story, and environment setup

Section 1: Intro and customer story

This is the most important section. Before touching any code, SEs need to internalize the customer narrative. Every technical decision in the lab ties back to a real customer conversation.

CORE ACTIVITIES

- Facilitator presents the enterprise integration challenge: "Your customer's agents need controlled access to real enterprise systems. How do you connect them without building a bespoke integration for every system?"
- Learners sketch the four-box architecture on paper or whiteboard: Agent → MCP Client → MCP Server → Enterprise API
- Discuss trust boundaries: where does the schema live? Who controls what the agent can call?

DONE LOOKS LIKE

Learner can draw the four boxes, label them, explain where tool schemas are defined, and articulate why a standard protocol matters over custom integrations.

Facilitator tip

Ask: "How many different APIs does your largest customer integrate with today?" This grounds the abstract protocol in real pain. MCP's value is proportional to the number of systems an agent needs to reach.

Section 2: Cloud environment setup (AWS)

Minimal infrastructure to maximize time on the interesting parts. AWS Lambda with Function URLs provides a dedicated HTTPS endpoint without requiring API Gateway.

CORE ACTIVITIES

- Open AWS CloudShell (pre-authenticated, no local install needed)
- Pick a region (us-east-1 for lowest latency in lab)
- Create an IAM execution role for Lambda with basic CloudWatch permissions
- Create a "hello world" Lambda function and invoke it to confirm the toolchain works
- Find the CloudWatch log group for the function

DONE LOOKS LIKE

Learner can invoke a Lambda function once and find its log output in CloudWatch. The infrastructure foundation is confirmed working.

Why AWS over GCP or Azure?

Lambda Function URLs provide a direct HTTPS endpoint without extra gateway configuration. CloudShell eliminates local CLI setup. The lean serverless footprint means minimal infrastructure overhead for a 60-90 minute lab. In a real deployment, any cloud provider works—MCP is cloud-agnostic.

SECTION 3 DETAIL

Deploy mock enterprise search API

The mock endpoint is designed to be simple but realistic—enough metadata to validate relevance, grounding, and permissions.

Search API

Mock /search contract

```
# Request
GET /search?q=quarterly+renewal+process&top_k=3

# Response
{
  "query": "quarterly renewal process",
  "results": [
    {
      "doc_id": "DOC-042",
      "title": "Q4 2024 Renewal Playbook",
      "snippet": "Standard renewal process begins 90 days ... ",
      "url": "https://wiki.acme.co/renewals/q4-2024",
      "score": 0.94,
      "acl_group": "sales-team"
    }
  ]
}
```

CORE ACTIVITIES

- Deploy the search Lambda with deterministic canned documents (5-8 docs across 2-3 query topics)
- Enable Lambda Function URL with auth type NONE (lab simplification; note production requires auth)
- Test with `curl` and confirm structured JSON response

DONE LOOKS LIKE

`curl` returns deterministic JSON for at least two different sample queries.

SECTION 4 DETAIL

MCP server wiring

Build an MCP server using the official Python SDK (FastMCP). One tool, one schema, stdio transport.

MCP server wiring

MCP server tool definition

```python
from mcp.server.fastmcp import FastMCP
import httpx, sys, json

mcp = FastMCP("enterprise-search")

@mcp.tool()
async def kb_search(query: str, top_k: int = 3) → str:
    """Search the enterprise knowledge base.
    Returns ranked results with titles, snippets, and permissions."""
    url = f"{ENDPOINT}/search?q={query}&top_k={top_k}"
    # Log to stderr, NEVER stdout (stdout = JSON-RPC)
    print(f"Calling: {url}", file=sys.stderr)
    resp = await httpx.AsyncClient().get(url)
    return json.dumps(resp.json())
```

Why stdio transport?

Reduces setup to zero—no HTTP server, no OAuth, no Origin validation. Keeps focus on tool schema and integration flow. Streamable HTTP is the production transport; we teach it conceptually.

Critical rule: stdout is sacred

For stdio servers, `stdout` must only contain JSON-RPC frames. All logging goes to `stderr`. This is the #1 real-world failure mode—we make it a teachable moment.

DONE LOOKS LIKE

MCP Inspector discovers the `kb_search` tool, shows the input schema (`query: str, top_k: int`), and the output JSON shape.

SECTION 5 DETAIL

End-to-end query validation

Section 5: End-to-end query validation

The moment of truth. Run a realistic SE scenario query and prove the integration works end-to-end.

CORE ACTIVITIES

- Run a scenario query: "What is our Q4 renewal process?"
- Trace the flow: user prompt → agent selects kb_search tool → MCP client sends JSON-RPC call → server calls mock API → results return → agent synthesizes answer with citations
- Validate three dimensions of "meaningful results":

Validation dimension	What to check	Why it matters
RELEVANCE	Top-k results match the query intent; scores are reasonable	Proves the search layer is working
GROUNDING	Agent's answer references returned documents, not free-text hallucination	Core RAG value proposition for customers
PERMISSIONS	Results include `acl_group`; agent respects access boundaries	Enterprise-grade governance story

DONE LOOKS LIKE

Learners can explain where grounding happens and point to which document snippet supported each claim in the agent's response.

End-to-end validation flow

SECTION 6 DETAIL

Troubleshooting and debugging

Section 6: Troubleshooting and debugging

Deliberately break one layer, then fix it. This builds the confidence SEs need to troubleshoot during live demos and customer POVs.

Failure scenarios to practice

Break this	Symptom	Debug with	Fix
Wrong endpoint URL	Tool call fails silently or returns error	MCP Inspector + server stderr	Correct the ENDPOINT variable
Schema mismatch	Tool discovered but wrong args	MCP Inspector schema tab	Align tool signature with expected schema
stdout corruption	Server "breaks randomly"	Check for print() without file=stderr	Redirect all logs to stderr
Lambda error (500)	Mock API returns error	CloudWatch logs	Fix Lambda handler code

> **Triage pattern: "client, server, endpoint"**
> Teach SEs to isolate fault domains in order: (1) Does MCP Inspector see the tool? If not, it's a **server** problem. (2) Does the tool call return data? If not, it's an **endpoint** problem. (3) Does the agent use the data correctly? If not, it's a **client/agent** problem.

DONE LOOKS LIKE

Learner can isolate the fault domain in under five minutes using the three-layer triage pattern and the right debugging tool for each layer.

Troubleshooting triage

SECTION 7 + SUCCESS CRITERIA

Wrap-up, positioning, and success criteria

Section 7: Wrap-up and customer positioning

Convert hands-on experience into customer-ready talk tracks. This is where lab knowledge becomes demo confidence.

CORE ACTIVITIES

- Each learner practices a **two-minute positioning narrative** with a partner: "Here's what MCP does, here's why it matters for your enterprise integration story"
- Each learner practices a **one-minute "how it works"** explanation using the architecture diagram
- Group discussion: "What's the production delta?" Cover auth, Origin validation, least privilege, prompt injection risk controls

DONE LOOKS LIKE

Learner can deliver both narratives unprompted, and articulate three things they would add for a production deployment.

The production delta conversation

SEs will be asked "Is this production-ready?" in every demo. Teach them to answer confidently: "The protocol is production-ready. For your environment, we'd add OAuth-aligned authorization, Origin header validation for HTTP transports, and least-privilege tool scoping. Let me show you what that looks like."

Overall success criteria

How the SE and the facilitator know the lab achieved its goals.

- ✓ **End-to-end success:** Each learner can run one query through MCP to the mock search API and receive a grounded, citation-backed answer.
- ✓ **Vocabulary fluency:** Each learner can describe the flow using correct terms—tool schema, JSON-RPC transport, stdio vs Streamable HTTP, MCP Inspector debugging surfaces.
- ✓ **Production awareness:** Each learner can articulate the production delta—auth, Origin validation, least privilege, prompt injection risk controls.
- ✓ **Customer readiness:** Each learner can deliver a two-minute positioning narrative connecting MCP to real customer integration pain.

Assessment approach

No formal test. Facilitator observes during pair exercises and the group discussion. Quick pulse check: "On a scale of 1-5, how confident are you demoing MCP to a customer tomorrow?" Target: 4+ average.

PART 2 — REASONING

Technical choices and trade-offs

How I approached the design of this lab, and why.

Technical choices

- **AWS Lambda + Function URLs:** Lean serverless footprint with zero gateway overhead. Function URLs provide a dedicated HTTPS endpoint without API Gateway configuration. CloudShell eliminates local CLI setup entirely—SEs open a browser tab and start building.
- **Mock "search" over "model":** Enterprise search mirrors the most common customer integration pattern: retrieve authoritative context, then generate grounded answers. This is how Glean positions workplace search and agentic workflows—the lab teaches the exact narrative SEs need.
- **One narrow MCP tool (`kb_search`):** One tool is enough to teach discovery, schema, invocation, and return payload interpretation. It keeps the lab focused while enabling natural discussion of multi-tool scenarios and tool filtering for follow-up sessions.
- **Official MCP Python SDK (FastMCP):** FastMCP provides a clean decorator-based API that matches how MCP tutorials introduce the protocol. It supports both stdio and Streamable HTTP, so the same code can be upgraded for production-shaped demos later.
- **stdio transport for the lab:** Eliminates OAuth flows, Origin validation, and HTTP server setup. Streamable HTTP is the production transport for remote servers, but it adds 20+ minutes of cognitive load that would crowd out the core concepts in a 60-90 minute session.
- **MCP Inspector for debugging:** Provides interactive protocol-level testing (discovery, tool calls, schema inspection) without hiding details behind a chat UI. This is the tool SEs will actually use when troubleshooting customer POVs.

Key trade-offs: realism vs time-box

Lab simplification	Production reality	Why acceptable
stdio transport	Streamable HTTP with auth	Teaches core MCP concepts without OAuth overhead
Deterministic mock data	Live search index	Makes validation reproducible, debugging faster
Function URL auth: NONE	IAM or OAuth-protected endpoint	Removes auth complexity from the critical path
Security as teachable overlay	Full implementation required	SEs learn the vocabulary and requirements without the setup time
Single tool	Multi-tool, filtered discovery	One tool teaches the full pattern; more tools are additive, not conceptually new

Simplification philosophy

Keep the concept set to "tool discovery + one tool call." This matches how official MCP tutorials introduce the protocol, and it's enough for SEs to build a credible mental model. Everything else (multi-tool, resources, prompts, OAuth) is additive, not conceptually new.

PART 2 — REASONING (CONTINUED)

Business impact and enablement fit

Business and GTM relevance

- **The customer story this lab enables:** "Your agents need controlled access to real enterprise systems. MCP provides a standardized connector surface—like a universal port for agent integrations—reducing bespoke integration work across every system your agents need to reach."
- **Enterprise search as the grounding layer:** RAG is the core pattern for reducing hallucinations in knowledge-intensive tasks. A search integration is the strongest default demo because it shows real retrieval, real grounding, and real citations—not just "AI magic."
- **Governance and consent positioning:** MCP emphasizes authorization for user-specific data and enterprise environments. SEs can speak credibly about permissions, auditing, and least privilege—the exact questions enterprise security teams will ask.
- **Discovery conversations:** After this lab, SEs can map a customer's "systems of record" (search, ticketing, CRM, knowledge base) to MCP server tool surfaces. Reference implementations exist across many categories, making the ecosystem immediately credible.
- **Demo-to-POV bridge:** The lab teaches SEs to define "meaningful results" criteria using repeatable test queries and validation dimensions (relevance, grounding, permissions). This is the bridge from "impressive demo" to "measurable customer value."

Enablement fit

- **Position in SE enablement path:** This is a **mid-tier, intermediate** lab. Prerequisites include basic cloud and API literacy (covered in onboarding), but it introduces a new protocol and integration pattern. It sits after "Platform Overview" and "Core Demo Skills" in the curriculum, before advanced topics like multi-agent orchestration or custom connector development.
- **Reusability:** The lab structure (mock API + MCP server + validation) is a reusable template. Swap the mock search for a mock ticketing system, CRM, or document store to create a library of integration labs from the same foundation.
- **Iteration ideas for v2 and beyond:**

v2: Streamable HTTP + Auth

Upgrade from stdio to Streamable HTTP with OAuth. Teaches the production transport and security model. Adds 30 minutes; makes it a half-day session.

v3: Multi-tool + Observability

Add a second tool (e.g., ticketing), introduce tool filtering, and add OpenTelemetry tracing for production-grade observability.

Stretch: Customer-facing workshop

Adapt the lab into a 2-hour customer workshop where technical champions build their own MCP server for their enterprise search system. SEs co-facilitate.

Stretch: Evaluation framework

Build a lightweight scoring harness that compares agent answers against a golden-answer set. Introduces evaluation rigor for POV-stage conversations.

PART 3 — QUICK REFERENCE

Cheat sheet: concepts and architecture

Keep this open during the lab or reference later.

Key MCP concepts

Concept	What it means
MCP Server	Exposes tools, resources, and prompts to clients via a standardized schema
MCP Client	Discovers server capabilities and calls tools with structured arguments
Tools	Callable functions for side-effecting actions or API calls (our lab focus)
Resources	Read-only data artifacts, analogous to "file-like" context
Prompts	Reusable prompt templates exposed by a server to the client
JSON-RPC	Message encoding format: request/response semantics over the transport
stdio transport	Client launches server as a subprocess. Simplest option for local work
Streamable HTTP	Server runs independently, supports multiple clients. Production transport
MCP Inspector	Interactive debugging tool for protocol-level testing of discovery + tool calls

Architecture at a glance

PART 3 — QUICK REFERENCE (CONTINUED)

Cheat sheet: commands and troubleshooting

Critical rules

stdout is sacred (stdio)

Never write anything to stdout except JSON-RPC frames. All logging goes to `stderr`. Use: `print( ... , file=sys.stderr)`

SSE is deprecated

Server-Sent Events as a standalone MCP transport is deprecated. Use **stdio** (local) or **Streamable HTTP** (remote) for all new work.

Core commands

```
# Test the mock API directly
curl "https://<function-url>/search?q=renewal+process&top_k=3"

# Run MCP Inspector against your server
npx @anthropic-ai/mcp-inspector python server.py

# Run the MCP server locally (stdio)
python server.py

# Check Lambda logs
aws logs tail /aws/lambda/mock-search --follow
```

Troubleshooting quick reference

Symptom	Likely cause	Check this
No tools show up in Inspector	Server not running or not discoverable	Confirm server process is alive; check stderr for startup errors
Server "breaks randomly"	stdout corruption from print()	Search for `print(` without `file=sys.stderr`
Tool call returns empty/error	Wrong endpoint URL	Check ENDPOINT variable; try curl directly
Mock API returns 403 or 500	Lambda misconfiguration	CloudWatch log group for the Lambda function
Agent ignores search results	Response format mismatch	Confirm tool returns valid JSON string; check Inspector output tab

PART 3 — QUICK REFERENCE (CONTINUED)

Cheat sheet: customer-facing talk tracks

Reusable phrases SEs can adapt for demos and POV conversations.

"MCP is a standard interface for tool discovery and structured tool calls—like a universal port for agent integrations. Instead of building a custom connector for every system, you declare your tools once and any MCP-compatible agent can discover and use them."

"We keep enterprise permissions intact. The agent only sees what the user is allowed to see. Every tool call goes through a defined trust boundary with structured schemas—not open-ended access to your systems."

"We ground responses in retrieved sources, then generate. That means every answer comes with citations you can verify. It's the difference between 'the AI said so' and 'here's the source document.'"

"The protocol handles the plumbing—tool discovery, schema validation, transport security. Your team focuses on which enterprise systems to connect and what governance policies to apply."

"This isn't a black box. MCP Inspector gives you protocol-level visibility into every tool call, every schema, every response. You can debug an integration in minutes, not days."

FUTURE ITERATIONS

Stretch ideas and evolution roadmap

How this lab grows from a single session into a sustainable enablement program.

Iteration roadmap

✓ **v1: Current lab** TODAY

Single tool, stdio transport, mock search API, MCP Inspector debugging. Teaches the core mental model and gives SEs demo confidence in 60-90 minutes.

2 **v2: Production transport + auth** NEXT QUARTER

Upgrade to Streamable HTTP transport with OAuth 2.1 authorization. Adds Origin header validation. Half-day session. SEs can now speak credibly to security teams.

3 **v3: Multi-tool + observability** FUTURE

Add a second enterprise tool (ticketing or CRM), introduce tool filtering and routing, add OpenTelemetry tracing. SEs can demo complex multi-system agent workflows.

4 **v4: Customer-facing workshop** STRETCH

Adapt into a 2-hour workshop where customer technical champions build their own MCP server connected to their enterprise search. SEs co-facilitate. Deepens post-sales relationships.

Additional stretch ideas

Evaluation framework

Build a lightweight scoring harness that compares agent answers against a golden-answer set. Uses basic IR metrics (precision@k). Bridges "impressive demo" to "measurable value" for POV-stage conversations.

Prompt injection red-teaming

Add a module where SEs attempt prompt injection against the MCP tool boundary. Demonstrates why tool scoping, input validation, and human-in-the-loop controls matter. Great for security-conscious enterprise customers.

Self-service lab kit

Package the lab as a self-paced GitHub repo with a Terraform/CDK template for the AWS infrastructure. New SEs can run through it independently during onboarding. Reduces facilitator dependency.

Competitive positioning module

Add a 15-minute segment comparing MCP-based integration to proprietary alternatives (custom SDKs, GraphQL gateways, vendor-locked connectors). Gives SEs ammunition for competitive deals.

Reusable template

The lab structure—mock API + MCP server + validation + troubleshooting + positioning—is a reusable template. Swap the mock search for any enterprise system (ticketing, CRM, document store, HRIS) to create a library of integration labs from the same foundation. One architecture, many customer stories.

SUMMARY

One protocol. Many enterprise systems. Grounded answers.

This lab gives SEs the hands-on confidence to demo, position, and troubleshoot MCP integrations with real customer impact.

- ✓ 60-90 minutes, hands-on, instructor-led
- ✓ AWS Lambda + MCP Python SDK + MCP Inspector
- ✓ Mock enterprise search with relevance, grounding, and permissions validation
- ✓ Troubleshooting framework: client, server, endpoint triage
- ✓ Customer-ready talk tracks and positioning narratives
- ✓ Reusable template for a library of integration labs

Prepared by John Yzaguirre
Technical Enablement Manager Assignment