Model Context Protocol (MCP): Connecting AI Models to Real Systems

Udaiappa Ramachandran (Udai)
CTO|CSO @ Akumina Inc., | Microsoft MVP (AI)
Web: https://udai.io
LinkedIn: https://linkedin.com/in/udair
Meetup: https://meetup.com/nashuaug
https://meetup.com/devboston

Agenda

- Why MCP exists and the problem it solves
- MCP architecture: Client, Server, and Tools
- MCP transports: stdio vs SSE
- MCP vs tools, function calling, and plugins
- Demo: MCP in action (agent execution & tools)
- Key takeaways and Q&A

Why MCP Exists

- LLMs need tools, data, and actions to be useful
- Ad-hoc plugins and direct function calling do not scale across models and systems
- Security, governance, and reuse are hard
- MCP defines a standard contract between models and systems

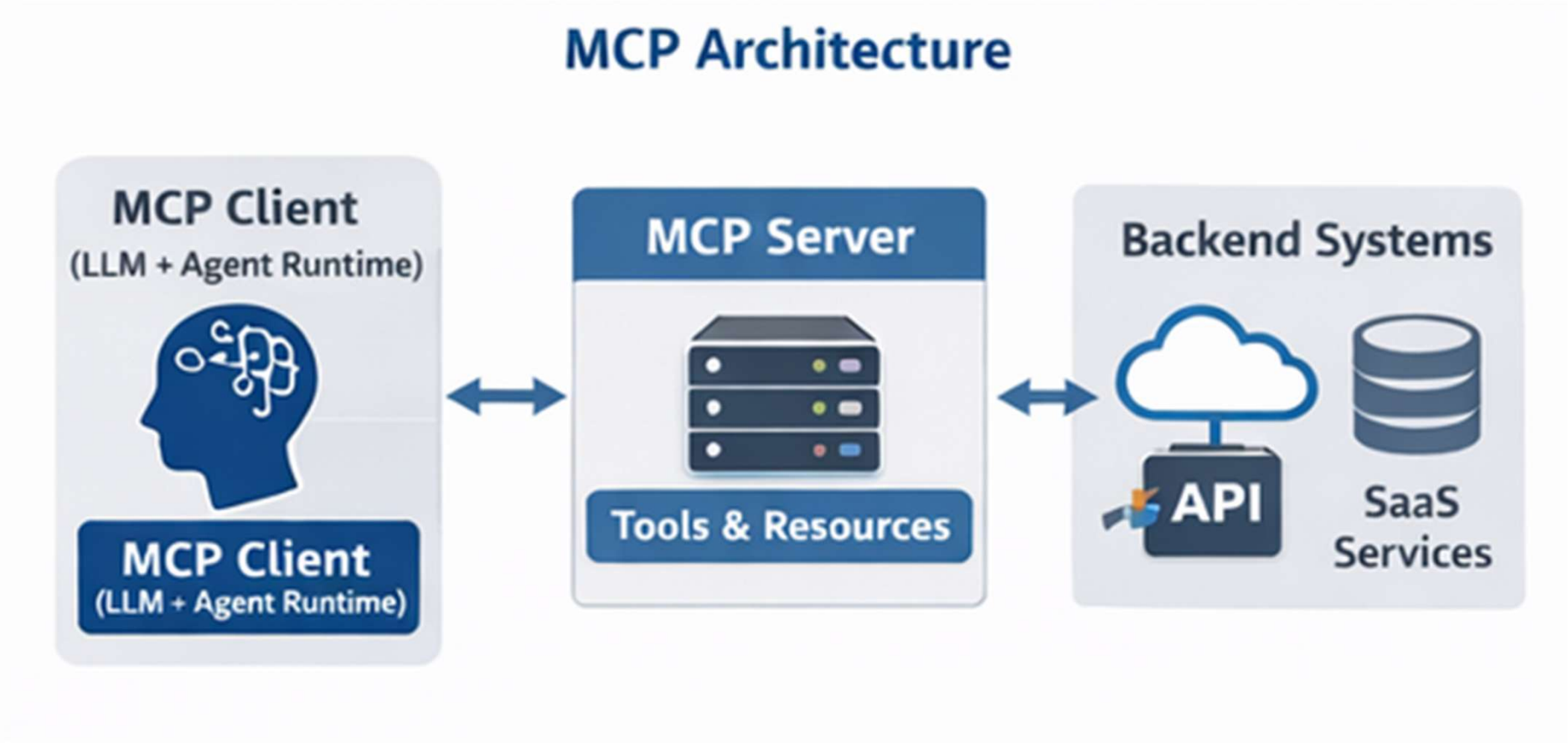
What Is MCP?

- Open protocol for exposing tools, resources, and prompts to models
- Transport-agnostic and model-agnostic
- Focuses on discoverability, invocation, and governance
- Think of MCP as 'HTTP for AI tools'

Core MCP Components

- MCP Client: speaks the protocol and hosts LLM / Agent Logic
- MCP Server: Exposes tools, resources, and prompts
- Backend Systems: REST APIs, databases, SaaS Platforms

MCP Architecture Overview



MCP Objects

- Tools: Callable actions with side effects
- Resources: read-only or queryable data
- Prompts: reusable instruction templates
- All defined via JSON schemas

MCP Transports

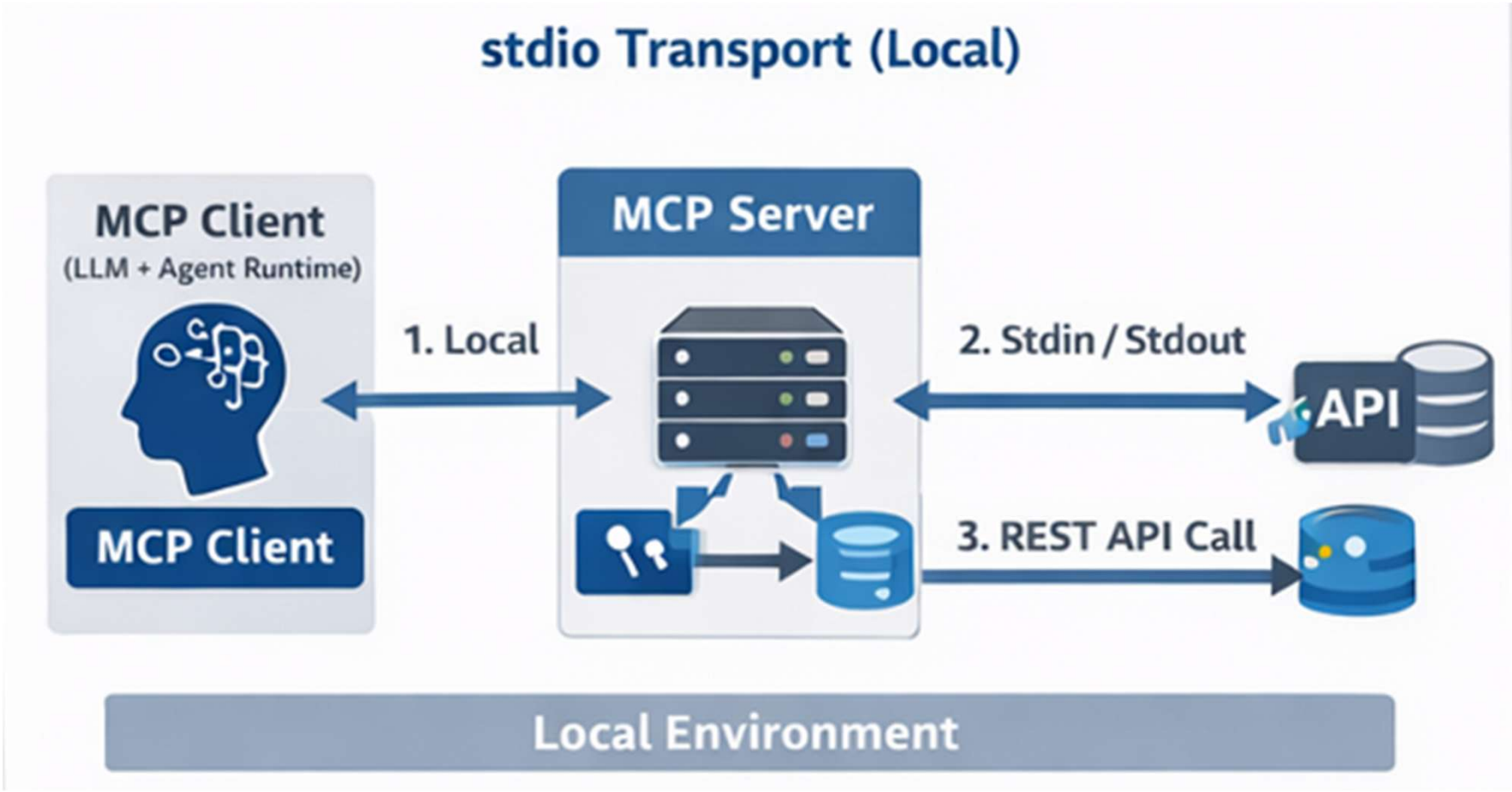
- MCP separates protocol from transport
- Stdio: local, process-based communication
- SSE: remote, HTTP-based communication
- Same MCP server logic, different deployment models
- Transports affect deployment and security, not MCP semantics

stdio vs SSE Comparison

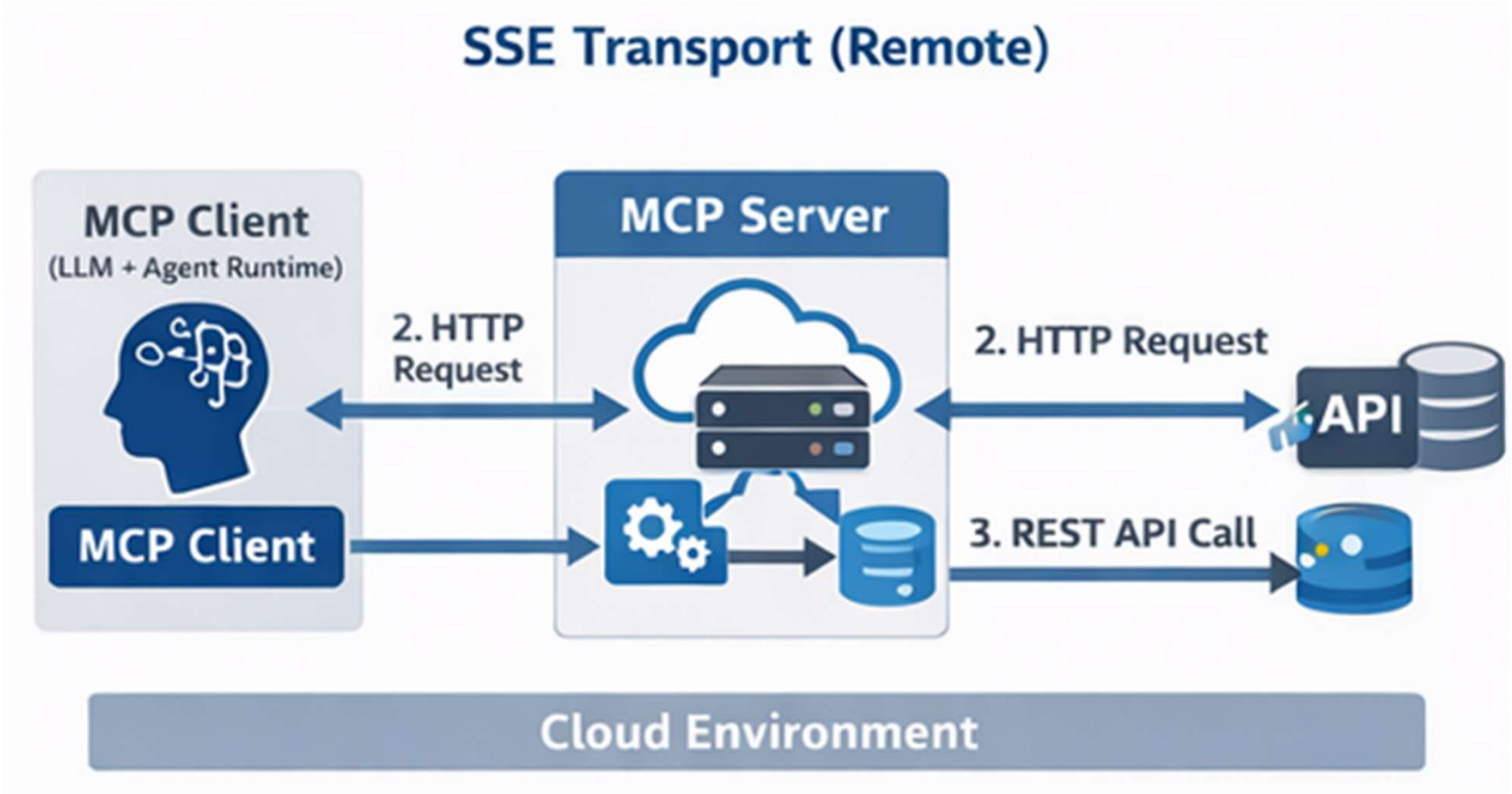
	stdio (Local)	HTTP/SSE (Remote)
Scope	Local	Remote
Deployment	Embedded	Service
Security	OS-level	OAuth / API Keys
Scalability	Single User	Multi-Tenant
Streaming	Limited	Full Support
Speed	Fast	Network latency
Cloud Ready	X	✅

Both transports expose the same MCP contract

Stdio Transport (Local)



SSE Transport (Remote)



MCP + REST APIs

- REST APIs already define inputs, outputs, and auth
- MCP acts as a semantic adapter for LLMs
- MCP server handles validation, auth, and orchestration
- MCP does not replace REST — it wraps it

Tools Invocation Lifecycle

- User asks a question
- LLM decides a tool is needed
- MCP Client invokes tool via MCP
- MCP Server validates and calls REST backend
- Result returned to LLM for final response

Security Model

- MCP Server is the trust boundary
- Secrets never exposed to the LLM
- Supports OAuth, API keys, managed identity
- Auditing, rate limiting, and authorization enforced centrally

Function Calling and Tools

- Function Calling (Model Capability)
  - Model-level feature to emit structured intent
  - Produces JSON matching a schema
  - No execution, security, or transport
  - Tied to a specific model runtime
  - Key point: Function calling answers “what does the model want to do?”
- Tools (Conceptual Abstraction)
  - A logical capability with name + input/output schema
  - Can be local, remote, synchronous, or async
  - Independent of how it’s invoked or transported
  - Key point: “Tool” is a design concept, not a protocol or product

Plugins and MCP

- Plugins (Legacy Tool Packaging)
  - Early vendor-specific implementation of tools
  - Combined:
    - Open API spec
    - Auth
    - UI integration
  - Designed for chat UI, not agent runtimes
  - Limited governance and reuse
  - Key point: Plugins were an early delivery mechanism for tools, not a new layer.
- MCP (Protocol + Runtime Contract)
  - Standardizes tool discovery, invocation, validation, and security
  - Transport-agnostic (stdio, SSE)
  - Decouples models from systems
  - Designed for agents, not UI widgets
  - Key point: MCP answers “how is this capability executed safely and consistently?”

MCP vs Tools Vs Function Calling Vs Plugins

	Function Calling	Tools	Plugins	MCP
Layer	Model	Design	Packaging	Protocol
Executes Code	X	X	✅	✅
Model-Specific	✅	X	✅	X
Transport-aware	X	X	**HTTP only**	**Stdio / SSE**
Agent-ready	⚠️	⚠️	X	✅
Governance	X	X	**Limited**	**Strong**
Reusable across models	X	✅	X	X

MCP can use function calling internally, but is not dependent on any specific model's implementation.

MCP in Agentic Systems

- Agents combine reasoning, memory, and tools
- MCP provides deterministic and governed tool access
- Works with single-agent and multi-agent systems
- Critical for production reliability

Demo: MCP Client + Tools in Practice

- Create Agent in Microsoft Foundry
- Run single agent execution
- Add MCP Tools
- Configuration Review

Key Takeaways

- MCP standardizes how models use tools, decoupling AI models from backend systems
- Provides a governed execution layer with security, validation, and reuse
- Supports multiple transports:
  - stdio for local tools and developer workflows
  - SSE for shared, cloud, and enterprise tools
- Enables production-grade AI systems that scale across models and agents
- Real-world systems often use both transports
- What MCP Is Not
  - Not a model or LLM
  - Not an agent framework
  - Not a replacement for REST
  - Not tied to any vendor

Reference

- https://modelcontextprotocol.io/docs/getting-started/intro
- https://github.com/modelcontextprotocol

Thank you for your time and trust!

Boston Azure AI – Dev Day