Agents, MCP, Tool Discovery, Authentication and Authorization

Student reading guide. This document addresses a central architectural question: if Google ADK agents already support tools and external APIs, why use Model Context Protocol (MCP)? It also explains what discovery actually means, what must be configured first, and why tools on one MCP server may require different credentials.

1. Agent versus MCP: two different jobs

AI agent: software that uses an AI model, instructions, and optionally tools to pursue a task. Google ADK is a framework for implementing and orchestrating such agents. MCP: a protocol for a client application to obtain descriptions of tools, resources and prompts from an MCP server and interact with them through standardized messages. MCP is not itself an agent or reasoning engine.

Question	Google ADK agent	MCP
Who decides to use a tool?	Agent/model and application logic	MCP does not decide
How are tools supplied?	Local functions, adapters, remote APIs, MCP tools	Standardized tools/list and tools/call
Is MCP required?	No	Only when using MCP integration
Does it provide API keys?	No	No

2. Three ways to provide the same nutrition tool

Option A: direct Python tool. The developer implements lookup_food() and registers it in ADK. Option B: REST API tool. A registered Python wrapper calls an authenticated REST endpoint. Option C: MCP tool. ADK connects to a configured MCP server and receives tool metadata through the MCP protocol.

```
User -> ADK agent -> local Python function -> nutrition database
User -> ADK agent -> REST wrapper -> nutrition REST API
User -> ADK agent -> MCP client -> MCP server -> nutrition API
```

For a single ADK project with a few functions, A or B is often simpler. MCP becomes attractive when multiple compatible applications reuse independently maintained tool integrations.

3. What "tool discovery" really means

Discovery does not mean searching the internet for MCP servers, signing up for services, acquiring API keys, or receiving permission automatically. The developer or host must first select and configure an MCP server and arrange any required connection authentication. The connected MCP client can then ask which tools the server advertises.

```
1. Select MCP server (manual/configured)
2. Configure transport, endpoint and authentication
3. Connect; complete protocol initialization
4. Request tools/list -> names, descriptions, input schemas
5. Agent/application chooses permitted tool
6. Invoke tools/call -> server checks authorization
```

Some clients refresh the list when the server signals a change; others cache or filter it. A newly listed tool is not guaranteed to become available to the model immediately.

4. Example: what does tools/list reveal?

```
{"jsonrpc":"2.0","id":1,"method":"tools/list"}

Tool description (simplified):
{"name":"lookup_food",
 "description":"Look up food nutrients",
 "inputSchema":{"type":"object",
   "properties":{"food":{"type":"string"}},
   "required":["food"]}}
```

The description tells the client how to formulate a tool call. It does not contain a secret API key, certify that the current user has access, or guarantee that the downstream nutrition service is working.

5. Authentication versus authorization

Authentication: establishing who the caller is (for example, an OAuth access token presented to an HTTP MCP server). Authorization: deciding what that caller is allowed to do (for example, permitting lookup_food but denying delete_food). A connection can be authenticated while a particular tool call is denied.

Layer	Example	Responsibility
Agent to MCP server	OAuth token / other configured access mechanism	MCP server validates caller and enforces permissions
MCP server to nutrition API	Vendor API key / service identity	Server secures downstream credentials
MCP server to calendar API	User-delegated OAuth	User must consent; server enforces user scope

6. Can one MCP server expose tools with different keys?

Yes. One MCP server might provide lookup_food using its own nutrition API key, read_calendar using an individual user's OAuth authorization, and search_products using another vendor credential. Those are separate downstream authentication arrangements. The agent usually does not need to see the underlying secrets.

```
ADK agent --[MCP connection auth]--> MCP server
                                     |-- lookup_food -> API key A
                                     |-- read_calendar -> user OAuth B
                                     `-- search_products -> API key C
```

The MCP server may advertise only authorized tools, advertise a tool that later returns an authorization-required error, or support a user-connection flow. This behavior depends on the server and host implementation. Tool discovery and successful execution are separate.

7. What happens when credentials are missing?

Example: the agent sees read_calendar in tools/list but the user has not connected a calendar account. The agent requests tools/call; the MCP server checks authorization and refuses or signals the need for authorization. The host application must arrange a supported consent or account-linking flow. The agent cannot generate an OAuth grant or invent a valid key. After the user authorizes access, the tool can be retried according to application policy.

For remote HTTP MCP, protocol-level authorization can use OAuth mechanisms. Downstream vendor authorization is not automatically standardized into one universal credential flow. For local stdio MCP servers, credential handling often relies on the launching host and environment.

8. Does MCP eliminate duplicated integration code?

No. Every application still needs MCP client support or an adapter, server configuration, credentials/consent, error handling and tool approval policy. What can be reused is the tool implementation and its discoverable schema. A conventional REST API plus OpenAPI and reusable clients can also centralize integrations; MCP is an AI-tool-oriented interoperability convention, not magic code elimination.

Work	Still needed per client?
Configure MCP server and client	Usually yes
Authenticate or obtain user consent	Yes, as required
Implement underlying vendor API calls	Usually centralized at MCP server
Describe each tool to the model	Can be discovered through MCP
Authorize each tool invocation	Must be enforced server-side
Handle tool failure and unavailable service	Yes

9. Security design: do not trust discovery alone

A discovered tool may perform a sensitive action. A secure deployment should authenticate callers, authorize each invocation, restrict scopes, validate tool arguments, avoid exposing secrets to the model, protect against prompt injection from tool outputs, log sensitive actions carefully, and require confirmation for high-impact operations. A broad server-level credential must not automatically authorize every user to access every downstream account.

10. When should a CS651 project use MCP?

Project situation	Suggested choice
One ADK agent with two local tools	Direct ADK functions
Several agents in one Python codebase	Shared Python tools / modules
One agent calling a stable REST service	Direct REST adapter often sufficient
Many agent frameworks using shared capabilities	Consider MCP
External provider already exposes an MCP server	MCP may reduce custom adapter work
User-specific accounts with complex consent	Evaluate security and operational cost carefully

11. Discussion exercise: design a nutrition MCP server

Assume one MCP server exposes (1) lookup_food using a server-managed API key, (2) read_calendar requiring each user's Google OAuth consent, and (3) delete_meal_log requiring elevated authorization. Explain: (a) which tools may appear in tools/list before consent; (b) which credentials live in the MCP server; (c) what happens if read_calendar is called without consent; (d) why authenticating the ADK application does not authorize delete_meal_log; and (e) whether REST + OpenAPI would be simpler for one small application.

12. Key takeaways

Agent = decision-making system. MCP = standardized tool/context interface. Discovery means listing capabilities on an already configured connection, not automatically finding servers or acquiring keys. One server can front many external services with different authentication requirements. Discoverability is not permission. MCP offers interoperability and reuse, but does not remove security, registration, configuration or integration work.

Further reading

Model Context Protocol specification: https://modelcontextprotocol.io/specification ; MCP authorization: https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization ; Google ADK documentation: https://google.github.io/adk-docs/ . Check current versions and provider-specific requirements when implementing.