# Integrate MCP Tools with Azure AI Agents

## 1. Introduction

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/1-introduction

# Introduction

Completed

- 2 minutes

AI agents are capable of performing a wide range of tasks, but many tasks still require them to interact with tools outside the large language model. Agents may need to access APIs, databases, or internal services. Manually integrating and maintaining these tools can quickly become complex, especially as your system grows, or changes frequently.

Model Context Protocol (MCP) servers can help solve this problem by integrating with AI agents. Connecting an Azure AI Agent to a Model Context Protocol (MCP) server can provide your agent with a catalog of tools accessible on demand. This approach makes your AI solution more robust, scalable, and easier to maintain.

Suppose you're working for a retailer that specializes in cosmetics. Your team wants to build an AI assistant that can help manage inventory by checking product stock levels and recent sales trends. Using an MCP server, you can connect the assistant to a set of tools that can make inventory assessments and provide recommendations to the team.

In this module, you learn how to set up an MCP server and client, and connect tools to an Azure AI Agent dynamically. You also practice creating your own AI MCP tool solution with Microsoft Foundry Agent Service.

## 2. Understand MCP tool discovery

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/2-understand-mcp-tool-discovery

# Understand MCP tool discovery

Completed

- 5 minutes

As AI agents become more capable, the range of tools and services they can access also grows. However, registering new tools, managing, updating, and integrating them can quickly become complex and time-consuming. Dynamic tool discovery helps solve this problem by enabling agents to find and use tools automatically at runtime.

## Advantages of the Microsoft Connector Protocol for AI agents

The Microsoft Connector Protocol (MCP) provides several benefits that enhance the capabilities and flexibility of AI agents:

- **Dynamic Tool Discovery:**
  AI agents can automatically receive a list of available tools from a server, along with descriptions of their functions. Unlike traditional APIs, which often require manual coding for each integration and updates whenever the API changes, MCP enables an **"integrate once"** approach that improves adaptability and reduces maintenance.
- **Interoperability Across LLMs:**
  MCP works seamlessly with different large language models (LLMs), allowing developers to **switch or evaluate core models** for improved performance without reworking integrations.
- **Standardized Security:**
  MCP provides a **consistent authentication method**, simplifying secure access across multiple MCP servers. This eliminates the need to manage separate keys or authentication protocols for each API, making it easier to scale AI agent deployments.

## What is dynamic tool discovery?

Dynamic tool discovery is a mechanism that allows an AI agent to discover available external tools without needing hardcoded knowledge of each one. Instead of manually adding or updating every tool your agent can use, the agent queries a centralized Model Context Protocol (MCP) server. This server acts as a live catalog, exposing tools that the agent can understand and call.

This approach means:

- Tools can be added, updated, or removed centrally without modifying the agent code.
- Agents can always use the latest version of a tool, improving accuracy and reliability.
- The complexity of managing tools shifts away from the agent and into a dedicated service.

## How does MCP enable dynamic tool discovery?

An MCP server hosts a set of functions that are exposed as tools using the `@mcp.tool` decorator. Tools are a primitive type in the MCP that enables servers to expose executable functionality to clients. A client can connect to the server and fetch these tools dynamically. The client then generates function wrappers that are added to the Azure AI Agent's tool definitions. This setup creates a flexible pipeline:

- The MCP server hosts available tools.
- The MCP client dynamically discovers the tools.
- The Azure AI Agent uses the available tools to respond to user requests.

## Why use dynamic tool discovery with MCP?

This approach provides several benefits:

- Scalability: Easily add new tools or update existing ones without redeploying agents.
- Modularity: Agents can remain simple, focusing on delegation rather than managing tool details.
- Maintainability: Centralized tool management reduces duplication and errors.
- Flexibility: Supports diverse tool types and complex workflows by aggregating capabilities.

Dynamic tool discovery is especially useful in environments where tools evolve rapidly or where many teams manage different APIs and services. Using tools allows AI agents to adapt to changing capabilities in real time, interact with external systems securely, and perform actions that go beyond language generation.

## 3. Integrate agent tools using an MCP server and client

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/3-mcp-client-server-setup

# Integrate agent tools using an MCP server and client

Completed

- 5 minutes

To dynamically connect tools to your Azure AI Agent, you first need a functioning Model Context Protocol (MCP) setup. This includes both the MCP server, which hosts your tool catalog, and the MCP client, which fetches those tools and makes them usable by your agent.

## What is the MCP Server?

The MCP server acts as a registry for tools your agent can use. You can initialize your MCP server using `FastMCP("server-name")`. The FastMCP class uses Python type hints and document strings to automatically generate tool definitions, making it easy to create and maintain MCP tools. These definitions are then served over HTTP when requested by the client. Because tool definitions live on the server, you can update or add new tools at any time, without having to modify or redeploy your agent.

## What is the MCP Client?

A standard MCP client acts as a bridge between your MCP server and the Azure AI Agent Service. The client initializes an MCP client session and connects to the server. Afterwards, it performs three key tasks:

- Discovers available tools from the MCP server using `session.list_tools()`.
- Generates Python function stubs that wrap the tools.
- Registers those functions with your agent.

This allows the agent to call any tool listed in the MCP catalog as if it were a native function, all without hardcoded logic.

## Register tools with an Azure AI Agent

When an MCP client session is initialized, the client can dynamically pull in tools from the MCP server. An MCP tool can be invoked using `session.call_tool(tool_name, tool_args)`. The tools should each

be wrapped in an async function so that the agent is able to invoke them. Finally, those functions are bundled together and become part of the agent's toolset and are available during runtime for any user request.

**Overview of MCP agent tool integration**

- The **MCP server** hosts tool definitions decorated with `@mcp.tool`.
- The **MCP client** initializes an MCP client connection to the server.
- The **MCP client** fetches the available tool definitions with `session.list_tools()`.
- Each tool is wrapped in an async function that invokes `session.call_tool`
- The tool functions are bundled into `FunctionTool` that makes them usable by the agent.
- The `FunctionTool` is registered to the agent's toolset.

Now your agent is able to access and invoke your tools through natural language interaction. By setting up the MCP server and client, you create a clean separation between tool management and agent logic—enabling your system to adapt quickly as new tools become available.

### 4. Use Azure AI agents with MCP servers

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/4-use-azure-ai-agents-with-mcp

# Use Azure AI agents with MCP servers

Completed

- 5 minutes

You can enhance your Microsoft Foundry agent by connecting it to Model Context Protocol (MCP) servers. MCP servers provide tools and contextual data that your agent can use to perform tasks, extending its capabilities beyond built-in functions. Azure AI Agent Service includes support for remote MCP servers, allowing your agent to quickly connect to your server and access tools.

When you use the Microsoft Foundry Agent Service to connect to your MCP server, you don't need to manually create an MCP client session or add any function tools to your agent. Instead, you create an MCP tool object that connects to your MCP server. Then you add information about the MCP server to the agent thread when invoking a prompt. This also allows you to connect and use different tools from multiple servers depending on your needs.

## Integrating remote MCP servers

To connect to an MCP server, you need:

- A remote MCP server endpoint (for example, https://api.githubcopilot.com/mcp/).
- A Microsoft Foundry agent configured to use the MCP tool.

You can connect to multiple MCP servers by adding them as separate tools, each with:

- `server_label` : A unique identifier for the MCP server (e.g., GitHub).
- `server_url` : The MCP server's URL.
- `allowed_tools` (optional): A list of specific tools the agent is allowed to access.

The MCP tool also supports custom headers, which let you pass:

- Authentication keys (API keys, OAuth tokens).
- Other required headers for the MCP server.
    - These headers are included in `tool_resources` during each run and are not stored between runs.

## Invoking tools

When using the Azure MCP Tool object, you don't need to wrap function tools or invoke `session.call_tool` . Instead, the tools are automatically invoked when necessary during an agent run. To automatically invoke MCP tools:

- Create the `McpTool` object with the server label and url.
- Use `update_headers` to apply any headers required by the server.
- Use the `set_approval_mode` to determine whether approval is required. Supported values are:
    - `always` : A developer needs to provide approval for every call. If you don't provide a value, this one is the default.
    - `never` : No approval is required.
- Create a `ToolSet` object and add the `McpTool` object
- Create an agent run and specify the `toolset` property
- When the run completes, you should see the results of any invoked tools in the response.

If the model tries to invoke a tool in your MCP server with approval required, you get a run status of requires_action. - In the `requires_action` field, you can get more details on which tool in the MCP server is called and any arguments to be passed. - Review the tool and arguments so that you can make an informed decision for approval. - Submit your approval to the agent with call_id by setting approve to true.

MCP integration is a key step toward creating richer, more context-aware AI agents. As the MCP ecosystem grows, you'll have even more opportunities to bring specialized tools into your workflows and deliver smarter, more dynamic solutions.

## 5. Exercise - Connect MCP tools to Azure AI Agents

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/5-exercise

# Exercise - Connect MCP tools to Azure AI Agents

Completed

- 30 minutes

If you have an Azure subscription, you can complete this exercise to develop a Model Context Protocol (MCP) client-server application that dynamically registers tools to an Azure AI Agent.

> **Note**
>
> If you don't have an Azure subscription, you can sign up for an account, which includes credits for the first 30 days.

Launch the exercise and follow the instructions.

Launch Exercise

## 6. Module assessment

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/6-knowledge-check

# Module assessment

Completed

- 3 minutes

### 7. Summary

https://learn.microsoft.com/en-us/training/modules/connect-agent-to-mcp-tools/7-summary

# Summary

Completed

- 1 minute

In this module, you learned how to integrate external tools with Microsoft Foundry Agent Service using the Model Context Protocol (MCP).

By connecting your agent to an MCP server, you can dynamically discover and register tools at runtime without hardcoding APIs or redeploying your agent. Using an MCP client, you generated function wrappers from discovered tools and connected them directly to your agent. This integration allows your agent to adapt to evolving toolsets, and create more flexible AI solutions that can grow alongside your applications.

> **Tip**
>
> To learn more about MCP, see the **Model Context Protocol User Guide** and **AI Agents MCP Integration**. To learn more about using Microsoft Foundry Agent Service with MCP, visit **Connect to Model Context Protocol servers**.