Connect AI agents to tools using Model Context Protocol (MCP)

In this exercise, you'll create an agent that can connect to an MCP server and automatically discover callable functions.

You'll build a simple inventory assessment agent for a cosmetics retailer. Using the MCP server, the agent will be able to retrieve information about the inventory and make restock or clearance suggestions.

> **Tip**: The code used in this exercise is based on the for Azure AI Foundry and MCP SDKs for Python. You can develop similar solutions using the SDKs for Microsoft .NET. Refer to Azure AI Foundry SDK client libraries and MCP C# SDK for details.

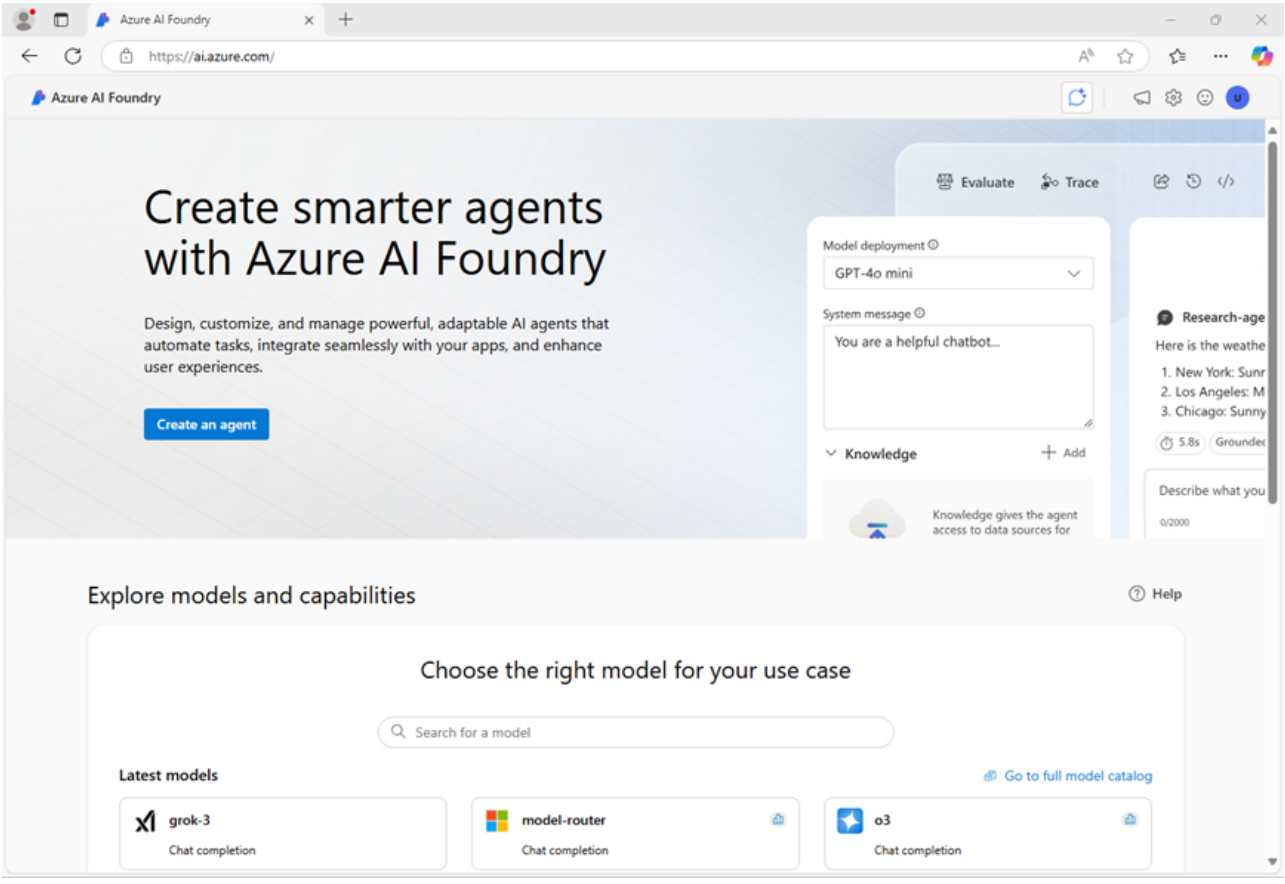
This exercise should take approximately **30** minutes to complete.

> **Note**: Some of the technologies used in this exercise are in preview or in active development. You may experience some unexpected behavior, warnings, or errors.

Create an Azure AI Foundry project

Let's start by creating an Azure AI Foundry project.

1. In a web browser, open the Azure AI Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Azure AI Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):



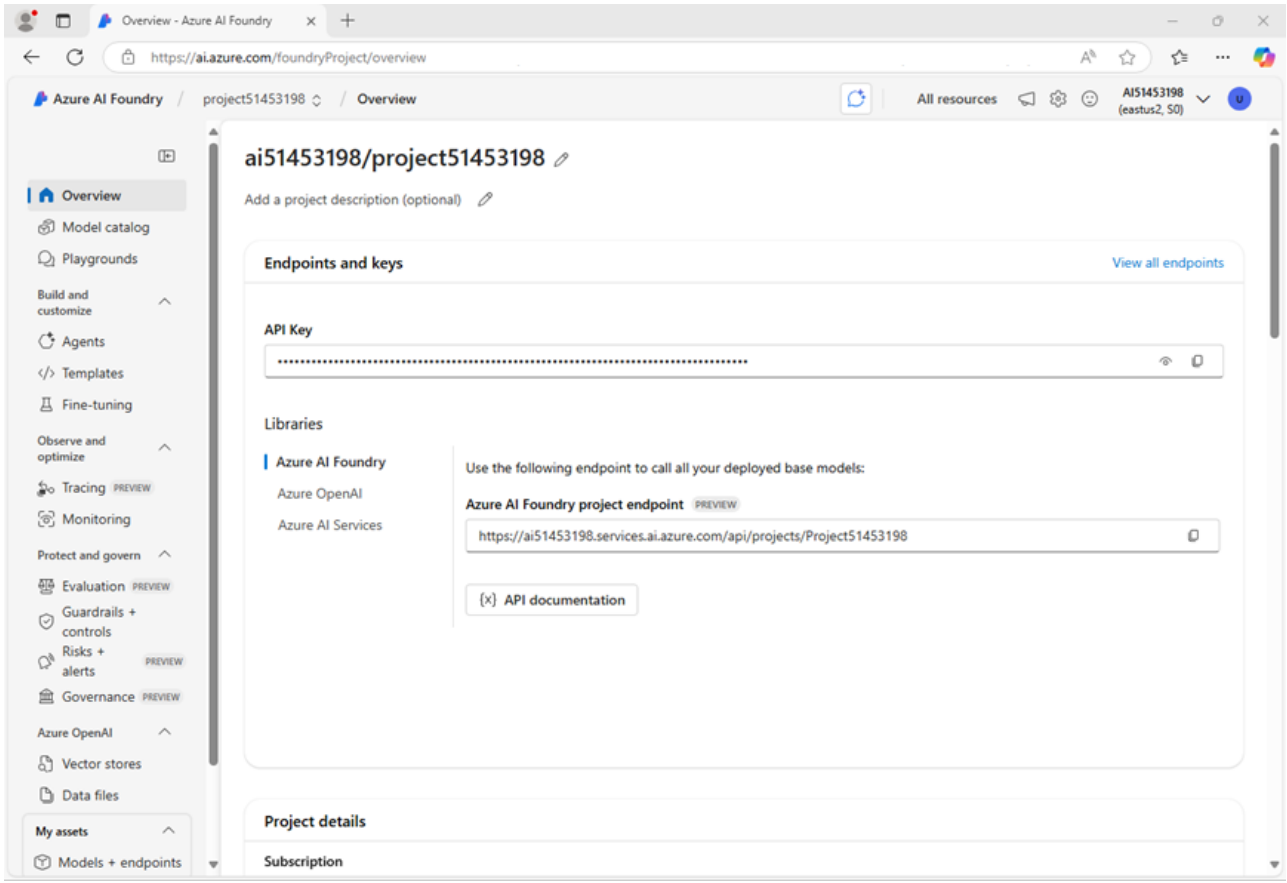
2. In the home page, select **Create an agent**.
3. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
4. Confirm the following settings for your project:
    - **Azure AI Foundry resource**: *A valid name for your Azure AI Foundry resource*
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Select any **AI Services supported location****

> * Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

5. Select **Create** and wait for your project to be created.
6. If prompted, deploy a **gpt-4o** model using either the *Global Standard* or *Standard* deployment option (depending on your quota availability).

> **Note**: If quota is available, a GPT-4o base model may be deployed automatically when creating your Agent and project.

7. When your project is created, the Agents playground will be opened.
8. In the navigation pane on the left, select **Overview** to see the main page for your project; which looks like this:



9. Copy the **Azure AI Foundry project endpoint** value to a notepad, as you'll use it to connect to your project in a client application.

Develop an agent that uses MCP function tools

Now that you've created your project in AI Foundry, let's develop an app that integrates an AI agent with an MCP server.

Clone the repo containing the application code

1. Open a new browser tab (keeping the Azure AI Foundry portal open in the existing tab). Then in the new tab, browse to the Azure portal at `https://portal.azure.com`; signing in with your Azure credentials if prompted.

   Close any welcome notifications to see the Azure portal home page.

2. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment with no storage in your subscription.

   The cloud shell provides a command-line interface in a pane at the bottom of the Azure portal. You can resize or maximize this pane to make it easier to work in.

   > **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

3. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

4. In the cloud shell pane, enter the following commands to clone the GitHub repo containing the code files for this exercise (type the command, or copy it to the clipboard and then right-click in the command line and paste as plain text):

   ```
   rm -r ai-agents -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-agents ai-agents
   ```

   > **Tip**: As you enter commands into the cloudshell, the output may take up a large amount of the screen buffer and the cursor on the current line may be obscured. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

5. Enter the following command to change the working directory to the folder containing the code files and list them all.

   ```
   cd ai-agents/Labfiles/07-use-agent-tools-with-mcp/Python
   ls -a -l
   ```

The provided files include the client and server application code. The Model Context Protocal provides a standardized way to connect AI models to different data sources and tools. We separate `client.py` and `server.py` to keep the agent logic and tool definitions modular and simulate real-world architecture.

`server.py` defines the tools the agent can use, simulating backend services or business logic. `client.py` handles the AI agent setup, user prompts, and calling the tools when needed.

Configure the application settings

1. In the cloud shell command-line pane, enter the following command to install the libraries you'll use:

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-ai-projects mcp
```

> **Note:** You can ignore any warning or error messages displayed during the library installation.

2. Enter the following command to edit the configuration file that has been provided:

```
code .env
```

The file is opened in a code editor.

3. In the code file, replace the **your_project_endpoint** placeholder with the endpoint for your project (copied from the project **Overview** page in the Azure AI Foundry portal) and ensure that the MODEL_DEPLOYMENT_NAME variable is set to your model deployment name (which should be *gpt-4o*).

4. After you've replaced the placeholder, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

Define function tools

1. Enter the following command to edit the code file that has been provided for your function code:

```
code server.py
```

In this code file, you'll define the tools the agent can use to simulate a backend service for the retail store.

2. Find the comment **Add an inventory check tool** and add the following code:

```
# Add an inventory check tool
@mcp.tool()
def get_inventory_levels() -> dict:
```

```
    """Returns current inventory for all products."""
    return {
        "Moisturizer": 6,
        "Shampoo": 8,
        "Body Spray": 28,
        "Hair Gel": 5,
        "Lip Balm": 12,
        "Skin Serum": 9,
        "Cleanser": 30,
        "Conditioner": 3,
        "Setting Powder": 17,
        "Dry Shampoo": 45
    }
```

This dictionary represents a sample inventory. The `@mcp.tool()` annotation will allow the LLM to discover your function.

3. Find the comment **Add a weekly sales tool** and add the following code:

```
# Add a weekly sales tool
@mcp.tool()
def get_weekly_sales() -> dict:
    """Returns number of units sold last week."""
    return {
        "Moisturizer": 22,
        "Shampoo": 18,
        "Body Spray": 3,
        "Hair Gel": 2,
        "Lip Balm": 14,
        "Skin Serum": 19,
        "Cleanser": 4,
        "Conditioner": 1,
        "Setting Powder": 13,
        "Dry Shampoo": 17
    }
```

4. Save the file (*CTRL+S*).

Connect the MCP tools to your agent

1. Enter the following command to begin editing the agent code.

```
code client.py
```

In this file, you'll prepare the AI agent, accept user prompts, and invoke the function tools.

> **Tip**: As you add code to the code file, be sure to maintain the correct indentation.

2. Review the existing function `connect_to_server`. This function starts the server and retrieves the tools available. The rest of the file includes comments where you'll add the necessary code to implement your inventory agent.

3. Find the comment **Add references** and add the following code to import the classes:

```
# Add references
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from azure.ai.agents import AgentsClient
from azure.ai.agents.models import FunctionTool, MessageRole, ListSortOrder
from azure.identity import DefaultAzureCredential
```

4. Find the comment **Connect to the agents client** and add the following code to connect to the Azure AI project using the current Azure credentials.

```
# Connect to the agents client
agents_client = AgentsClient(
    endpoint=project_endpoint,
    credential=DefaultAzureCredential(
        exclude_environment_credential=True,
        exclude_managed_identity_credential=True
    )
)
```

5. Under the comment **List tools available on the server**, add the following code:

```
# List tools available on the server
response = await session.list_tools()
tools = response.tools
```

6. Under the comment **Build a function for each tool** and add the following code:

```
# Build a function for each tool
def make_tool_func(tool_name):
    async def tool_func(**kwargs):
        result = await session.call_tool(tool_name, kwargs)
        return result

    tool_func.__name__ = tool_name
    return tool_func

functions_dict = {tool.name: make_tool_func(tool.name) for tool in tools}
mcp_function_tool = FunctionTool(functions=list(functions_dict.values()))
```

This code dynamically wraps tools available in the MCP server so that they can be called by the AI agent. Each tool is turned into an async function and then bundled into a `FunctionTool` for the agent to use.

7. Find the comment **Create the agent** and add the following code:

```
# Create the agent
agent = agents_client.create_agent(
    model=model_deployment,
    name="inventory-agent",
    instructions="""
    You are an inventory assistant. Here are some general guidelines:
    - Recommend restock if item inventory < 10  and weekly sales > 15
    - Recommend clearance if item inventory > 20 and weekly sales < 5
    """,
    tools=mcp_function_tool.definitions
)
```

8. Find the comment **Enable auto function calling** and add the following code:

```
# Enable auto function calling
agents_client.enable_auto_function_calls(tools=mcp_function_tool)
```

9. Under the comment **Create a thread for the chat session**, add the following code:

```
# Create a thread for the chat session
thread = agents_client.threads.create()
```

10. Locate the comment **Invoke the prompt** and add the following code:

```
# Invoke the prompt
message = agents_client.messages.create(
    thread_id=thread.id,
    role=MessageRole.USER,
    content=user_input,
)
run = agents_client.runs.create(thread_id=thread.id, agent_id=agent.id)
```

11. Locate the comment **Retrieve the matching function tool** and add the following code:

```
# Retrieve the matching function tool
function_name = tool_call.function.name
args_json = tool_call.function.arguments
kwargs = json.loads(args_json)
```

```
required_function = functions_dict.get(function_name)

# Invoke the function
output = await required_function(**kwargs)
```

This code uses the information from the agent thread's tool call. The function name and arguments are retrieved and used to invoke the matching function.

12. Under the comment **Append the output text**, add the following code:

```
# Append the output text
tool_outputs.append({
    "tool_call_id": tool_call.id,
    "output": output.content[0].text,
})
```

13. Under the comment **Submit the tool call output**, add the following code:

```
# Submit the tool call output
agents_client.runs.submit_tool_outputs(thread_id=thread.id, run_id=run.id,
tool_outputs=tool_outputs)
```

This code will signal to the agent thread that the required action is complete and update the tool call outputs.

14. Find the comment **Display the response** and add the following code:

```
# Display the response
messages = agents_client.messages.list(thread_id=thread.id,
order=ListSortOrder.ASCENDING)
for message in messages:
    if message.text_messages:
        last_msg = message.text_messages[-1]
        print(f"{message.role}:\n{last_msg.text.value}\n")
```

15. Save the code file (*CTRL+S*) when you have finished. You can also close the code editor (*CTRL+Q*); though you may want to keep it open in case you need to make any edits to the code you added. In either case, keep the cloud shell command-line pane open.

Sign into Azure and run the app

1. In the cloud shell command-line pane, enter the following command to sign into Azure.

```
az login
```

You must sign into Azure - even though the cloud shell session is already authenticated.

> **Note**: In most scenarios, just using *az login* will be sufficient. However, if you have subscriptions in multiple tenants, you may need to specify the tenant by using the *--tenant* parameter. See Sign into Azure interactively using the Azure CLI for details.

2. When prompted, follow the instructions to open the sign-in page in a new tab and enter the authentication code provided and your Azure credentials. Then complete the sign in process in the command line, selecting the subscription containing your Azure AI Foundry hub if prompted.

3. After you have signed in, enter the following command to run the application:

```
python client.py
```

4. When prompted, enter a query such as:

```
What are the current inventory levels?
```

> **Tip**: If the app fails because the rate limit is exceeded. Wait a few seconds and try again. If there is insufficient quota available in your subscription, the model may not be able to respond.

You should see some output similar to the folloiwng:

```
MessageRole.AGENT:
Here are the current inventory levels:

- Moisturizer: 6
- Shampoo: 8
- Body Spray: 28
- Hair Gel: 5
- Lip Balm: 12
- Skin Serum: 9
- Cleanser: 30
- Conditioner: 3
- Setting Powder: 17
- Dry Shampoo: 45
```

5. You can continue the conversation if you like. The thread is *stateful*, so it retains the conversation history - meaning that the agent has the full context for each response.

Try entering prompts such as:

```
Are there any products that should be restocked?
```

```
Which products would you recommend for clearance?
```

```
What are the best sellers this week?
```

Enter `quit` when you're done.

Clean up

Now that you've finished the exercise, you should delete the cloud resources you've created to avoid unnecessary resource usage.

1. Open the Azure portal at `https://portal.azure.com` and view the contents of the resource group where you deployed the hub resources used in this exercise.
2. On the toolbar, select **Delete resource group**.
3. Enter the resource group name and confirm that you want to delete it.