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

In this exercise, you'll build an agent that connects to a cloud-hosted MCP server. The agent will use AI-powered search to help developers find accurate, real-time answers from Microsoft's official documentation. This is useful for building assistants that support developers with up-to-date guidance on tools like Azure, .NET, and Microsoft 365. The agent will use the provided `microsoft_docs_search` tool to query the documentation and return relevant results.

> **Tip**: The code used in this exercise is based on the Azure AI Agent service MCP support sample repository. Refer to Azure OpenAI demos or visit Connect to Model Context Protocol servers for more details.

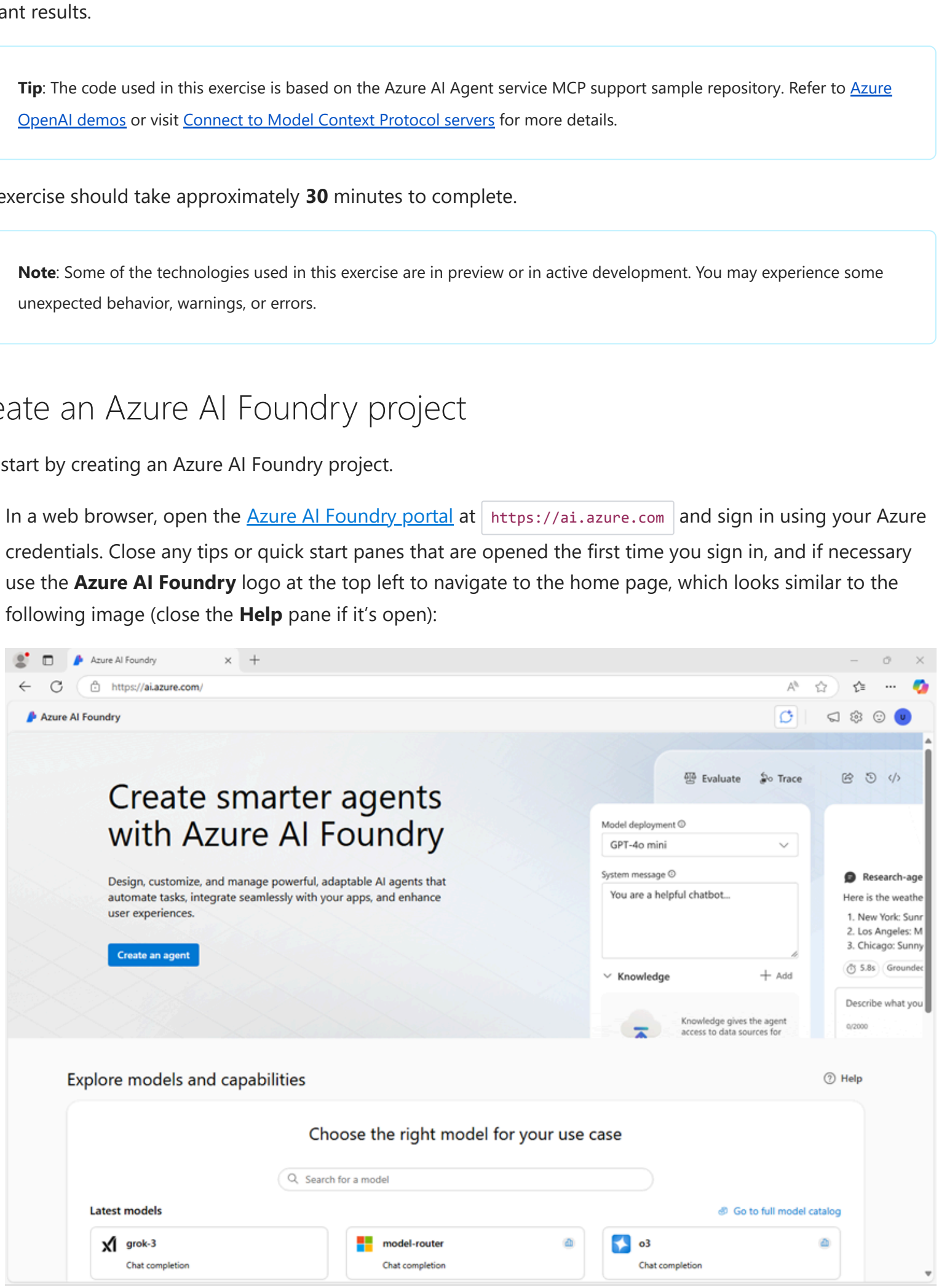
This exercise should take approximately **30** minutes to complete.

> **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 of the following supported locations: **

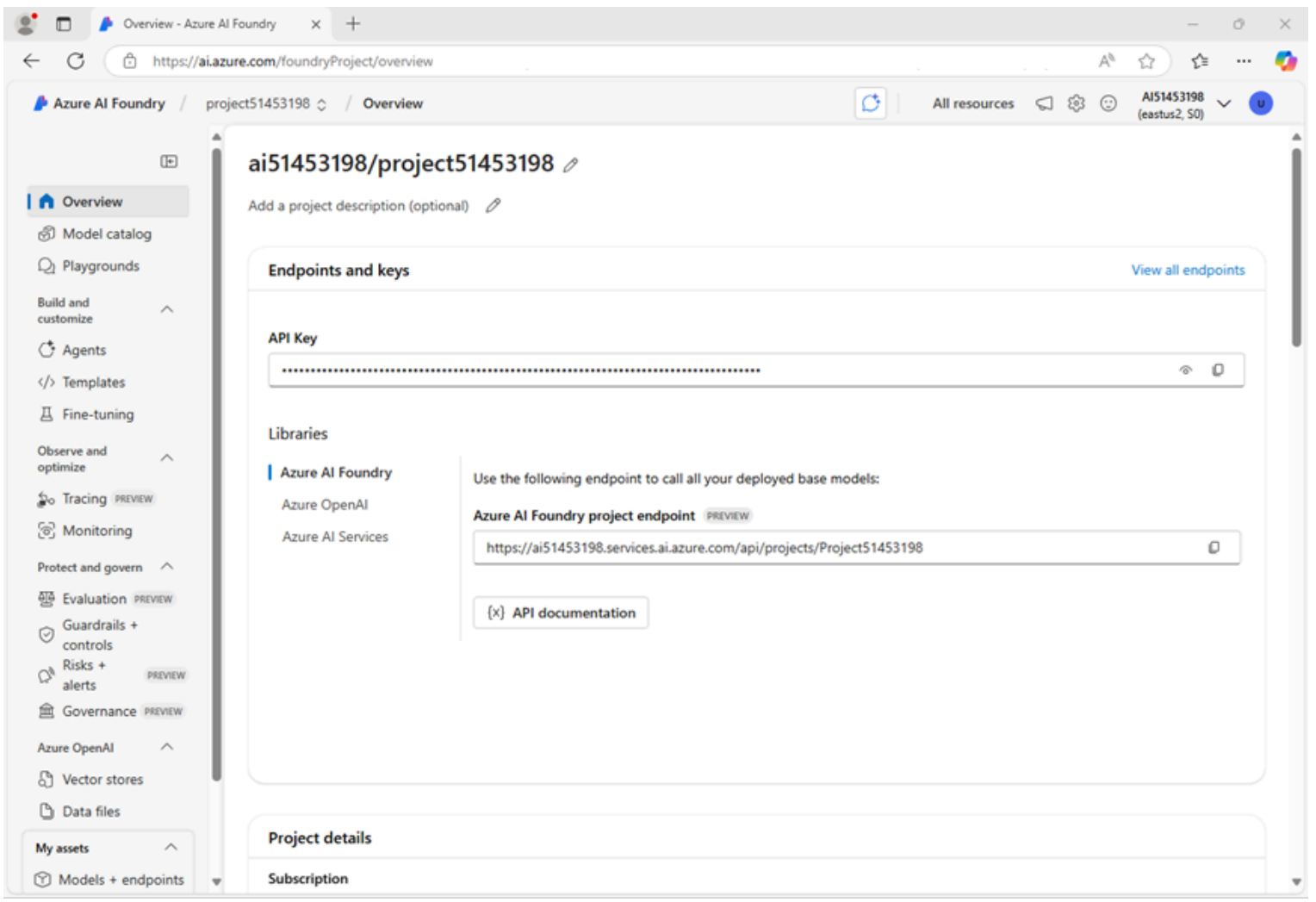
- West US 2
- West US
- Norway East
- Switzerland North
- UAE North
- South India

> * 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. 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):

   Code Copy

   ```
   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.

   Code Copy

   ```
   cd ai-agents/Labfiles/03c-use-agent-tools-with-mcp/Python
   ls -a -l
   ```

## Configure the application settings

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

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt --pre 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 Copy

   ```
   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.

## Connect an Azure AI Agent to a remote MCP server

In this task, you'll connect to a remote MCP server, prepare the AI agent, and run a user prompt.

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

```
code client.py
```

   The file is opened in the code editor.

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

```
# Add references
from azure.identity import DefaultAzureCredential
from azure.ai.agents import AgentsClient
from azure.ai.agents.models import McpTool, ToolSet, ListSortOrder
```

3. 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
     )
```

4. Under the comment **Initialize agent MCP tool**, add the following code:

```
# Initialize agent MCP tool
mcp_tool = McpTool(
     server_label=mcp_server_label,
     server_url=mcp_server_url,
)

mcp_tool.set_approval_mode("never")

toolset = ToolSet()
toolset.add(mcp_tool)
```

This code will connect to the Microsft Learn Docs remote MCP server. This is a cloud-hosted service that enables clients to access trusted and up-to-date information directly from Microsoft's official documentation.

5. Under the comment **Create a new agent** and add the following code:

Code Copy

```
# Create a new agent
agent = agents_client.create_agent(
     model=model_deployment,
     name="my-mcp-agent",
     instructions="""
     You have access to an MCP server called `microsoft.docs.mcp` - this tool allows you to
     search through Microsoft's latest official documentation. Use the available MCP tools
     to answer questions and perform tasks."""
)
```

In this code, you provide instructions for the agent and provide it with the MCO tool definitions.

6. Find the comment **Create thread for communication** and add the following code:

Code Copy

```
# Create thread for communication
thread = agents_client.threads.create()
print(f"Created thread, ID: {thread.id}")
```

7. Find the comment **Create a message on the thread** and add the following code:

Code Copy

```
# Create a message on the thread
prompt = input("\nHow can I help?: ")
message = agents_client.messages.create(
     thread_id=thread.id,
     role="user",
     content=prompt,
)
print(f"Created message, ID: {message.id}")
```

8. Find the comment **Set approval mode** and add the following code:

Code Copy

```
# Set approval mode
mcp_tool.set_approval_mode("never")
```

This allows the agent to automatically invoke the MCP tools without requiring user approval. If you want to require approval, you must supply a header value using `mcp_tool.update_headers`.

9. Find the comment **Create and process agent run in thread with MCP tools** and add the following code:

Code Copy

```
# Create and process agent run in thread with MCP tools
run = agents_client.runs.create_and_process(thread_id=thread.id, agent_id=agent.id,
toolset=toolset)
print(f"Created run, ID: {run.id}")
```

The AI Agent automatically invokes the connected MCP tools to process the prompt request. To illustrate this process, the code provided under the comment **Display run steps and tool calls** will output any invoked tools from the MCP server.

10. 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.

Code Copy

```
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:

Code Copy

```
python client.py
```

4. When prompted, enter a request for technical information such as:

Code Copy

```
Give me the Azure CLI commands to create an Azure Container App with a managed identity.
```

5. Wait for the agent to process your prompt, using the MCP server to find a suitable tool to retrieve the requested information. You should see some output similar to the following:

Code Copy

```
Created agent, ID: <<agent-id>>
MCP Server: mslearn at https://learn.microsoft.com/api/mcp
Created thread, ID: <<thread-id>>
Created message, ID: <<message-id>>
Created run, ID: <<run-id>>
Run completed with status: RunStatus.COMPLETED
Step <<step1-id>> status: completed

Step <<step2-id>> status: completed
MCP Tool calls:
    Tool Call ID: <<tool-call-id>>
    Type: mcp
    Type: microsoft_docs_search



Conversation:
--------------------------------------------------
ASSISTANT: You can use Azure CLI to create an Azure Container App with a managed identity
(either system-assigned or user-assigned). Below are the relevant commands and workflow:

 ---

 ### **1. Create a Resource Group**
 '''azurecli
 az group create --name myResourceGroup --location eastus
 '''




 By following these steps, you can deploy an Azure Container App with either system-assigned
or user-assigned managed identities to integrate seamlessly with other Azure services.
 --------------------------------------------------
 USER: Give me the Azure CLI commands to create an Azure Container App with a managed
identity.
 --------------------------------------------------
 Deleted agent
```

Notice that the agent was able to invoke the MCP tool `microsoft_docs_search` automatically to fulfill the request.

6. You can run the app again (using the command `python client.py`) to ask for different information, In each case, the agent will attempt to find technical documentation by using the MCP tool.

## 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.