Build AI agents with portal and VS Code

In this exercise, you'll build a complete AI agent solution using both the Microsoft Foundry portal and the Foundry Toolkit VS Code extension. You'll start by creating a basic agent in the portal with grounding data and built-in tools, then interact with it programmatically using VS Code to use advanced capabilities like code interpreter for data analysis.

This exercise takes approximately **45** minutes.

> **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.

Prerequisites

Before starting this exercise, ensure you have:

- An Azure subscription with sufficient permissions and quota to provision Azure AI resources
- Visual Studio Code installed on your local machine
- Python 3.13 or later installed
- Git installed on your local machine
- Basic familiarity with Azure AI services and Python programming

> * Python 3.13 is available, but some dependencies are not yet compiled for that release. The lab has been successfully tested with Python 3.13.12.

Create a Microsoft Foundry Project

Microsoft Foundry uses projects to organize models, resources, data, and other assets used to develop an AI solution.

1. In a web browser, open the 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 **Foundry** logo at the top left to navigate to the home page.

   > **Important**: For this lab, you're using the **New** Foundry experience.

2. In the top banner, select **Start building** to try the new Microsoft Foundry Experience.

3. When prompted, create a **new** project, and enter a valid name for your project (e.g., `it-support-agent-project`).
4. Expand **Advanced options** and specify the following settings:

   - **Microsoft Foundry resource**: *A valid name for your Foundry resource*
   - **Region**: *Select one available near you***
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *Select your resource group, or create a new one*

   > * 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. When your project is created, a welcome dialog may appear. Select **Next** to read through the welcome message, and then select **Create agent**.

   You can also select **Start building** on the home page, and select **Create agents** from the drop-down menu.
7. Set the **Agent name** to `it-support-agent` and create the agent.

The playground will open for your newly created agent. You'll see that an available deployed model is already selected for you.

Configure your agent with instructions and grounding data

Now that you have an agent created, let's configure it with instructions and add grounding data.

1. In the agent playground, set the **Instructions** to:

   Code Copy

   ```
   You are an IT Support Agent for Contoso Corporation.
   You help employees with technical issues and IT policy questions.

   Guidelines:
   - Always be professional and helpful
   - Use the IT policy documentation to answer questions accurately
   - If you don't know the answer, admit it and suggest contacting IT support directly
   - When creating tickets, collect all necessary information before proceeding
   ```

2. Download the IT policy document from the lab repository. Open a new browser tab and navigate to:

   Code Copy

   ```
   https://raw.githubusercontent.com/MicrosoftLearning/mslearn-ai-agents/main/Labfiles/01-build-agent-portal-and-vscode/IT_Policy.txt
   ```

   Save the file to your local machine.

   > **Note**: This document contains sample IT policies for password resets, software installation requests, and hardware troubleshooting.

3. Return to the agent playground. In the **Tools** section, select **Add**, and then add both **File search** and **</> Code interpreter**.
4. To the right of **Add**, select **Upload files**. Under **Attach files**, browse to and upload the `IT_Policy.txt` file you just downloaded, and then select **Attach**.
5. Wait for the file to be indexed. You'll see a confirmation when it's ready.
6. Now let's add some performance data for the code interpreter to analyze. Download the system performance data file from:

   Code Copy

   ```
   https://raw.githubusercontent.com/MicrosoftLearning/mslearn-ai-agents/main/Labfiles/01-build-agent-portal-and-vscode/system_performance.csv
   ```

   Save this file to your local machine.
7. To the right of **</> Code interpreter**, select **+ Files**, and then upload the `system_performance.csv` file you just downloaded.

> **Note**: This CSV file contains simulated system metrics (CPU, memory, disk usage) over time that the agent can analyze.

8. Save the agent.

Test your agent

Let's test the agent to see how it responds using the grounding data.

1. In the chat interface on the right side of the playground, enter the following prompt:

```
What's the policy for password resets?
```

2. Review the response. The agent should reference the IT policy document and provide accurate information about password reset procedures.

3. Try another prompt:

```
How do I request new software?
```

4. Again, review the response and observe how the agent uses the grounding data.

5. Now test the code interpreter with a data analysis request:

```
Can you analyze the system performance data and tell me if there are any concerning trends?
```

6. The agent should use the code interpreter to analyze the CSV file and provide insights about system performance.

7. Try asking for a visualization:

```
Create a chart showing CPU usage over time from the performance data
```

8. The agent will use code interpreter to generate visualizations and analysis.

Great! You've created an agent with grounding data, file search, and code interpreter capabilities. In the next section, you'll interact with this agent programmatically using VS Code.

Interact with your agent using VS Code

As a developer, you may spend some time working in the Foundry portal; but you're also likely to spend a lot of time in Visual Studio Code. The Foundry Toolkit for VS Code extension provides a convenient way to work with Foundry project resources without leaving the development environment.

Install and configure the VS Code extension

If you already have installed the Foundry Toolkit extension, you can skip this section.

1. Open Visual Studio Code.

2. Select **Extensions** from the left pane (or press **Ctrl+Shift+X**).

3. Search the extensions marketplace for the `Foundry Toolkit for VS Code` extension from Microsoft and select **Install**.

   Installing the Foundry Toolkit Extension will add the AI Toolkit extension to VS Code.

   > **Note**: The extension is currently listed as **Foundry Toolkit**, but some VS Code labels, commands, or older screenshots may still refer to **AI Toolkit**. In this lab, treat those names as referring to the same extension experience.

4. After installing the extension, select the AI Toolkit icon in the sidebar.

   You should be prompted to sign in to your Azure account if you haven't already.

Test your agent in VS Code

Before writing any code, you can interact with your agent directly in the extension interface.

1. Under **Microsoft Foundry Resources**, choose **Set Default Project**

   If a default project is already active, the project name will appear in the resources list. You can select a different project by selecting the same **Select project** icon.

2. Expand the project section. Under **Prompt Agents**, you should see the `it-support-agent` you created in the portal. Select the agent name to open the Agent Builder interface.

   The agent playground will appear in the Agent Builder interface, allowing you to interact with the agent and configure its settings without leaving VS Code.

3. In the playground chat pane, type a question such as:

   Code Copy

   ```
   What is the policy for reporting a lost or stolen device?
   ```

4. Review the agent's response. It should use the grounding data you uploaded earlier to provide relevant IT policy information.

   > **Tip**: You can use this built-in playground to quickly test your agent's instructions and knowledge without writing any code.

Create a client application to interact with your agent

Now let's create a client application that interacts with your agent programmatically.

1. In VS Code, open the Command Palette (**Ctrl+Shift+P** or **View > Command Palette**).
2. Type **Git: Clone** and select it from the list.
3. Enter the repository URL:

   Code Copy

   ```
   https://github.com/MicrosoftLearning/mslearn-ai-agents.git
   ```

4. Choose a location on your local machine to clone the repository.
5. When prompted, select **Open** to open the cloned repository in VS Code.
6. Once the repository opens, select **File > Open Folder** and navigate to `mslearn-ai-agents/Labfiles/01-build-agent-portal-and-vscode/Python`, then choose **Select Folder**.

7. In the Explorer pane, open the `agent_with_functions.py` file. If the file is empty, replace its contents with the following code.

8. Use the following code:

```python
import base64
import os
from pathlib import Path

from azure.ai.projects import AIProjectClient
from azure.identity import DefaultAzureCredential
from dotenv import load_dotenv


OUTPUT_DIR = Path("agent_outputs")


def get_output_path(filename):
    """Create a unique path for generated files."""
    OUTPUT_DIR.mkdir(exist_ok=True)
    file_name = Path(filename).name
    stem = Path(file_name).stem or "output"
    suffix = Path(file_name).suffix
    output_path = OUTPUT_DIR / file_name

    counter = 1
    while output_path.exists():
        output_path = OUTPUT_DIR / f"{stem}_{counter}{suffix}"
        counter += 1

    return output_path


def save_bytes(file_bytes, filename):
    """Save binary content to a local file."""
    output_path = get_output_path(filename)
    with open(output_path, "wb") as file_handle:
        file_handle.write(file_bytes)
    return output_path


def save_image(image_data, filename):
    """Save base64 image data to a file."""
    return save_bytes(base64.b64decode(image_data), filename)


def download_container_file(openai_client, annotation, downloaded_files):
    """Download a cited container file once and return its local path."""
    cache_key = (annotation.container_id, annotation.file_id)
    if cache_key in downloaded_files:
        return downloaded_files[cache_key]

    file_content = openai_client.containers.files.content.retrieve(
        file_id=annotation.file_id,
        container_id=annotation.container_id,
    )
    output_path = save_bytes(
        file_content.read(),
        annotation.filename or f"{annotation.file_id}.bin",
    )
    downloaded_files[cache_key] = output_path
    return output_path
```

```python
def format_output_text(content_item, openai_client, downloaded_files):
    """Replace sandbox file citations with local file paths."""
    text = content_item.text or ""
    replacements = []
    referenced_files = set()

    for annotation in content_item.annotations or []:
        if getattr(annotation, "type", "") != "container_file_citation":
            continue

        output_path = download_container_file(openai_client, annotation, downloaded_files)
        replacement_text = f"{annotation.filename} (saved to {output_path})"
        referenced_files.add(output_path)

        start_index = getattr(annotation, "start_index", None)
        end_index = getattr(annotation, "end_index", None)
        if start_index is not None and end_index is not None:
            replacements.append((start_index, end_index, replacement_text))
            continue

        annotated_text = getattr(annotation, "text", "")
        if annotated_text:
            text = text.replace(annotated_text, replacement_text)

    for start_index, end_index, replacement_text in sorted(replacements, reverse=True):
        text = f"{text[:start_index]}{replacement_text}{text[end_index:]}"

    return text, referenced_files


def main():
    # Initialize the project client
    load_dotenv()
    project_endpoint = os.environ.get("PROJECT_ENDPOINT")
    agent_name = os.environ.get("AGENT_NAME", "it-support-agent")

    if not project_endpoint:
        print("Error: PROJECT_ENDPOINT environment variable not set")
        print("Please set it in your .env file or environment")
        return

    print("Connecting to Microsoft Foundry project...")
    credential = DefaultAzureCredential()
    project_client = AIProjectClient(
        credential=credential,
        endpoint=project_endpoint
    )

    # Get the OpenAI client for Responses API
    openai_client = project_client.get_openai_client()

    # Get the agent created in the portal
    print(f"Loading agent: {agent_name}")
    agent = project_client.agents.get(agent_name=agent_name)
    print(f"Connected to agent: {agent.name} (id: {agent.id})")

    # Create a conversation
    conversation = openai_client.conversations.create(items=[])
```

```
print(f"Conversation created (id: {conversation.id})")

# Chat loop
print("\n" + "="*60)
print("IT Support Agent Ready!")
print("Ask questions, request data analysis, or get help.")
print("Type 'exit' to quit.")
print("="*60 + "\n")

while True:
    user_input = input("You: ").strip()

    if user_input.lower() in ['exit', 'quit', 'bye']:
        print("Goodbye!")
        break

    if not user_input:
        continue

        # Add user message to conversation
        openai_client.conversations.items.create(
            conversation_id=conversation.id,
            items=[{"type": "message", "role": "user", "content": user_input}]
        )

        # Get response from agent
        print("\n[Agent is thinking...]")
        response = openai_client.responses.create(
            conversation=conversation.id,
            extra_body={"agent_reference": {"name": agent.name, "type":
"agent_reference"}},
            input=""
        )

        # Display response and save any generated files locally
        handled_output = False
        downloaded_files = {}
        referenced_files = set()
        image_count = 0

    if hasattr(response, "output") and response.output:
        for item in response.output:
            item_type = getattr(item, "type", "")

            if item_type == "message" and getattr(item, "content", None):
                for content_item in item.content:
                    if getattr(content_item, "type", "") != "output_text":
                        continue

                    formatted_text, message_files = format_output_text(
                        content_item,
                        openai_client,
                        downloaded_files,
                    )
                    referenced_files.update(message_files)

                    if formatted_text:
                        print(f"\nAgent: {formatted_text}\n")
                        handled_output = True
```

```
            elif hasattr(item, "text") and item.text:
                print(f"\nAgent: {item.text}\n")
                handled_output = True

            elif item_type == "image":
                image_count += 1
                filename = f"chart_{image_count}.png"

                if hasattr(item, "image") and hasattr(item.image, "data"):
                    file_path = save_image(item.image.data, filename)
                    print(f"\n[Agent generated a chart - saved to: {file_path}]")
                else:
                    print("\n[Agent generated an image]")
                handled_output = True

    for file_path in downloaded_files.values():
        if file_path not in referenced_files:
            print(f"\n[Agent generated a file - saved to: {file_path}]")
            handled_output = True

        if not handled_output and hasattr(response, "output_text") and
response.output_text:
            print(f"\nAgent: {response.output_text}\n")

if __name__ == "__main__":
    main()
```

9. Save the `agent_with_functions.py` file (**Ctrl+S** or **File > Save**).

Configure environment and run the application

1. In the Explorer pane, you'll see `.env.example` and `requirements.txt` files already present in the folder.
2. Duplicate the `.env.example` file, and rename it to `.env`.
3. In the `.env` file, replace `your_project_endpoint_here` with your actual project endpoint:

Code Copy

```
PROJECT_ENDPOINT=<your_project_endpoint>
AGENT_NAME=it-support-agent
```

To get your project endpoint: In VS Code, open the **Foundry Toolkit** extension, right-click on your active project, and select **Copy Endpoint**. If **Copy Endpoint** isn't available in your installed version of Foundry Toolkit, open the Microsoft Foundry portal, go to your project, and copy the project endpoint from the project overview page instead.

4. Save the `.env` file (**Ctrl+S** or **File > Save**).
5. Open a terminal in VS Code (**Terminal > New Terminal**).
6. Install the required packages and login:

Code Copy

```
python -m venv labenv
.\labenv\Scripts\Activate.ps1
pip install -r requirements.txt
```

Code Copy

```
az login
```

7. Run the application:

Code Copy

```
python agent_with_functions.py
```

Test the client application

When the agent starts, try these prompts to test different capabilities:

1. Test policy search with file search:

Code Copy

```
What's the policy for password resets?
```

2. Request data analysis with code interpreter:

Code Copy

```
Analyze the system performance data and identify any periods where CPU usage exceeded 80%
```

3. Request a visualization:

Code Copy

```
Create a line chart showing memory usage trends over time
```

The application saves generated charts and cited files to the `agent_outputs` folder and prints the local file path in the terminal.

4. Ask for statistical analysis:

Code Copy

```
What are the average, minimum, and maximum values for disk usage in the performance data?
```

5. Combined analysis:

Code Copy

```
Find any correlation between high CPU usage and memory usage in the performance data
```

Observe how the agent uses both file search (for policy questions) and code interpreter (for data analysis) to fulfill your requests. The code interpreter will analyze the CSV data, perform calculations, and can even generate visualizations. Type `exit` when done testing.

Cleanup

To avoid unnecessary Azure charges, delete the resources you created:

1. In the Foundry portal, navigate to your project
2. Select **Settings** > **Delete project**
3. Alternatively, delete the entire resource group from the Azure portal