Lab 8 - Intelligent Agents with Azure AI Foundry and GPT-4o

Table of Contents:

Introduction - Setup

This workshop is designed to teach you about the Azure AI Agents Service and the associated Python SDK. It consists of multiple labs, each highlighting a specific feature of the Azure AI Agents Service. The labs are meant to be completed in order, as each one builds on the knowledge and work from the previous lab.

Prerequisites

1. Access to an Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
2. You need a GitHub account. If you don't have one, create it at GitHub.

Open the Workshop

The preferred way to run this workshop is using GitHub Codespaces. This option provides a pre-configured environment with all the tools and resources needed to complete the workshop. Alternatively, you can open the workshop locally using a Visual Studio Code Dev Container.

GitHub Codespaces

1. Go to the repository
2. Select the green "<> Code" button, a dropdown menu should appear.
3. Select the "Codespaces" tab.
4. Select the "+" button which will create your codespace environment.

VSCode Dev Container

Alternatively, you can open the project locally using a Visual Studio Code Dev Container, which will open the project in your local VS Code development environment using the Dev Containers extension

Additional Requirements:

1. Docker Desktop
2. git

Instructions

1. Start Docker Desktop
2. Clone the repository locally "`git clone https://github.com/microsoft/build-your-first-agent-with-azure-ai-agent-service-workshop.git`"
3. Start VSCode and open the folder that you cloned the repository into. VSCode will recognize that there is a devcontainer file and ask if you want to reopen to develop in a container. Select "Reopen in Container"

Lab Structure

Each lab in this workshop includes:

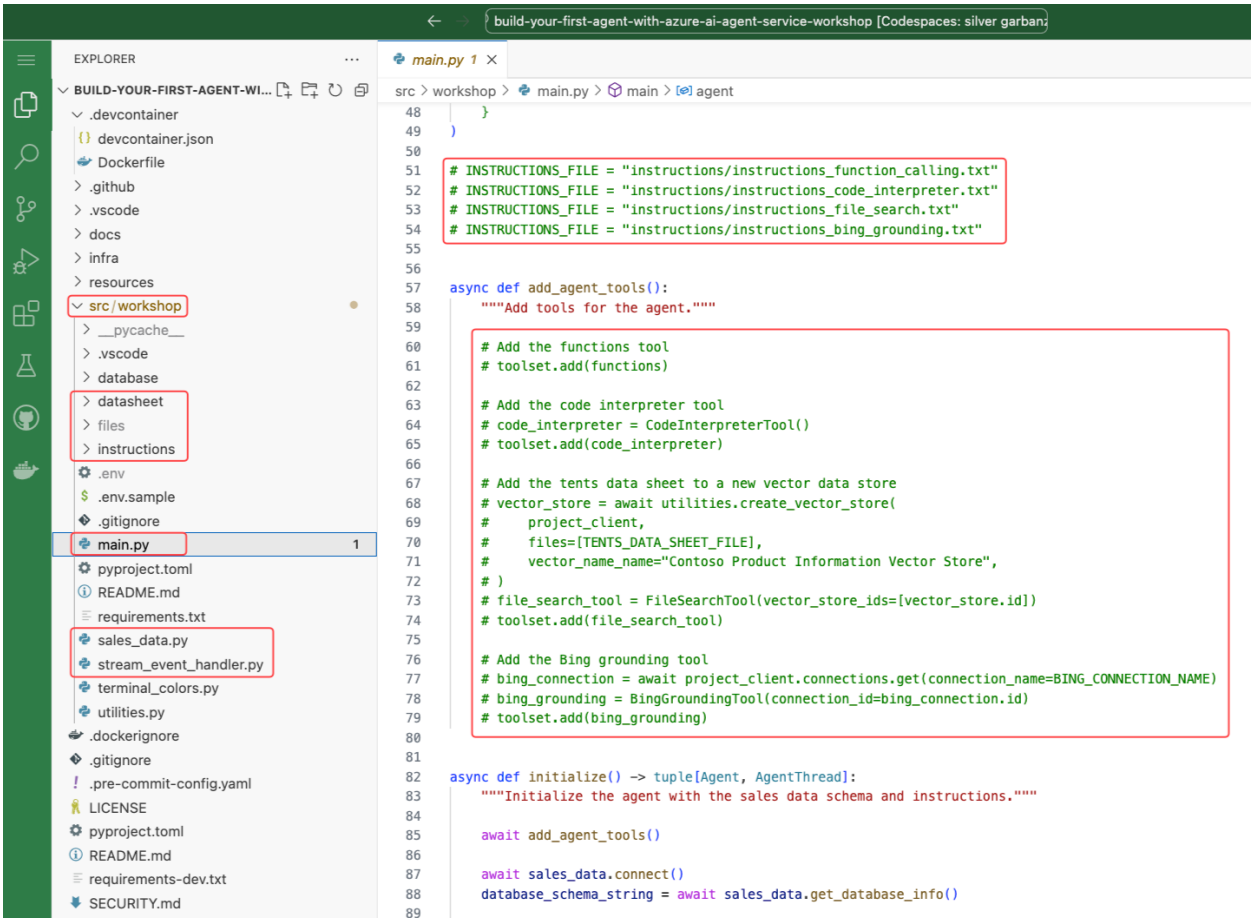
- An **Introduction**: Explains the relevant concepts.
- An **Exercise**: Guides you through the process of implementing the feature.

Project Structure

The workshop's source code is located in the **src/workshop** folder. Be sure to familiarize yourself with the key **subfolders** and **files** you'll be working with throughout the workshop.

1. The **files** folder: Contains the files created by the agent app. The files folder is created during agent execution and is not checked into source control. As a result, you will NOT see this folder in your forked repository - but you will see it during runtime.
2. The **instructions** folder: Contains the instructions passed to the LLM.
3. The **main.py** file: The entry point for the app, containing its main logic.
4. The **sales_data.py** file: The function logic to execute dynamic SQL queries against the SQLite database.

5. The **stream_event_handler.py** file: Contains the event handler logic for token streaming.



Authenticate with Azure

You need to authenticate with Azure so the agent app can access the Azure AI Agents Service and models. Follow these steps:

1. Ensure the Codespace has been created.
2. In the Codespace, open a new terminal window by selecting **Terminal** > **New Terminal** from the **VS Code menu**.
3. Run the following command to authenticate with Azure:

   ```
   az login --use-device-code
   ```

Note: You'll be prompted to open a browser link and log in to your Azure account. Be sure to copy the authentication code first.

- A browser window will open automatically, select your account type and click “Next”.
- Sign in with your Azure subscription “Username” and “Password”.
- Paste the authentication code.
- Select “OK”, then “Done”.

Warning: If you have multiple Azure tenants, then you will need to select the appropriate tenant when authenticating.

```
az login --use-device-code --tenant <tenant_id>
```

4. Next, select the appropriate subscription from the command line.
5. Leave the terminal window open for the next steps.

Deploy the Azure Resources

The following resources will be created in the rg-contoso-agent-workshop resource group in your Azure subscription.

- An **Azure AI Foundry hub** named **agent-wksp**
- An **Azure AI Foundry project** named **Agent Service Workshop**
- A **Serverless (pay-as-you-go) GPT-4o model deployment** named **gpt-4o (Global 2024-08-06)**. See pricing details here
- A **Grounding with Bing Search** resource. See the documentation and pricing for details.

Warning! "You will need **140K TPM** quota availability for the **gpt-4o Global Standard SKU**, not because the agent uses lots of tokens, but due to the frequency of calls made by the agent to the model. Review your quota availability in the AI Foundry Management Center

We have provided a bash/Powershell script to automate the deployment of the resources required for the workshop. Alternatively, you may deploy resources manually using Azure AI Foundry studio. Select the desired tab.

Automated deployment

The script `deploy.sh` deploys to the `eastus2` region by default; edit the file to change the region or resource names. To run the script, open the VS Code terminal and run the following command:

```
cd infra && ./deploy.sh
```

Workshop Configuration File

The deploy script generates the “src/workshop/.env” file, which contains the project connection string, model deployment name, and Bing connection name.

Your “.env” file should look similar to this but with your project connection string.

MODEL_DEPLOYMENT_NAME="gpt-4o"

BING_CONNECTION_NAME="groundingwithbingsearch"

PROJECT_CONNECTION_STRING="<your_project_connection_string>"

Introduction – Part 1 – Function Calling

Function calling enables Large Language Models to interact with external systems. The LLM determines when to invoke a function based on instructions, function definitions, and user prompts. The LLM then returns structured data that can be used by the agent app to invoke a function.

It's up to the developer to implement the function logic within the agent app. In this workshop, we use function logic to execute SQLite queries that are dynamically generated by the LLM.

Enabling Function Calling

If you're familiar with Azure OpenAI Function Calling, it requires defining a function schema for the LLM. Azure AI Agent Service supports this approach and also offers a more flexible option.

With the Azure AI Agent Service and its Python SDK, you can define the function schema directly within the Python function's docstring. This approach keeps the definition and implementation together, simplifying maintenance and enhancing readability.

For example, in the sales_data.py file, the async_fetch_sales_data_using_sqlite_query function uses a docstring to specify its signature, inputs, and outputs. The SDK parses this docstring to generate the callable function for the LLM:

```
async def async_fetch_sales_data_using_sqlite_query(self: "SalesData", sqlite_query: str) -
> str:
    """
    This function is used to answer user questions about Contoso sales data by executing
SQLite queries against the database.

:param sqlite_query: The input should be a well-formed SQLite query to extract information
based on the user's question. The query result will be returned as a JSON object.
    :return: Return data in JSON serializable format.
```

```
    :rtype: str
    """
```

Dynamic SQL Generation

When the app starts, it incorporates the database schema and key data into the instructions for the Azure AI Agent Service. Using this input, the LLM generates SQLite-compatible SQL queries to respond to user requests expressed in natural language.

Lab Exercise

In this lab, you'll enable the function logic to execute dynamic SQL queries against the SQLite database. The function is called by the LLM to answer user questions about Contoso sales data.

Open the main.py.

Uncomment the following lines by removing the "# " characters

```
# INSTRUCTIONS_FILE = "instructions/instructions_function_calling.txt"
# toolset.add(functions)
```

Warning: The lines to be uncommented are not adjacent. When removing the # character, ensure you also delete the space that follows it.

After uncommenting, your code should look like this:

```
INSTRUCTIONS_FILE = "instructions/instructions_function_calling.txt"
# INSTRUCTIONS_FILE = "instructions/instructions_code_interpreter.txt"
# INSTRUCTIONS_FILE = "instructions/instructions_file_search.txt"
# INSTRUCTIONS_FILE = "instructions/instructions_bing_grounding.txt"

async def add_agent_tools():
    """Add tools for the agent."""

# Add the functions tool
    toolset.add(functions)

# Add the code interpreter tool
    # code_interpreter = CodeInterpreterTool()
    # toolset.add(code_interpreter)
```

```
# Add the tents data sheet to a new vector data store
   # vector_store = await utilities.create_vector_store(
   #    project_client,
   #    files=[TENTS_DATA_SHEET_FILE],
   #    vector_name_name="Contoso Product Information Vector Store",
   # )
   # file_search_tool = FileSearchTool(vector_store_ids=[vector_store.id])
   # toolset.add(file_search_tool)

# Add the Bing grounding tool
   # bing_connection = await
project_client.connections.get(connection_name=BING_CONNECTION_NAME)
   # bing_grounding = BingGroundingTool(connection_id=bing_connection.id)
   # toolset.add(bing_grounding)
```

Review the Instructions

Open the src/workshop/instructions/instructions_function_calling.txt file.

Tip: In VS Code, press Alt + Z (Windows/Linux) or Option + Z (Mac) to enable word wrap mode.

Review how the instructions define the agent app's behavior including role definition, context, tool descriptions, response guidance, safety and conduct, etc.

Run the Agent App

Press F5 to run the app.

In the terminal, you'll see the app start, and the agent app will prompt you to Enter your query.

Start a Conversation with the Agent

Ask questions like:
- What were the sales by region?
- What was last quarter's revenue?
- Which products sell best in Europe?
- Total shipping costs by region?

Tip: Try asking for help in different languages.

Query: Show the 3 most recent transaction details

Warning: The agent may refuse this query due to default behavior requiring aggregated results. Reword if necessary.

Query: What are the sales by region?

Example SQL: SELECT region, SUM(revenue) AS total_revenue FROM sales_data GROUP BY region;

The LLM:
1. Generates SQL
2. Calls async_fetch_sales_data_using_sqlite_query
3. Returns data as a Markdown table

Query: Show sales by category in Europe

Query: Breakout sales by footwear

Query: Show sales by region as a pie chart (not supported yet)

(Optional) Debug the App

Set a breakpoint in async_fetch_sales_data_using_sqlite_query in sales_data.py.

Note: Exit the app first. Then set the breakpoint. Then run using the debugger icon.

Ask More Questions

Try:
- What regions have the highest sales?
- What were the sales of tents in the United States in April 2022?

Disable the Breakpoint

Remember to disable the breakpoint before running again.

Stop the Agent App

Type 'exit' to stop the app and clean up resources.

Introduction – Part 2 – Code Interpreter

The Azure AI Agent Service Code Interpreter enables the LLM to safely execute Python code for tasks such as creating charts or performing complex data analyses based on user queries. It makes use of natural language processing (NLP), sales data from an SQLite database, and user prompts to automate code generation. The LLM-generated Python code executes within a secure sandbox environment, running on a restricted subset of Python to ensure safe and controlled execution.

Lab Exercise

In this lab, you'll enable the Code Interpreter to execute Python code generated by the LLM.

1. Open the `main.py`.

2. Define a new instructions file for our agent: uncomment the following lines by removing the “"# "“ characters:

```
# INSTRUCTIONS_FILE = "instructions/instructions_code_interpreter.txt

# code_interpreter = CodeInterpreterTool()
    # toolset.add(code_interpreter)
```

Warning: The lines to be uncommented are not adjacent. When removing the # character, ensure you also delete the space that follows it.

3. Review the code in the `main.py` file. After uncommenting, your code should look like this:

```
INSTRUCTIONS_FILE = "instructions/instructions_function_calling.txt"
INSTRUCTIONS_FILE = "instructions/instructions_code_interpreter.txt"
# INSTRUCTIONS_FILE = "instructions/instructions_file_search.txt"
# INSTRUCTIONS_FILE = "instructions/instructions_bing_grounding.txt"
```

```
async def add_agent_tools():
    """Add tools for the agent."""

# Add the functions tool
    toolset.add(functions)

# Add the code interpreter tool
    code_interpreter = CodeInterpreterTool()
    toolset.add(code_interpreter)

# Add the tents data sheet to a new vector data store
    # vector_store = await utilities.create_vector_store(
    #    project_client,
    #    files=[TENTS_DATA_SHEET_FILE],
    #    vector_name_name="Contoso Product Information Vector Store",
    # )
    # file_search_tool = FileSearchTool(vector_store_ids=[vector_store.id])
    # toolset.add(file_search_tool)

# Add the Bing grounding tool
    # bing_connection = await
project_client.connections.get(connection_name=BING_CONNECTION_NAME)
    # bing_grounding = BingGroundingTool(connection_id=bing_connection.id)
    # toolset.add(bing_grounding)
```

Review the Instructions

1. Open the “src/workshop/instructions/instructions_code_interpreter.txt” file. This file replaces the instructions used in the previous lab.
2. The Tools section now includes a “Visualization and Code Interpretation” capability, allowing the agent to:

- Use the code interpreter to run programs generated by the LLM (e.g., for downloading or visualizing data).
  - Create charts and graphs, using the user’s language for labels, titles, and other chart text.
  - Export visualizations as PNG files and data as CSV files.

Run the Agent App

1. Press F5 to run the app.
2. In the terminal, the app will start, and the agent app will prompt you to “Enter your query”.

Start a Conversation with the Agent

Try these questions:

1. “Show sales by region as a pie chart”

The pie chart image will be saved in the “files” subfolder. Open it in VS Code to view.

Info on what is happening:
- SQL generated: `SELECT region, SUM(revenue) AS total_revenue FROM sales_data GROUP BY region;`
- LLM calls `async_fetch_sales_data_using_sqlite_query`
- LLM writes and runs Python code to create a pie chart

2. Download the sales by region data

Result saved in the files folder.

Info on what is happening:
- Default format is CSV
- Other formats like JSON or Excel can be requested

3. Download as JSON

The agent infers your intent even without explicit instructions.

4. Continue asking questions about Contoso sales data to see the Code Interpreter in action.

Stop the Agent App

When you're done, type exit to clean up the agent resources and stop the app.

Introduction – Part 3 – File Search

Grounding a conversation with documents is highly effective, especially for retrieving product details that may not be available in an operational database. The Azure AI Agent Service includes a File Search tool, which enables agents to retrieve information directly from uploaded files, such as user-supplied documents or product data, enabling a RAG-style search experience.

In this lab, you'll learn how to enable the document search and upload the Tents Data Sheet to a vector store for the agent. Once activated, the tool allows the agent to search the file and deliver relevant responses. Documents can be uploaded to the agent for all users or linked to a specific user thread, or linked to the Code Interpreter.

When the app starts, a vector store is created, the Contoso tents datasheet PDF file is uploaded to the vector store, and it is made available to the agent.

Normally, you wouldn't create a new vector store and upload documents each time the app starts. Instead, you'd create the vector store once, upload potentially thousands of documents, and connect the store to the agent.

A vector store is a database optimized for storing and searching vectors (numeric representations of text data). The File Search tool uses the vector store for semantic search to search for relevant information in the uploaded document.

Lab Exercise

1. Open the "src/workshop/datasheet/contoso-tents-datasheet.pdf" file from VS Code.

2. Review the file's contents to understand the information it contains.

Info: The PDF file includes detailed product descriptions for the tents sold by Contoso.

3. Open the file `main.py`.

4. Uncomment the following lines:

```
# INSTRUCTIONS_FILE = "instructions/instructions_file_search.txt"
```

```
# vector_store = await utilities.create_vector_store(
  #    project_client,
  #    files=[TENTS_DATA_SHEET_FILE],
  #    vector_name_name="Contoso Product Information Vector Store",
  # )
  # file_search_tool = FileSearchTool(vector_store_ids=[vector_store.id])
  # toolset.add(file_search_tool)
```

5. Review the code in `main.py`. After uncommenting, your code should look like:

```
  INSTRUCTIONS_FILE = "instructions/instructions_function_calling.txt"
  INSTRUCTIONS_FILE = "instructions/instructions_code_interpreter.txt"
  INSTRUCTIONS_FILE = "instructions/instructions_file_search.txt"
  # INSTRUCTIONS_FILE = "instructions/instructions_bing_grounding.txt"

async def add_agent_tools():
    """Add tools for the agent."""

# Add the functions tool
    toolset.add(functions)

# Add the code interpreter tool
    code_interpreter = CodeInterpreterTool()
    toolset.add(code_interpreter)

# Add the tents data sheet to a new vector data store
    vector_store = await utilities.create_vector_store(
      project_client,
      files=[TENTS_DATA_SHEET_FILE],
      vector_name_name="Contoso Product Information Vector Store",
    )
    file_search_tool = FileSearchTool(vector_store_ids=[vector_store.id])
    toolset.add(file_search_tool)

# Add the Bing grounding tool
    # bing_connection = await
project_client.connections.get(connection_name=BING_CONNECTION_NAME)
    # bing_grounding = BingGroundingTool(connection_id=bing_connection.id)
    # toolset.add(bing_grounding)
```

Review the Instructions

1. Open the “src/workshop/instructions/instructions_file_search.txt” file.

2. The Tools section now includes a “Contoso Product Information Vector Store” capability:

- Search the vector store for additional Contoso product information.
  - Access this resource via the “FileSearchTool” SDK function defined in “main.py”.

Run the Agent App

1. Review the “create_vector_store” function in “utilities.py”. It uploads the Tents Data Sheet to a vector store.

Tip: You can set a breakpoint in this function if using the VS Code debugger.

2. Press F5 to run the app.
3. In the terminal, the app starts, and the agent prompts you to “Enter your query”.

Start a Conversation with the Agent

Try these queries:

1. “What brands of hiking shoes do we sell?”

Info: No files with hiking shoe info have been provided.

2. “What brands of tents do we sell?”

Agent pulls this from the Tents Data Sheet.

3. “What product type and categories are these brands associated with?”

4. “What were the sales of tents in 2024 by product type? Include the brands associated with each.”

Info: The agent might get this wrong. You can add more context in the instructions or datasheet.

5. “What were the sales of AlpineGear in 2024 by region?”

Info: Agent connects AlpineGear as a backpacking tent brand using the Tents Data Sheet.

6. “Contoso does not sell Family Camping tents from AlpineGear. Try again.”

Now the agent should correct itself.

Stop the Agent App

Type “exit” to clean up resources and stop the app.

Introduction – Part 4 – Grounding with Bing

Grounding conversations with Bing is one of several tools provided by the Azure AI Agent Service. Grounding with Bing allows your app to search for information relevant to the conversation. For example, you might want to search for competitive product information.

Access to Grounding with Bing Search:
This lab requires the Grounding with Bing Search service, which may not be available in your Azure subscription. Follow the lab instructions to check or create the resource. Even if you can’t use it, the lab explains how it works.

Lab Exercise

In this lab, you'll enable Bing Grounding to provide competitive sales analysis of Contoso products and categories.

Create a Grounding with Bing Search Resource

1. Go to the Azure portal to create the resource.
2. Select “Create” and choose “rg-agent-workshop” as the resource group.
3. Name the resource: `groundingwithbingsearch`.
4. Choose the “Grounding with Bing Search” pricing tier.
5. Accept terms, click “Review + create”, then “Create”.
6. After deployment, select “Go to resource”, then click “Go to Azure AI Foundry Portal”.

Create a Bing Search Connection in AI Foundry

1. In Azure AI Foundry, ensure “Agent-Service-Workshop” project is selected.
2. Go to “Management Center” → “Connected resources” → “+ New connection”.
3. Choose “Grounding with Bing Search” under Knowledge section.
4. Add the connection linked to `groundingwithbingsearch`.
5. Click “Close”.

Enable Grounding with Bing Search in the Agent App

1. Open `main.py`.

2. Uncomment these lines:

```
  # INSTRUCTIONS_FILE = "instructions/instructions_bing_grounding.txt"

# bing_connection =
project_client.connections.get(connection_name=BING_CONNECTION_NAME)
  # bing_grounding = BingGroundingTool(bing_connection)
  # toolset.add(bing_grounding)
```

3. After uncommenting, your code should look like:

```
  INSTRUCTIONS_FILE = "instructions/instructions_function_calling.txt"
  INSTRUCTIONS_FILE = "instructions/instructions_code_interpreter.txt"
  INSTRUCTIONS_FILE = "instructions/instructions_file_search.txt"
  INSTRUCTIONS_FILE = "instructions/instructions_bing_grounding.txt"

async def add_agent_tools():
    """Add tools for the agent."""

# Add the functions tool
    toolset.add(functions)

# Add the code interpreter tool
    code_interpreter = CodeInterpreterTool()
    toolset.add(code_interpreter)

# Add the tents data sheet
    vector_store = await utilities.create_vector_store(
      project_client,
      files=[TENTS_DATA_SHEET_FILE],
      vector_name_name="Contoso Product Information Vector Store",
    )
    file_search_tool = FileSearchTool(vector_store_ids=[vector_store.id])
    toolset.add(file_search_tool)

# Add the Bing grounding tool
    bing_connection = await
project_client.connections.get(connection_name=BING_CONNECTION_NAME)
    bing_grounding = BingGroundingTool(connection_id=bing_connection.id)
    toolset.add(bing_grounding)
```

Review the Instructions

1. Open `src/workshop/instructions/instructions_bing_grounding.txt`.
2. Tools section includes “Competitive Insights for Products and Categories”, which:
   - Uses Bing Search for competitor names, products, and prices.
   - Limits to outdoor camping and sports gear.
   - Ensures concise, relevant search results.

Run the Agent App

Press F5 to start the app. You'll be prompted to enter a query.

Start a Conversation with the Agent

Try these queries:

1. “What beginner tents do we sell?”
   - Info comes from vector store.

2. “What beginner tents do our competitors sell? Include prices.”
   - Info comes from Bing.

3. “Show as a bar chart”
   - Code Interpreter creates chart from Bing data. Output saved in `src/workshop/files`.

4. “Show the tents we sell by region that are a similar price to our competitors beginner tents.”
   - Combines Bing + database + reasoning.

5. “Download the data as a human-readable JSON file”
   - Uses Code Interpreter to export based on prior context.

Stop the Agent App

1. Type “save” to preserve app state.
2. Press “Shift+F5” to stop debugging.
3. Copy the Agent ID from terminal output. It will look like:
   `Agent ID: asst_pskNeFYuoCPogDnmfaqIUwoU`

Explore the Agent in Azure AI Foundry

1. Go to the [Azure AI Foundry portal](https://ai.azure.com/).
2. Select “Playgrounds” → “Try the Agents playground”.
3. Paste your “Agent ID” into the “Agent id” field.

Review the agent's behavior and instructions. Try previous queries or new ones. Note: The playground does not access the Contoso database.