Develop an Azure AI chat agent with the Microsoft Agent Framework SDK

In this exercise, you'll use Azure AI Agent Service and Microsoft Agent Framework to create an AI agent that processes expense claims.

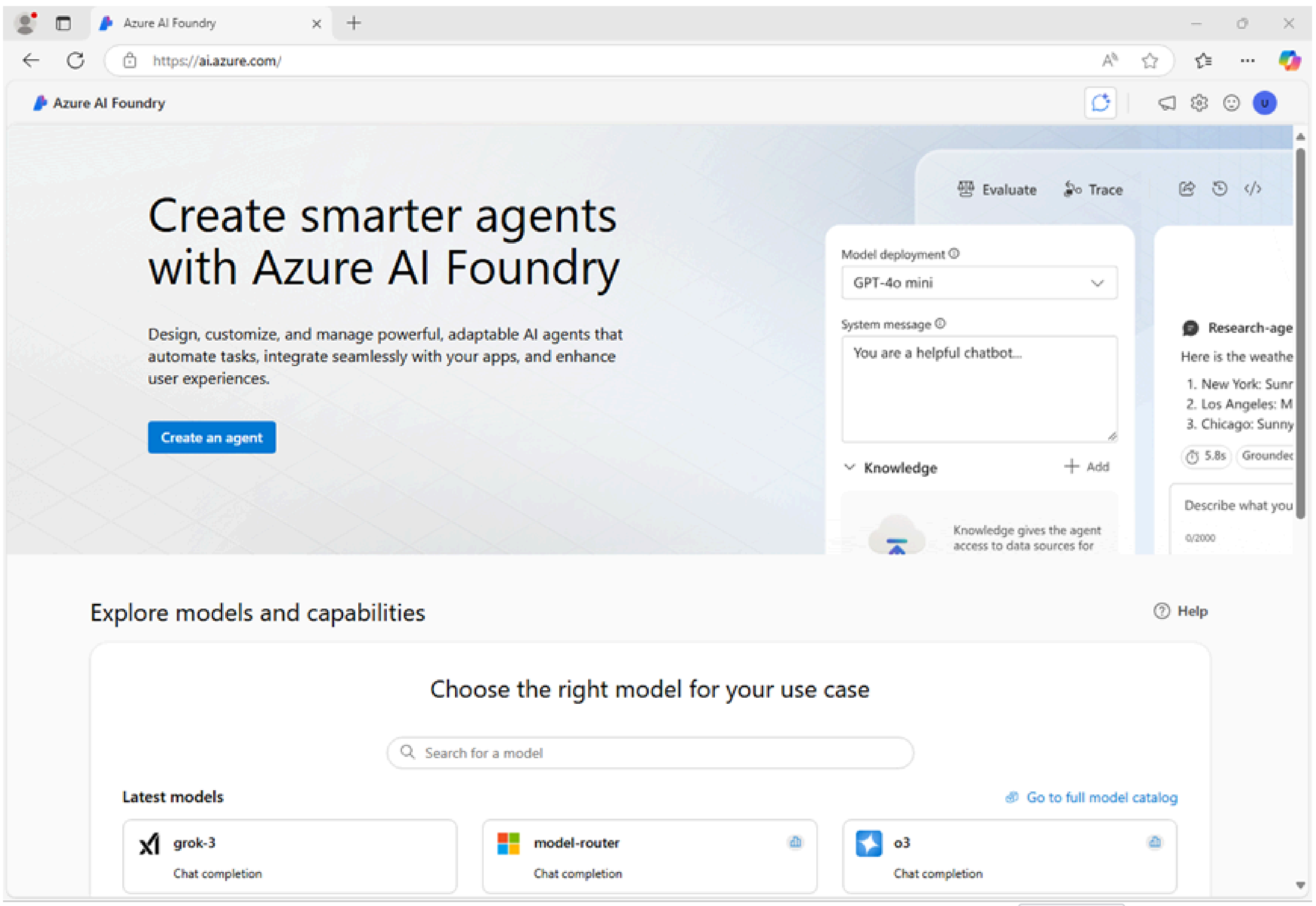
This exercise should take approximately **30** minutes to complete.

> **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.

Deploy a model in an Azure AI Foundry project

Let's start by deploying a model in 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):

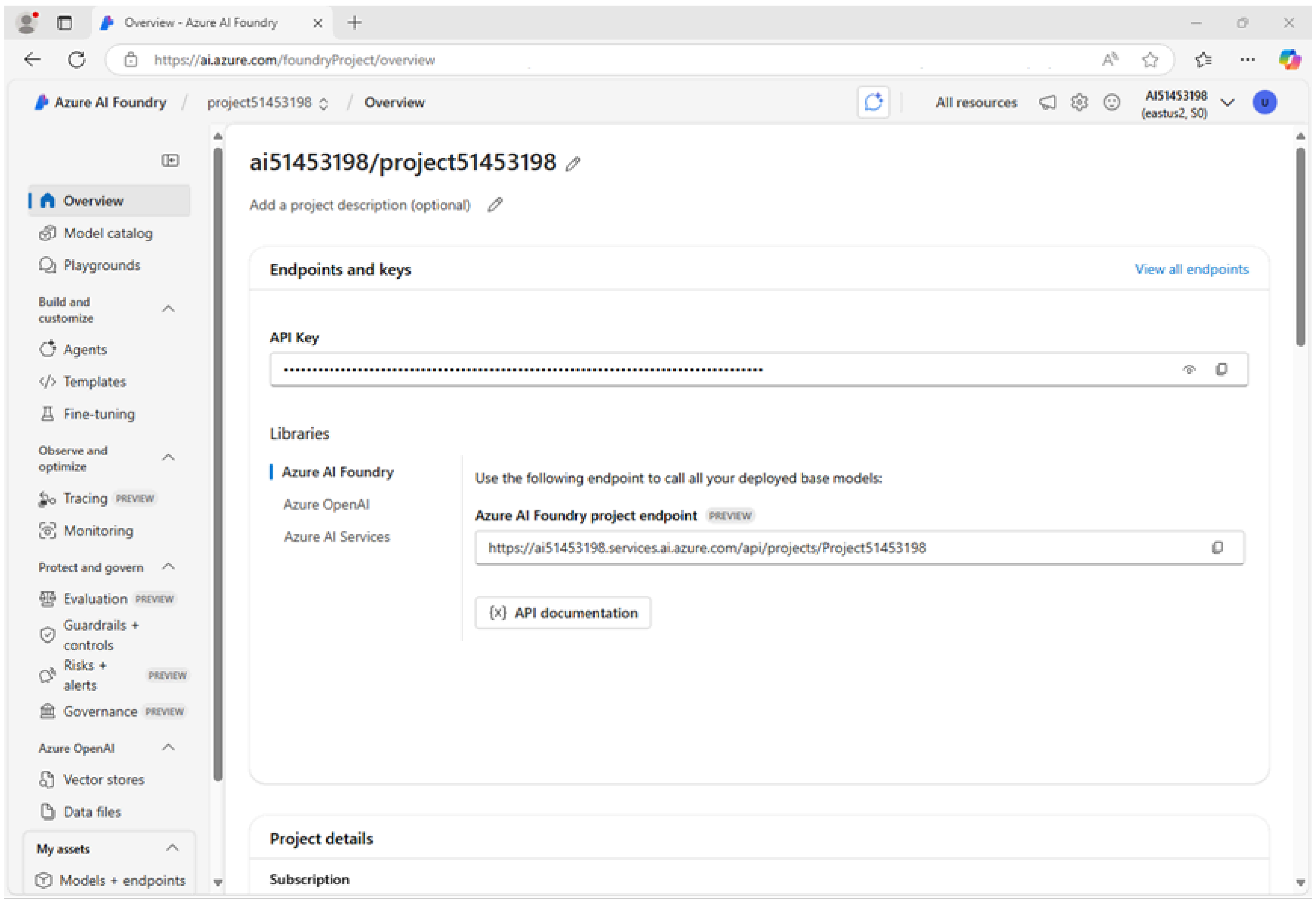

Azure AI Foundry
https://ai.azure.com/
Azure AI Foundry
Create smarter agents with Azure AI Foundry
Design, customize, and manage powerful, adaptable AI agents that automate tasks, integrate seamlessly with your apps, and enhance user experiences.
Create an agent
Evaluate
Trace
Model deployment
GPT-4o mini
System message
You are a helpful chatbot...
Knowledge
Add
Explore models and capabilities
Help
Choose the right model for your use case
Search for a model
Latest models
Go to full model catalog
grok-3
Chat completion
model-router
Chat completion
o3
Chat completion

2. On the home page, in the **Explore models and capabilities** section, search for the `gpt-4o` model; which we'll use in our project.
3. In the search results, select the **gpt-4o** model to see its details, and then at the top of the page for the model, select **Use this model**.
4. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
5. 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 Foundry recommended****

   > * 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.

6. Select **Create** and wait for your project, including the gpt-4 model deployment you selected, to be created.
7. When your project is created, the chat playground will be opened automatically.
8. In the **Setup** pane, note the name of your model deployment; which should be **gpt-4o**. You can confirm this by viewing the deployment in the **Models and endpoints** page (just open that page in the navigation pane on the left).

9. In the navigation pane on the left, select **Overview** to see the main page for your project; which looks like this:



ai51453198/project51453198

Add a project description (optional)

Endpoints and keys — View all endpoints

API Key: ••

Libraries: Azure AI Foundry | Azure OpenAI | Azure AI Services

Use the following endpoint to call all your deployed base models:

Azure AI Foundry project endpoint PREVIEW

https://ai51453198.services.ai.azure.com/api/projects/Project51453198

API documentation

Project details

Subscription

Create an agent client app

Now you're ready to create a client app that defines an agent and a custom function. Some code has been provided for you in a GitHub repository.

Prepare the environment

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. When the repo has been cloned, 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/04-agent-framework/python
   ls -a -l
   ```

   The provided files include application code a file for configuration settings, and a file containing expenses data.

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 azure-identity agent-framework
   ```

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 the **your_model_deployment** placeholder with the name you assigned to your gpt-4o model deployment.
4. After you've replaced the placeholders, 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.

Write code for an agent app

> **Tip**: As you add code, be sure to maintain the correct indentation. Use the existing comments as a guide, entering the new code at the same level of indentation.

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

   Code Copy

   ```
   code agent-framework.py
   ```

2. Review the code in the file. It contains:
    - Some **import** statements to add references to commonly used namespaces
    - A *main* function that loads a file containing expenses data, asks the user for instructions, and and then calls…
    - A **process_expenses_data** function in which the code to create and use your agent must be added
3. At the top of the file, after the existing **import** statement, find the comment **Add references**, and add the following code to reference the namespaces in the libraries you'll need to implement your agent:

Code Copy

```
# Add references
from agent_framework import AgentThread, ChatAgent
from agent_framework.azure import AzureAIAgentClient
from azure.identity.aio import AzureCliCredential
from pydantic import Field
from typing import Annotated
```

4. Near the bottom of the file, find the comment **Create a tool function for the email functionality**, and add the following code to define a function that your agent will use to send email (tools are a way to add custom functionality to agents)

Code Copy

```
# Create a tool function for the email functionality
def send_email(
 to: Annotated[str, Field(description="Who to send the email to")],
 subject: Annotated[str, Field(description="The subject of the email.")],
 body: Annotated[str, Field(description="The text body of the email.")]):
     print("\nTo:", to)
     print("Subject:", subject)
     print(body, "\n")
```

Note: The function *simulates* sending an email by printing it to the console. In a real application, you'd use an SMTP service or similar to actually send the email!

5. Back up above the **send_email** code, in the **process_expenses_data** function, find the comment **Create a chat agent**, and add the following code to create a **ChatAgent** object with the tools and instructions.

(Be sure to maintain the indentation level)

Code Copy

```
# Create a chat agent
async with (
    AzureCliCredential() as credential,
    ChatAgent(
        chat_client=AzureAIAgentClient(async_credential=credential),
        name="expenses_agent",
        instructions="""You are an AI assistant for expense claim submission.
                        When a user submits expenses data and requests an expense claim, use
the plug-in function to send an email to expenses@contoso.com with the subject 'Expense
Claim`and a body that contains itemized expenses with a total.
                        Then confirm to the user that you've done so.""",
        tools=send_email,
    ) as agent,
):
```

Note that the **AzureCliCredential** object will allow your code to authenticate to your Azure account. The **AzureAIAgentClient** object will automatically include the Azure AI Foundry project settings from the .env configuration.

6. Find the comment **Use the agent to process the expenses data**, and add the following code to create a thread for your agent to run on, and then invoke it with a chat message.

   (Be sure to maintain the indentation level):

   Code Copy

```
# Use the agent to process the expenses data
try:
    # Add the input prompt to a list of messages to be submitted
    prompt_messages = [f"{prompt}: {expenses_data}"]
    # Invoke the agent for the specified thread with the messages
    response = await agent.run(prompt_messages)
    # Display the response
    print(f"\n# Agent:\n{response}")
except Exception as e:
    # Something went wrong
    print (e)
```

7. Review that the completed code for your agent, using the comments to help you understand what each block of code does, and then save your code changes (**CTRL+S**).
8. Keep the code editor open in case you need to correct any typo's in the code, but resize the panes so you can see more of the command line console.

Sign into Azure and run the app

1. In the cloud shell command-line pane beneath the code editor, 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 agent-framework.py
   ```

   The application runs using the credentials for your authenticated Azure session to connect to your project and create and run the agent.

4. When asked what to do with the expenses data, enter the following prompt:

   Code Copy

   ```
   Submit an expense claim
   ```

5. When the application has finished, review the output. The agent should have composed an email for an expenses claim based on the data that was provided.

   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.

Summary

In this exercise, you used the Microsoft Agent Framework SDK to create an agent with a custom tool.

Clean up

If you've finished exploring Azure AI Agent Service, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. Return to the browser tab containing the Azure portal (or re-open the Azure portal at `https://portal.azure.com` in a new browser tab) and view the contents of the resource group where you deployed the 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.