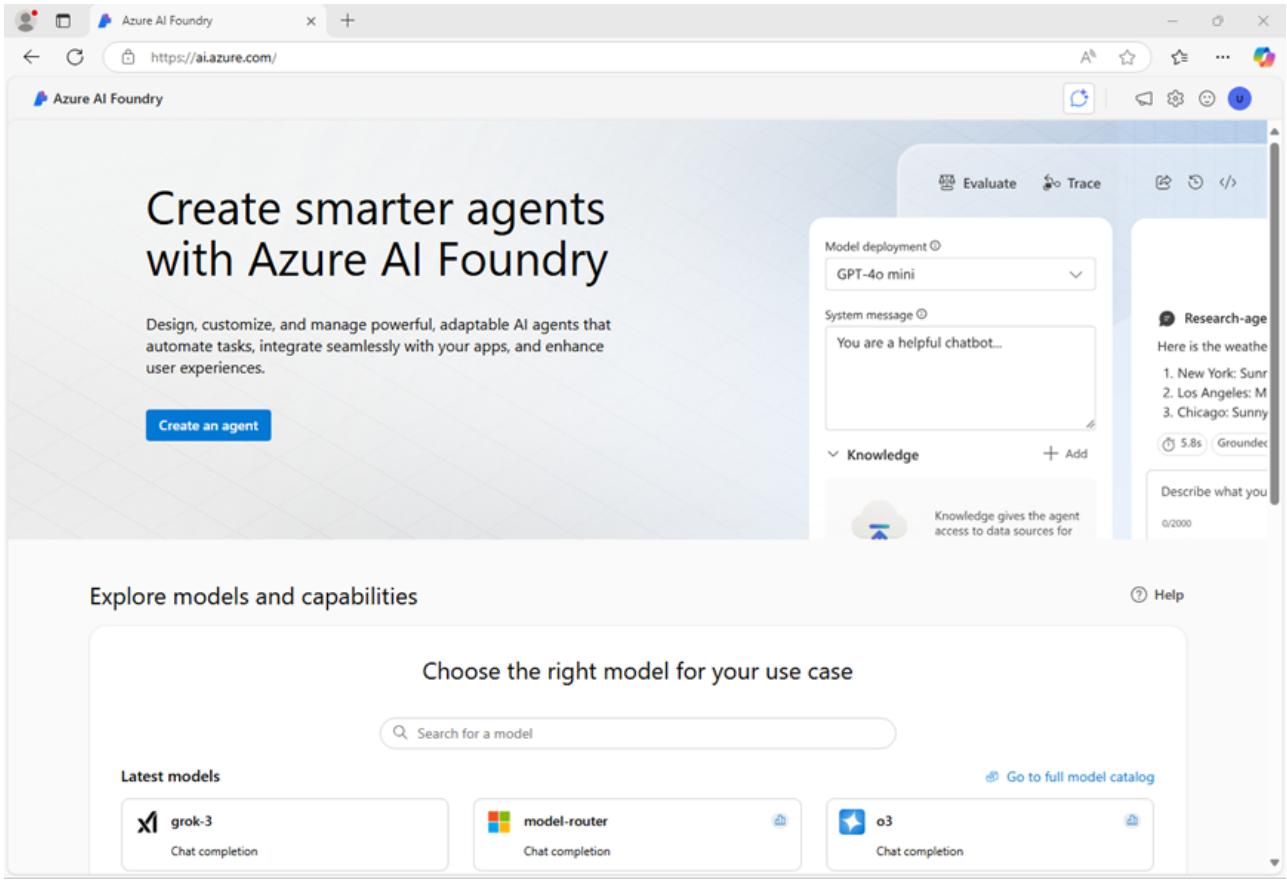
Build a Semantic Kernel object

In this exercise, you create an Azure OpenAI resource and provide the deployment information to the Semantic Kernel. The Semantic Kernel will connect to the endpoint and allow you to run prompts to the large language model (LLM) directly from your code. Let's get started!

This exercise takes approximately **10** minutes to complete.

Deploy a chat completion model

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, 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. Select **Customize** and specify the following settings for your hub:

- **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****

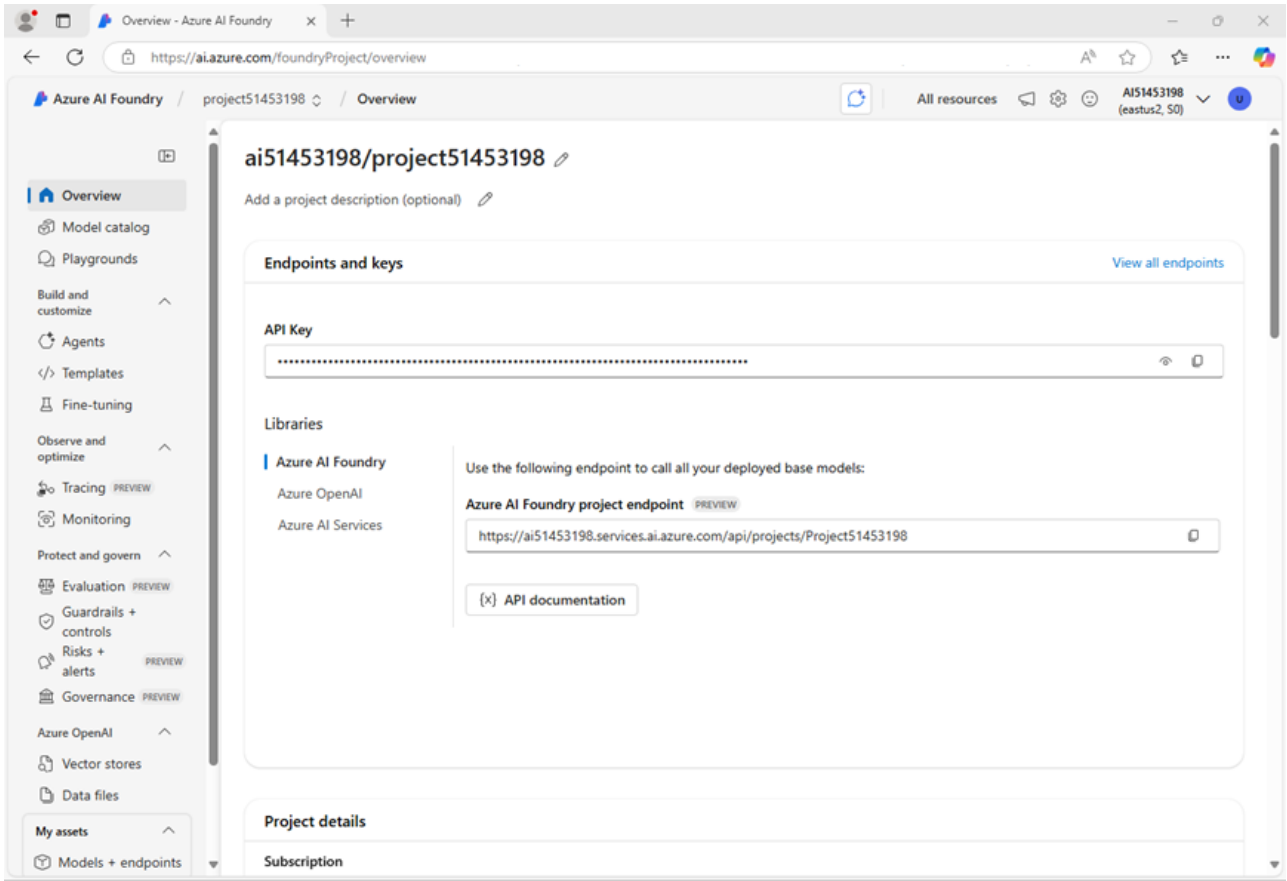
> * Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

6. Select **Create** and wait for your project, including the gpt-4o model deployment you selected, to be created.

7. When your project is created, the chat playground will be opened automatically.

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

> **Note**: If an *Insufficient permissions** error is displayed, use the **Fix me** button to resolve it.



9. Under the **Libraries** section of the overview page, select **Azure OpenAI**

   You'll use the data here in the next task to build your kernel. Remember to keep your keys private and secure!

Create a Semantic Kernel client application

Now that you deployed a model, you can use the Semantic Kernel SDK to create a client application that chats with it. Let's get started!

Configure your application

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 mslearn-ai-semantic-kernel -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-semantic-kernel
   mslearn-ai-semantic-kernel
   ```

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

5. After the repo has been cloned, navigate to the folder containing the application code files:

   > **Note**: Follow the steps for your chosen programming language.

   Python

   ```
   cd mslearn-ai-semantic-kernel/Labfiles/01-build-your-kernel/Python
   ```

   C#

   ```
   cd mslearn-ai-semantic-kernel/Labfiles/01-build-your-kernel/C-sharp
   ```

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

   Python

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install python-dotenv semantic-kernel[azure]
   ```

   C#

   ```
   dotnet add package Microsoft.SemanticKernel
   ```

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

   Python

   ```
   code .env
   ```

   C#

   ```
   code appsettings.json
   ```

   The file should open in a code editor.

8. In the code file, replace the **your_project_endpoint** and **your_project_api_key** placeholders with the Azure OpenAI endpoint and API key for your project (copied from the project **Overview** page in the Azure AI Foundry portal), and replace the **your_deployment_name** placeholder with the name you assigned to your gpt-4o model.

9. After you replace the placeholders, in the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes and then use the **CTRL+Q** command or **Right-click > Quit** to close the code editor while keeping the cloud shell command line open.

Run a prompt with Semantic Kernel

1. Enter the following commands to edit the code file:

   Python

   ```
   code kernel.py
   ```

   C#

```
code Program.cs
```

2. Under the comment **Import namespaces**, add the following language-specific code to import the namespaces you'll need to use the Semantic Kernel SDK.

Python

```
# Import namespaces
from semantic_kernel import Kernel
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion
from semantic_kernel.functions.kernel_arguments import KernelArguments
```

C#

```
using Microsoft.SemanticKernel;
```

3. Add the following code under the comment **Create a kernel with Azure OpenAI chat completion**:

Python

```
# Create a kernel with Azure OpenAI chat completion
kernel = Kernel()
chat_completion = AzureChatCompletion(
    api_key=api_key,
    endpoint=endpoint,
    deployment_name=deployment_name
)
kernel.add_service(chat_completion)
```

C#

```
// Create a kernel with Azure OpenAI chat completion
var builder =
Kernel.CreateBuilder().AddAzureOpenAIChatCompletion(deploymentName,
endpoint, apiKey);
Kernel kernel = builder.Build();
```

4. To test that your kernel and endpoint is working, enter the following code under the comment **Test the chat completion service**:

Python

```
# Test the chat completion service
response = await kernel.invoke_prompt(
    "Give me a list of 10 breakfast foods with eggs and cheese",
    KernelArguments())
print("Assistant > " + str(response))
```

C#

```
// Test the chat completion service
var result = await kernel.InvokePromptAsync("Give me a list of 10 breakfast
foods with eggs and cheese");
Console.WriteLine(result);
```

5. After you've updated the code, use the **CTRL+S** command to save your changes.

6. Use the **CTRL+Q** command to close the code editor.

7. Run the code using the following command:

Python

```
python kernel.py
```

C#

```
dotnet run
```

8. Check that you see a response similar to the following:

```
Certainly! Here's a list of popular breakfast foods that incorporate eggs
and cheese:

1. Omelette
2. Frittata
3. Breakfast burrito
4. Scrambled eggs with cheese
5. Quiche
6. Huevos rancheros
7. Cheese and egg sandwich
8. Egg and cheese bagel
9. Egg and cheese croissant
10. Baked eggs with cheese
```

> The response comes from the Azure OpenAI model you passed to the kernel. The Semantic Kernel SDK is able to connect to the large language model (LLM) and run the prompt. Notice how quickly you were able to receive responses from the LLM. The Semantic Kernel SDK makes building smart applications easy and efficient.

Summary

In this lab, you created an endpoint for the large language model (LLM) service, built a Semantic Kernel object, and ran a prompt using the Semantic Kernel SDK. Congratulations on completing this exercise!

Clean up

If you've finished exploring Azure OpenAI and Semantic Kernel, 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.