

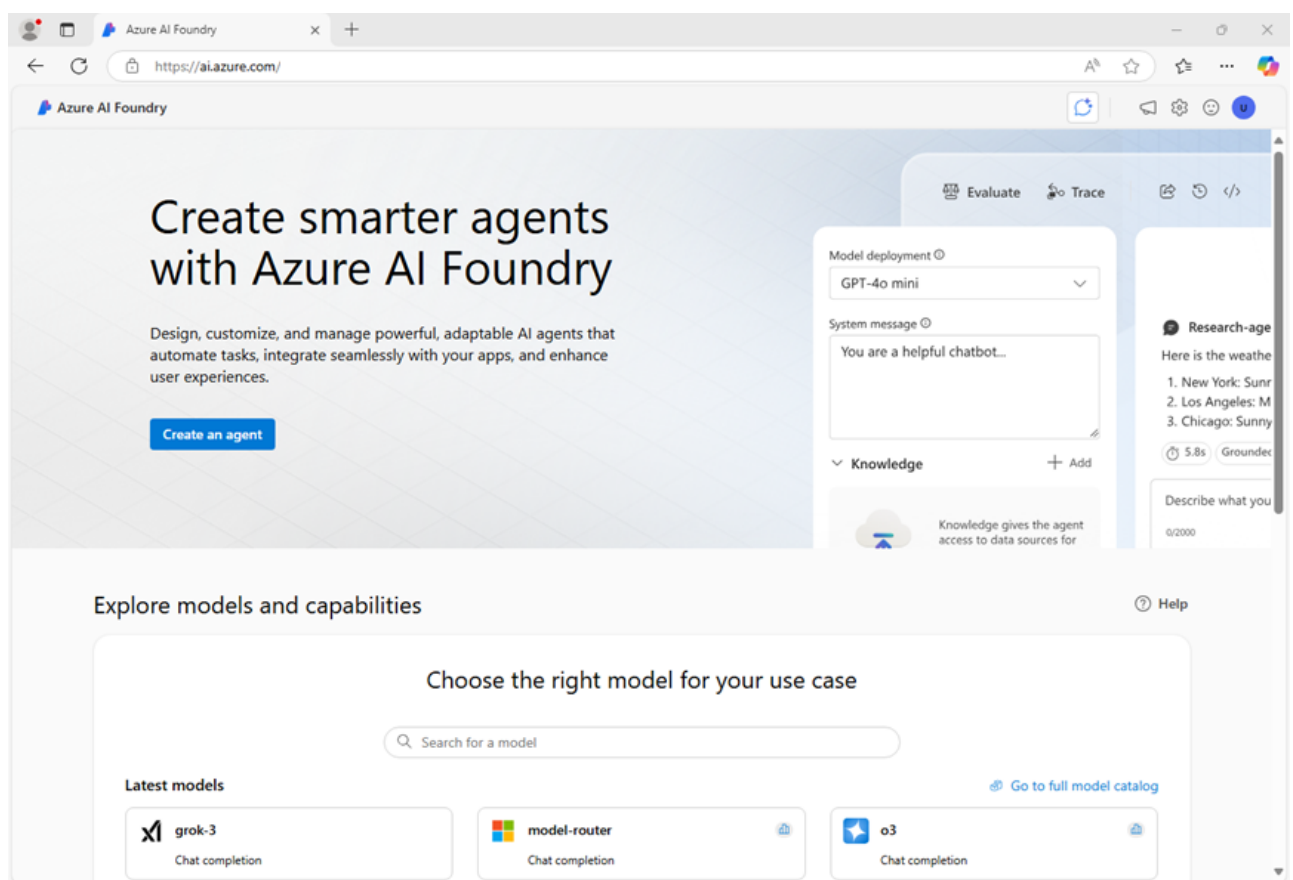
# Create Semantic Kernel plugins

In this exercise, you'll use Semantic Kernel to create an AI assistant that can search for and book flights for a user. You'll create custom plugin functions to help accomplish the task. Let's get started!

This exercise takes approximately **15** minutes to complete.

## Deploy a chat completion model

1. In a web browser, open the [Azure AI Foundry portal](https://ai.azure.com) 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):



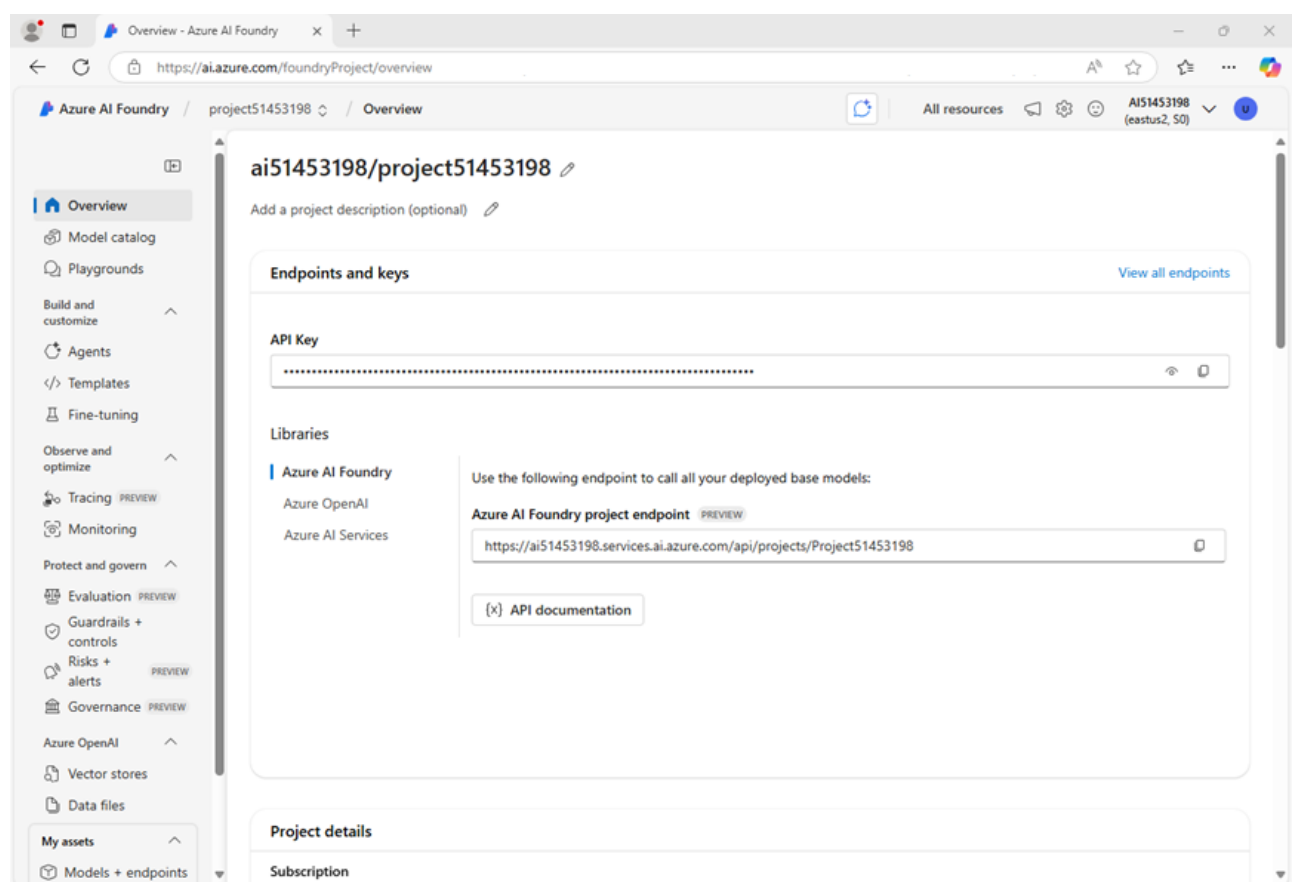
2. In the home page, in the **Explore models and capabilities** section, search for the **gpt-4o** model; which we'll use in our project.
3. In the search results, select the **gpt-4o** model to see its details, and then at the top of the page for the model, select **Use this model**.
4. When prompted 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 an AI client app

Now that you deployed a model, you're ready to create a Semantic Kernel client app that defines custom plugin functions. Some code is 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](https://portal.azure.com) 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/03-create-plugins/Python
```

### C#

```
cd mslearn-ai-semantic-kernel/Labfiles/03-create-plugins/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.

Now you're ready to begin the exercise. Good luck!

## Create a flight booking plugin

Now you create a plugin class for your travel assistant. The class includes functions to search for available flights and book a selected flight for the user. A **flights.json** file is included in the project which contains some sample flight information.

**Tip:** As you add code, be sure to maintain the correct indentation.

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

### Python

```
code flight_booking_plugin.py
```

## C#

```
code FlightBookingPlugin.cs
```

2. Add the following code under the comment **Create a plugin function with kernel function attributes**:

## Python

```
# Create a plugin function with kernel function attributes
@kernel_function(description="Searches for available flights based on the
destination and departure date in the format YYYY-MM-DD")
def search_flights(self, destination, departure_date):
    # Filter flights based on destination
```

## C#

```
// Create a plugin function with kernel function attributes
[KernelFunction("search_flights")]
[Description("Searches for available flights based on the destination and
departure date in the format YYYY-MM-DD")]
[return: Description("A list of available flights")]
public List<FlightModel> SearchFlights(string destination, string
departureDate)
{
    // Filter flights based on destination
}
```

The kernel function decorators help the AI understand how to call your function.

3. Add the following code under the comment **Filter flights based on destination**

## Python

```
# Filter flights based on destination
matching_flights = [
    flight for flight in self.flights
    if flight.Destination.lower() == destination.lower() and
    flight.DepartureDate == departure_date
]
return matching_flights
```

**C#**

```
// Filter flights based on destination
return flights.Where(flight =>
    flight.Destination.Equals(destination,
StringComparison.OrdinalIgnoreCase) &&
    flight.DepartureDate.Equals(departureDate)).ToList();
```

4. Add the following code under the comment **Create a kernel function to book flights**:

**Python**

```
# Create a kernel function to book flights
@kernel_function(description="Books a flight based on the flight ID
provided")
def book_flight(self, flight_id):
    # Add logic to book a flight
```

**C#**

```
// Create a kernel function to book flights
[KernelFunction("book_flight")]
[Description("Books a flight based on the flight ID provided")]
[return: Description("Booking confirmation message")]
public string BookFlight(int flightId)
{
    // Add logic to book a flight
}
```

5. Add the following code under the comment **Add logic to book a flight**:

**Python**

```
# Add logic to book a flight
flight = next((f for f in self.flights if f.Id == flight_id), None)

if flight is None:
    return "Flight not found. Please provide a valid flight ID."

if flight.IsBooked:
    return "You've already booked this flight."

flight.IsBooked = True
self.save_flights_to_file()

return (
```

```

        f"Flight booked successfully! Airline: {flight.Airline}, "
        f"Destination: {flight.Destination}, Departure: {flight.DepartureDate},
    "
        f"Price: ${flight.Price}."
    )

```

## C#

```

// Add logic to book a flight
var flight = flights.FirstOrDefault(f => f.Id == flightId);
if (flight == null)
{
    return "Flight not found. Please provide a valid flight ID.";
}

if (flight.IsBooked)
{
    return $"You've already booked this flight.";
}

flight.IsBooked = true;
SaveFlightsToFile();

return @$"Flight booked successfully! Airline: {flight.Airline},
        Destination: {flight.Destination},
        Departure: {flight.DepartureDate},
        Price: ${flight.Price}.";

```

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

## Python

```
code plugins.py
```

## C#

```
code Program.cs
```

7. Add the following code under the comment **Add a plugin to the kernel**:

## Python

```

# Add the plugin to the kernel
kernel.add_plugin(FlightBookingPlugin(), "flight_booking_plugin")

```

**C#**

```
// Add a plugin to the kernel*  
kernel.Plugins.AddFromType<FlightBookingPlugin>("FlightBookingPlugin");
```

8. Add the following code under the comment **Configure function choice behavior**

**Python**

```
# Configure function choice behavior  
settings = AzureChatPromptExecutionSettings(  
    function_choice_behavior=FunctionChoiceBehavior.Auto(),  
)
```

**C#**

```
// Configure function choice behavior  
OpenAIPromptExecutionSettings openAIPromptExecutionSettings = new()  
{  
    FunctionChoiceBehavior = FunctionChoiceBehavior.Auto()  
};
```

9. After you've updated the code, use the **CTRL+S** command to save your changes.
10. In the cloud shell command-line pane beneath the code editor, enter the following command to run the code:

**Python**

```
python plugins.py
```

**C#**

```
dotnet run
```

You should see output similar to the following:

```
User: Find me a flight to Tokyo on the 19  
Assistant: I found a flight to Tokyo on January 19th.  
  
- Airline: Air Japan  
- Price: $1200
```



```
Would you like to book this flight?
```

11. Enter "Yes" to invoke the `BookFlight` function

You should see output similar to the following:

```
User: Yes
Assistant: Congratulations! Your flight to Tokyo on January 19th with Air
Japan has been successfully booked.
The total price for the flight is $1200.
```

Now you successfully added plugin functions to your kernel that can be invoked automatically. Great work!

## Advertise selected functions

Now you configure your kernel so that only selected functions are available to assist in providing travel information. The assistant should be able to answer questions about flight availability, but not allow the user to book a flight.

1. Update the code under the comment **Configure function choice behavior** to the following code:

### Python

```
# Configure function choice behavior
settings=AzureChatPromptExecutionSettings(
    function_choice_behavior=FunctionChoiceBehavior.Auto(filters=
{"included_functions": ["search_flights"]}),
)
```

### C#

```
// Configure function choice behavior
KernelFunction searchFlights =
kernel.Plugins.GetFunction("FlightBookingPlugin", "search_flights");

PromptExecutionSettings openAIPromptExecutionSettings = new()
{
    FunctionChoiceBehavior = FunctionChoiceBehavior.Required(functions:
[searchFlights])
};
```

2. After you've updated the code, use the **CTRL+S** command to save your changes.
3. Run the code as is to observe the difference in behavior.

If you request to book the flight, you should see output similar to the following:

```
User: Find me a flight to Tokyo on the 19
Assistant: I found a flight to Tokyo on the 19th of January 2025. The flight
is with Air Japan and the price is $1200. Please let me know if you would
like to book this flight.
User: Yes
Assistant: I'm sorry, but I am just a virtual assistant and I don't have the
capability to book flights.
```

## Summary

In this exercise, you used the Semantic Kernel SDK and Azure OpenAI to create an AI assistant that can help users search for and book flights. Great work!

## 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](https://portal.azure.com) 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.