Contents

Solution Guide: Intelligent App Development with Copilot Stack Scenario

Step 1: Deploy Azure OpenAI Service and LLM Models

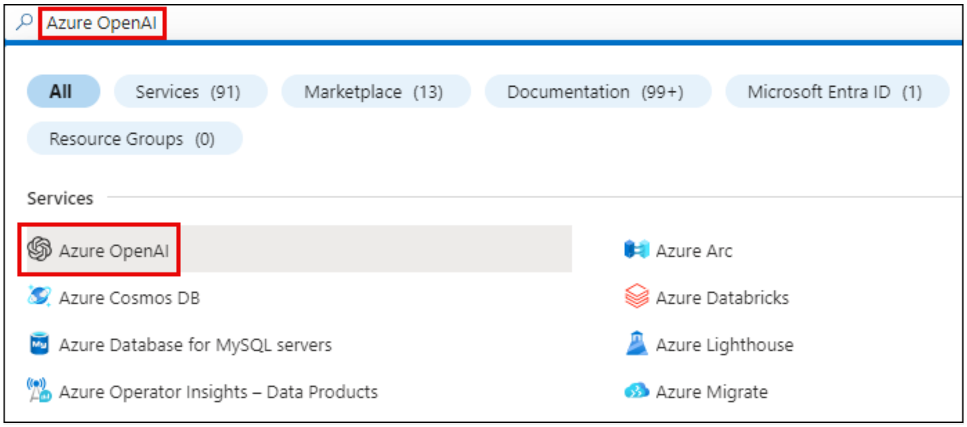
Notes & Guidance

Task 01 is all about helping the student set up the prerequisites for this Step. This includes necessary installations, environment options, and other libraries needed.

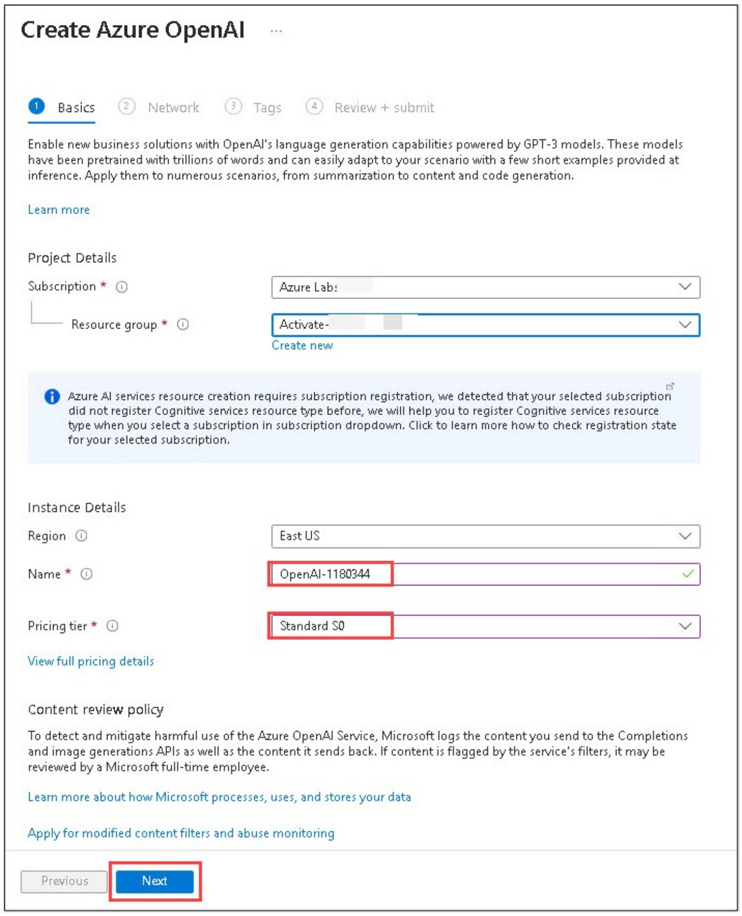
Task 1: Deploy an Azure OpenAI Service

In this task, you'll be deploying an Azure OpenAI service in the Azure Portal.

1. In the Azure Portal, search for **Azure OpenAI** and select it.



2. On **Azure AI Services | Azure OpenAI** blade, click on **create** enter the details required and deploy the Azure Open AI service.



Create Azure OpenAI ...

① Basics ② Network ③ Tags ④ Review + submit

Enable new business solutions with OpenAI's language generation capabilities powered by GPT-3 models. These models have been pretrained with trillions of words and can easily adapt to your scenario with a few short examples provided at inference. Apply them to numerous scenarios, from summarization to content and code generation.

Learn more

Project Details

Subscription * ⓘ Azure Labs [redacted]

Resource group * ⓘ Activate-[redacted]

Create new

ⓘ Azure AI services resource creation requires subscription registration, we detected that your selected subscription did not register Cognitive services resource type before, we will help you to register Cognitive services resource type when you select a subscription in subscription dropdown. Click to learn more how to check registration state for your selected subscription.

Instance Details

Region ⓘ East US

Name * ⓘ OpenAI-1180344 ✓

Pricing tier * ⓘ Standard S0

View full pricing details

Content review policy

To detect and mitigate harmful use of the Azure OpenAI Service, Microsoft logs the content you send to the Completions and image generations APIs as well as the content it sends back. If content is flagged by the service's filters, it may be reviewed by a Microsoft full-time employee.

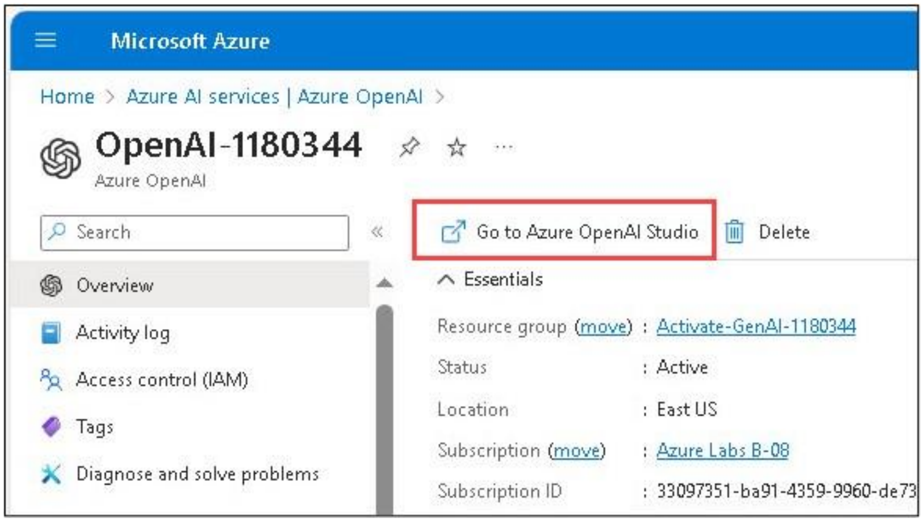
Learn more about how Microsoft processes, uses, and stores your data

Apply for modified content filters and abuse monitoring

Task 2: Deploy LLM models in Azure OpenAI Studio

In this task, you'll be deploying the OpenAI models from Azure OpenAI Studio.

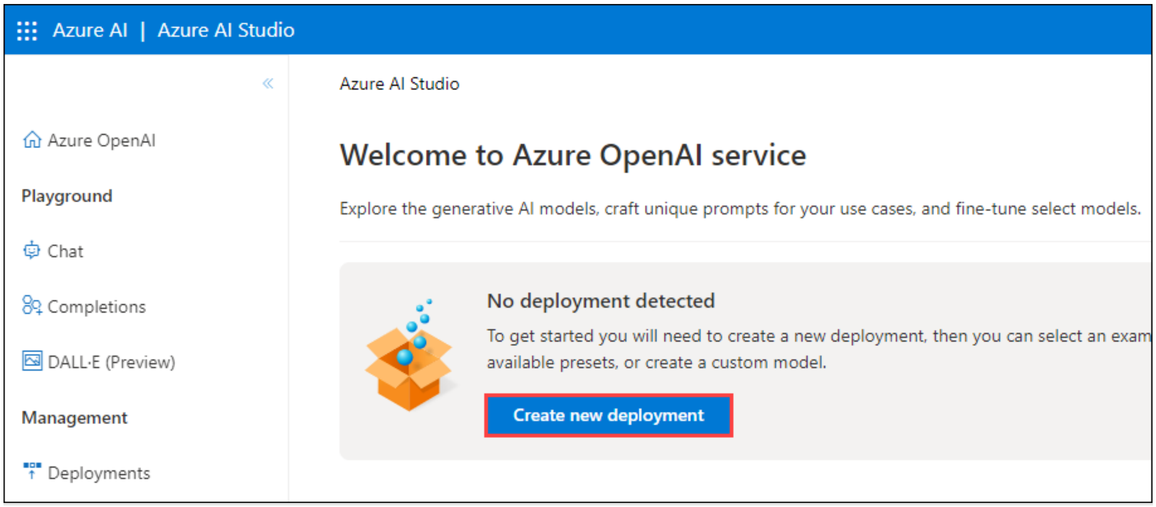
1. In the Azure OpenAI resource pane, click on **Go to Azure OpenAI Studio;** it will navigate to **Azure AI Studio**.


Microsoft Azure
Home > Azure AI services | Azure OpenAI >
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Go to Azure OpenAI Studio
Delete
Essentials

OpenAI-1180344
Azure OpenAI

Resource group (move) : Activate-GenAI-1180344
Status : Active
Location : East US
Subscription (move) : Azure Labs B-08
Subscription ID : 33097351-ba91-4359-9960-de73

2. On **the Welcome to Azure OpenAI Service** page, click on **Create new deployment**.



Welcome to Azure OpenAI service

Explore the generative AI models, craft unique prompts for your use cases, and fine-tune select models.

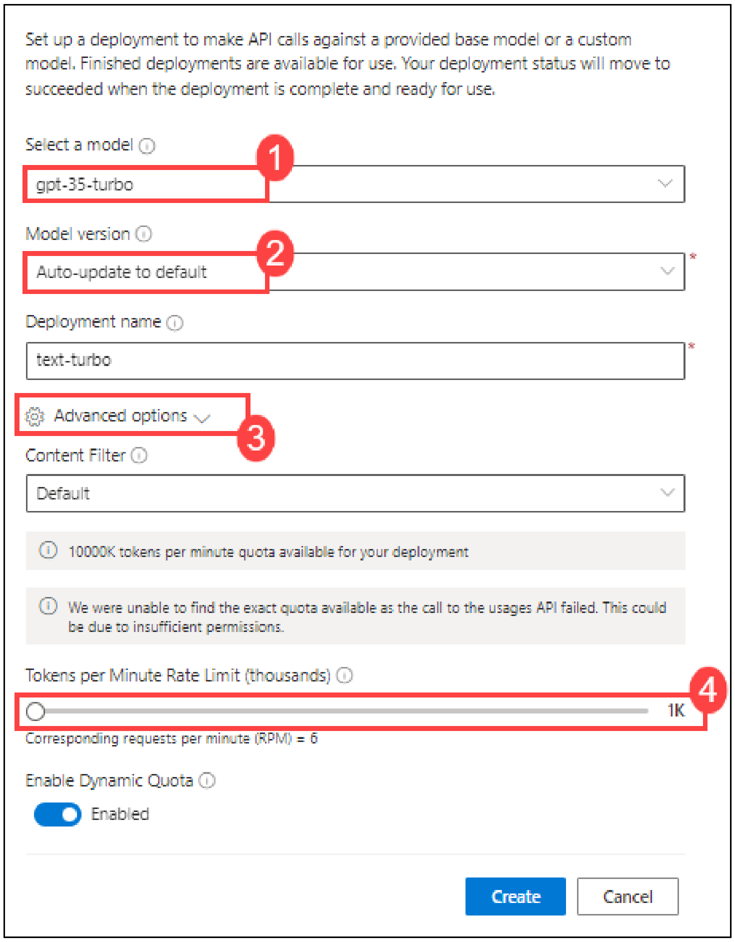
No deployment detected

To get started you will need to create a new deployment, then you can select an exam
available presets, or create a custom model.

3. In the **Deployments** page, click on **+ Create new deployment**.

4. Within the **Deploy model** pop-up interface, enter the following details and then click on **Advanced options (3),** followed by scaling down the **Tokens per Minute Rate Limit (thousands) (4)**:

   - **Select a model**: gpt-35-turbo (1)
   - **Model version**: *Use the default version* (2)
   - **Deployment name**: gpt-35-turbo
   - **Tokens per Minute Rate Limit (thousands)**: 20K



Set up a deployment to make API calls against a provided base model or a custom model. Finished deployments are available for use. Your deployment status will move to succeeded when the deployment is complete and ready for use.

Select a model: gpt-35-turbo

Model version: Auto-update to default *

Deployment name: text-turbo *

Advanced options

Content Filter: Default

10000K tokens per minute quota available for your deployment

We were unable to find the exact quota available as the call to the usages API failed. This could be due to insufficient permissions.

Tokens per Minute Rate Limit (thousands): 1K

Corresponding requests per minute (RPM) = 6

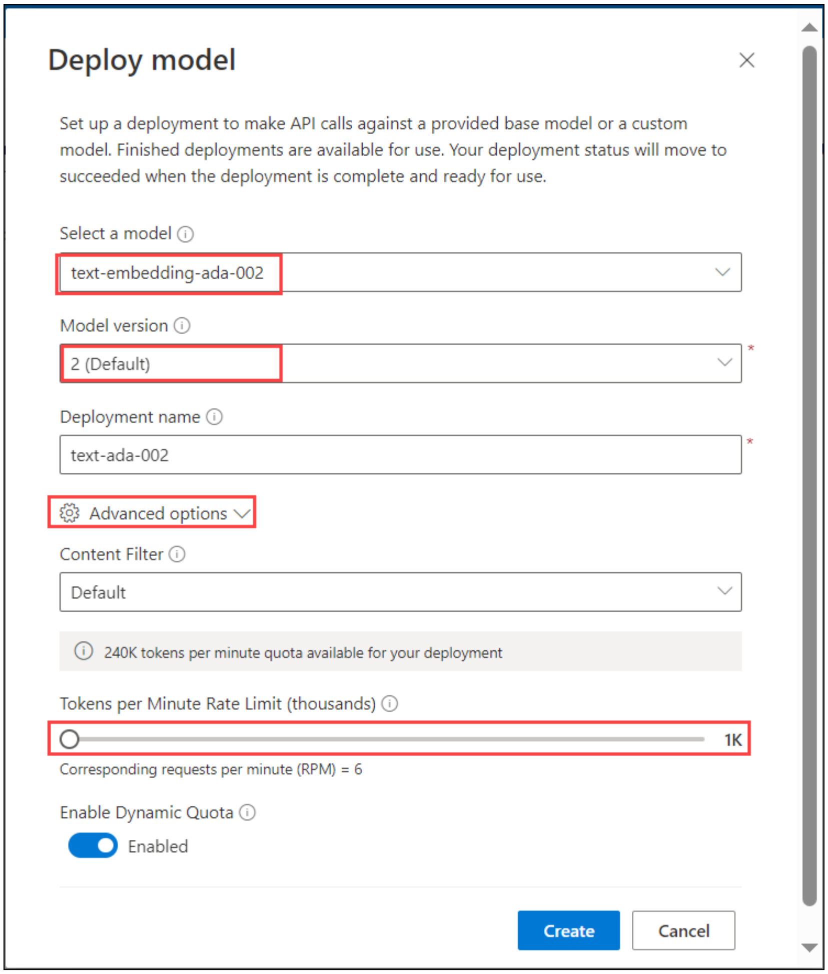
Enable Dynamic Quota: Enabled

5. Click on the **Create** button to deploy a model that you will be playing around with as you proceed.

6. In the **Deployments** page again, click on **+ Create new deployment**.

7. Within the **Deploy model** pop-up interface, enter the following details and then click on **Advanced options (3),** followed by scaling down the **Tokens per Minute Rate Limit (thousands) (4)**:

   - **Select a model**: text-embedding-ada-002 (1)
   - **Model version**: *Use the default version* (2)
   - **Deployment name**: text-embedding-ada-002
   - **Tokens per Minute Rate Limit (thousands)**: 20K



Deploy model

Set up a deployment to make API calls against a provided base model or a custom model. Finished deployments are available for use. Your deployment status will move to succeeded when the deployment is complete and ready for use.

Select a model

text-embedding-ada-002

Model version

2 (Default)

Deployment name

text-ada-002

Content Filter

Default

240K tokens per minute quota available for your deployment

Tokens per Minute Rate Limit (thousands)

1K

Corresponding requests per minute (RPM) = 6

Enable Dynamic Quota

Enabled

8. Click on the **Create** button to deploy a model that you will be playing around with as you proceed.

Step 2: Explore Semantic Kernel – Solution Guide

Task 1: Clone the repository for this course

If you have not already cloned the **Semantic Kernel GitHub Repo** code repository to the environment where you're working on this lab, follow these steps to do so. Otherwise, open the cloned folder in Visual Studio Code.

1. Open Visual Studio Code.

2. Create and Navigate to the directory `C:/Users/azureuser`.

```
cd C:/Users/azureuser
```

3. Clone the given repository.

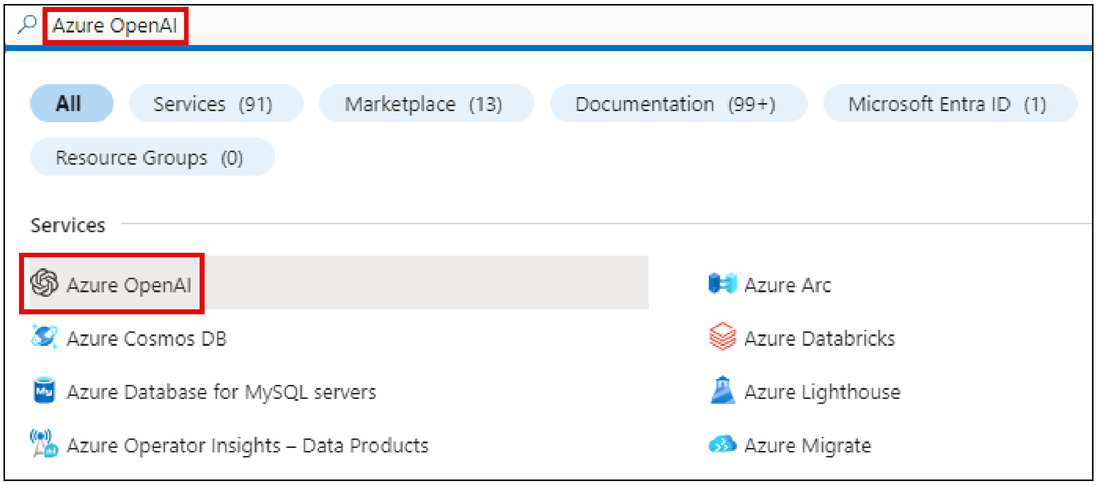
```
git clone https://github.com/microsoft/semantic-kernel
```

4. When the repository has been cloned, open the folder in Visual Studio Code.

Task 2: Retrieving the Azure OpenAI Service values

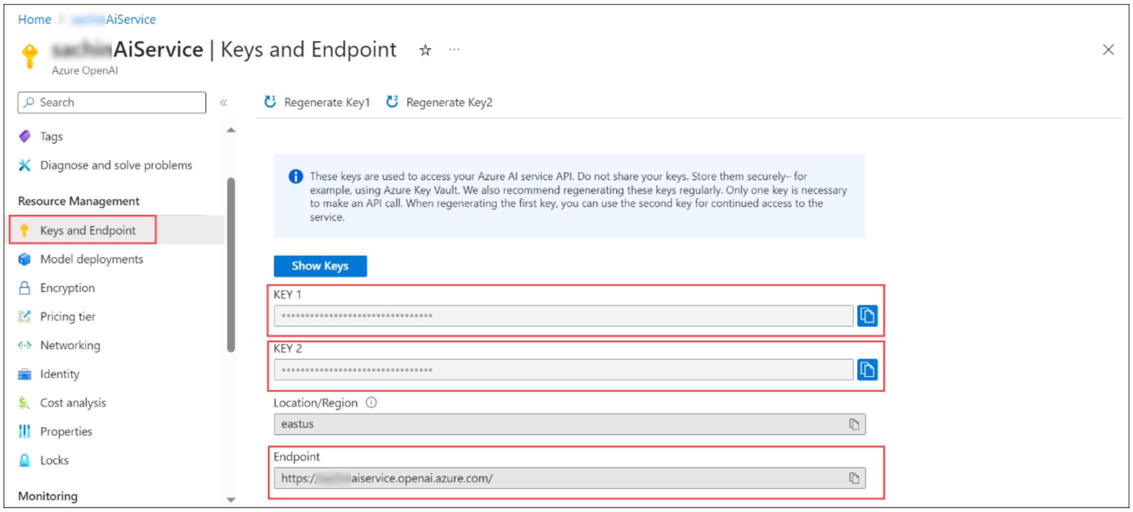
From the Azure portal, you need to retrieve the Azure OpenAI Service Key, Endpoint, and LLM model name deployed in the previous Step.

1. In the Azure Portal, search for **Azure OpenAI** and select it.



2. Go to the Azure OpenAI resource that you created previously and choose **Keys and Endpoints** from the left pane.

3. Copy any of the 2 Keys and the Endpoint and store it somewhere for future reference.



These keys are used to access your Azure AI service API. Do not share your keys. Store them securely– for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

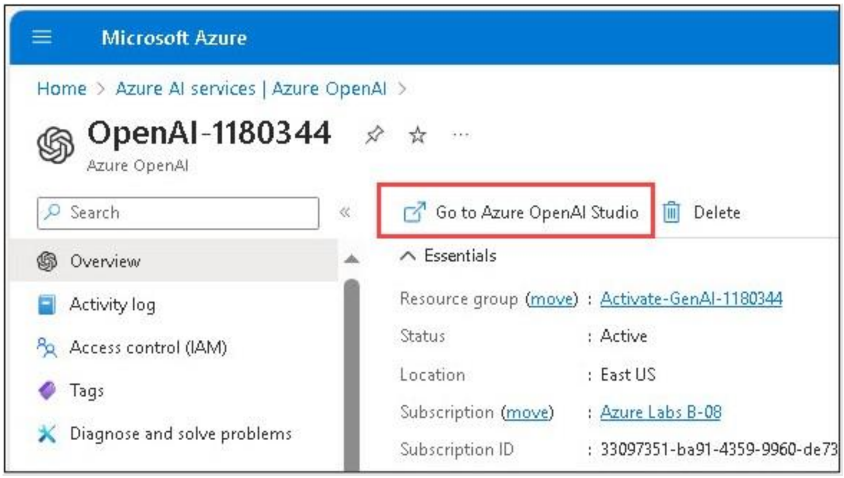
KEY 1
••••••••••••••••••••••••••••••••

KEY 2
••••••••••••••••••••••••••••••••

Location/Region
eastus

Endpoint
https://[redacted]aiservice.openai.azure.com/

4. From the Overview page, click **Go to Azure OpenAI Studio** to go to your deployed models.

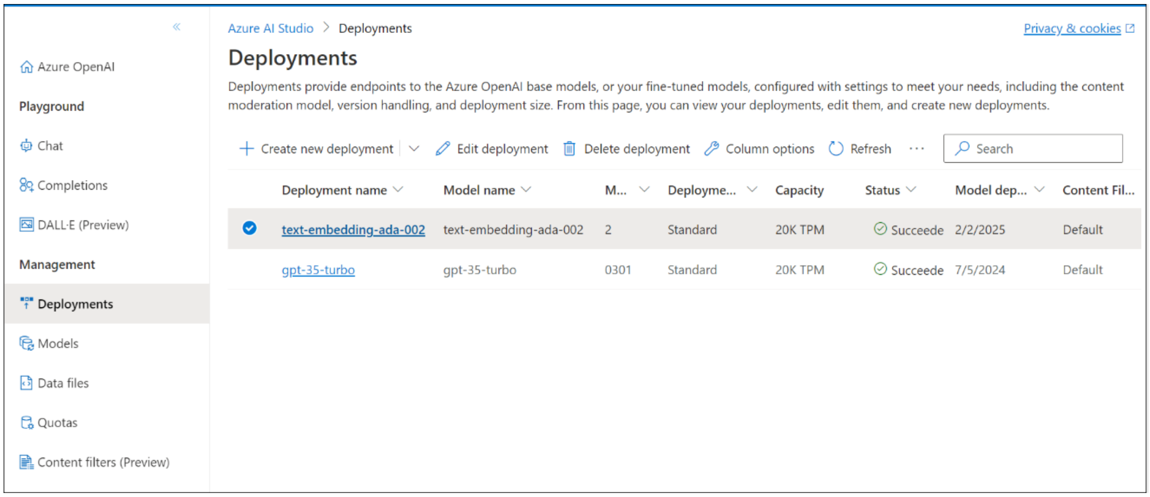

Microsoft Azure
Home > Azure AI services | Azure OpenAI >
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Go to Azure OpenAI Studio
Delete
Essentials

OpenAI-1180344

Azure OpenAI

Resource group (move) : Activate-GenAI-1180344
Status : Active
Location : East US
Subscription (move) : Azure Labs B-08
Subscription ID : 33097351-ba91-4359-9960-de73

5. Go to **Deployments,** and you will see your deployed AI models. Copy the deployment names of your AI model that you deployed previously and store them somewhere for future reference.



Deployments

Deployments provide endpoints to the Azure OpenAI base models, or your fine-tuned models, configured with settings to meet your needs, including the content moderation model, version handling, and deployment size. From this page, you can view your deployments, edit them, and create new deployments.

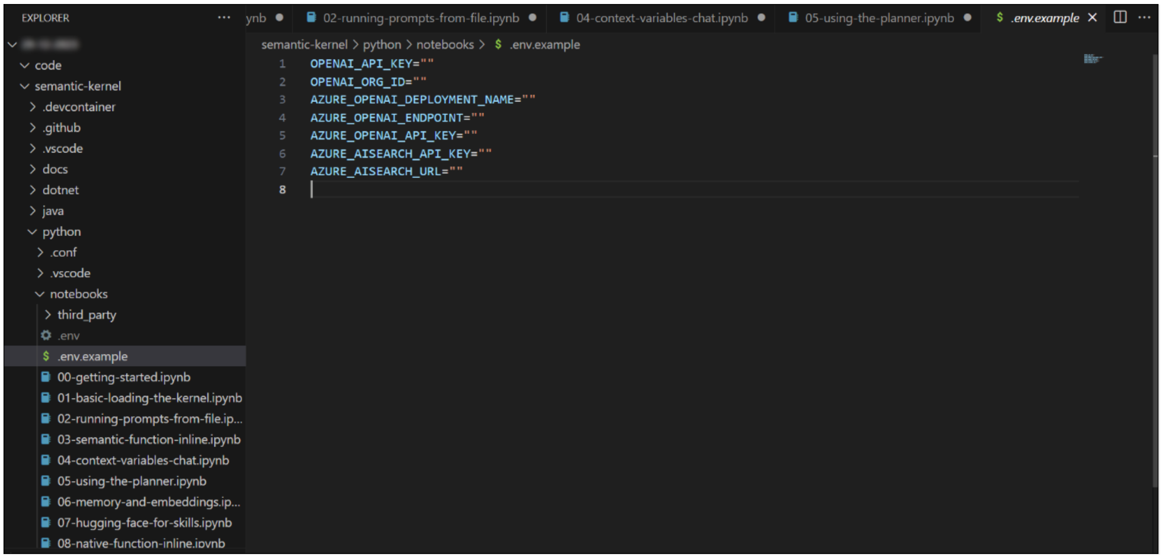
Deployment name	Model name	M...	Deployme...	Capacity	Status	Model dep...	Content Fil...
text-embedding-ada-002	text-embedding-ada-002	2	Standard	20K TPM	✓ Succeede	2/2/2025	Default
gpt-35-turbo	gpt-35-turbo	0301	Standard	20K TPM	✓ Succeede	7/5/2024	Default

Task 3: Run Jupyter Notebooks to get started with Semantic Kernel in the Python programming language

Inside VS Code, you need to run the required notebooks cell-by-cell successfully and observe the outputs for better understanding.

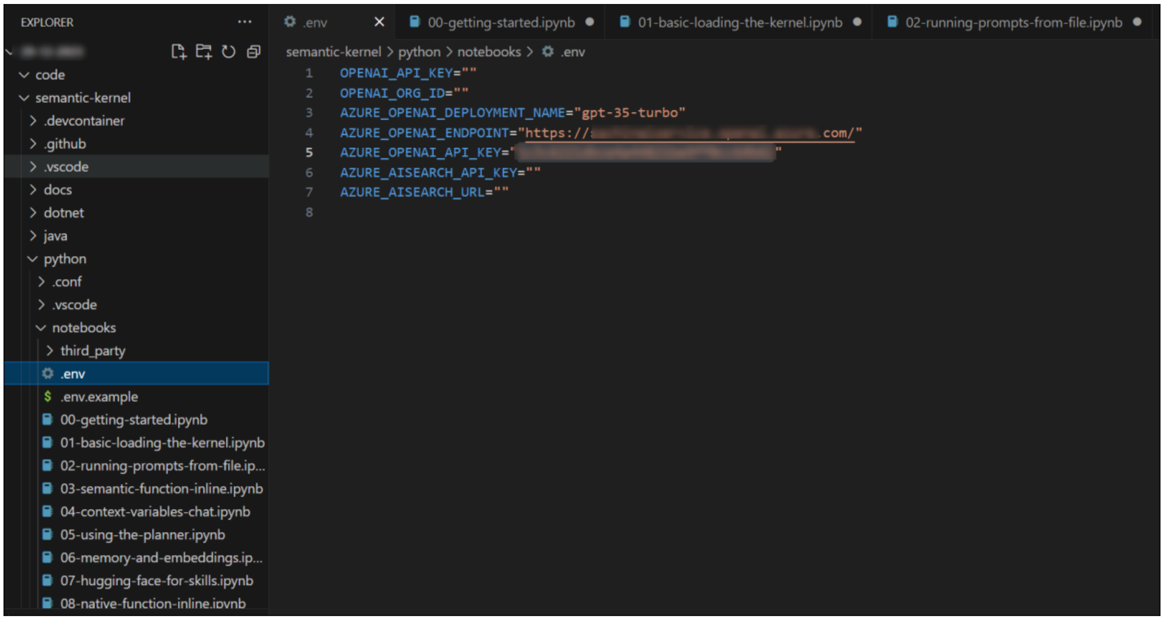
1. Go to the cloned folder in your **VS Code**, navigate to **semantic-kernel/python/samples/getting_starded,** and create a new **.env** file.

2. Go to **.env.example** , copy all its contents, and paste them into the newly created **.env** file.


EXPLORER
[redacted]
code
semantic-kernel
.devcontainer
.github
.vscode
docs
dotnet
java
python
.conf
.vscode
notebooks
third_party
.env
.env.example
00-getting-started.ipynb
01-basic-loading-the-kernel.ipynb
02-running-prompts-from-file.ip...
03-semantic-function-inline.ipynb
04-context-variables-chat.ipynb
05-using-the-planner.ipynb
06-memory-and-embeddings.ip...
07-hugging-face-for-skills.ipynb
08-native-function-inline.ipynb
ynb
02-running-prompts-from-file.ipynb
04-context-variables-chat.ipynb
05-using-the-planner.ipynb
.env.example
semantic-kernel > python > notebooks > $ .env.example

```
OPENAI_API_KEY=""
OPENAI_ORG_ID=""
AZURE_OPENAI_DEPLOYMENT_NAME=""
AZURE_OPENAI_ENDPOINT=""
AZURE_OPENAI_API_KEY=""
AZURE_AISEARCH_API_KEY=""
AZURE_AISEARCH_URL=""

```

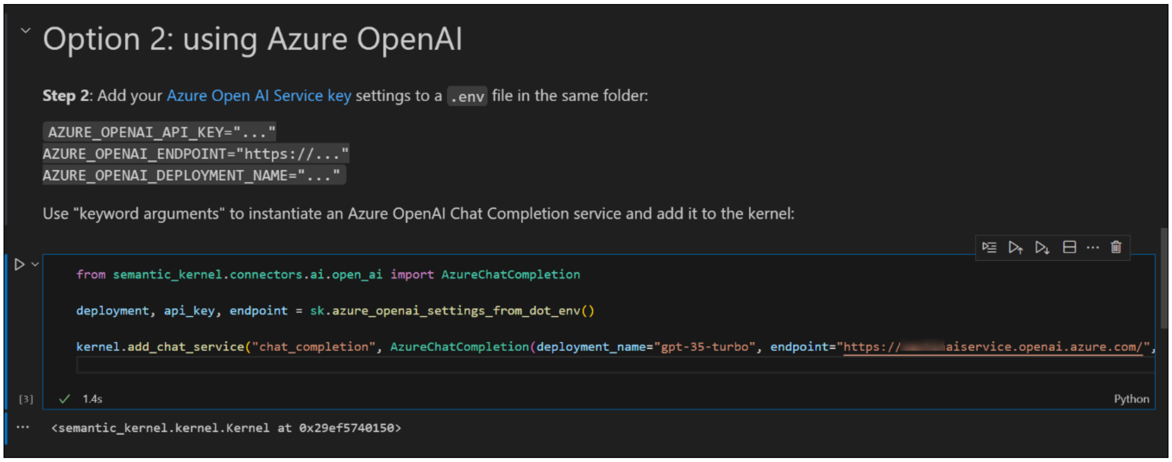
3. Modify the **.env** file by updating the **Azure OpenAI Model name, Endpoint, and Key** that you have retrieved in the earlier steps.


EXPLORER
[redacted]
code
semantic-kernel
.devcontainer
.github
.vscode
docs
dotnet
java
python
.conf
.vscode
notebooks
third_party
.env
.env.example
00-getting-started.ipynb
01-basic-loading-the-kernel.ipynb
02-running-prompts-from-file.ip...
03-semantic-function-inline.ipynb
04-context-variables-chat.ipynb
05-using-the-planner.ipynb
06-memory-and-embeddings.ip...
07-hugging-face-for-skills.ipynb
08-native-function-inline.ipynb
.env
00-getting-started.ipynb
01-basic-loading-the-kernel.ipynb
02-running-prompts-from-file.ipynb
semantic-kernel > python > notebooks > .env

```
OPENAI_API_KEY=""
OPENAI_ORG_ID=""
AZURE_OPENAI_DEPLOYMENT_NAME="gpt-35-turbo"
AZURE_OPENAI_ENDPOINT="https://[redacted]com/"
AZURE_OPENAI_API_KEY="[redacted]"
AZURE_AISEARCH_API_KEY=""
AZURE_AISEARCH_URL=""

```

4. Navigate to `00-getting-started.ipynb` and run the notebook cell-by-cell, where you will be importing Semantic Kernel SDK.
Note: Skip the cell of codes pertaining to OpenAI, i.e., **Option 1** in the notebook, and only run the cell of codes related to Azure **OpenAI,** i.e., starting from **Option 2** in the notebook.

5. Update the values of **deployment_name, endpoint, and api_key** in **Option 2**, which you have already retrieved, and run the function.



Option 2: using Azure OpenAI

Step 2: Add your Azure Open AI Service key settings to a `.env` file in the same folder:

```
AZURE_OPENAI_API_KEY="..."
AZURE_OPENAI_ENDPOINT="https://..."
AZURE_OPENAI_DEPLOYMENT_NAME="..."
```

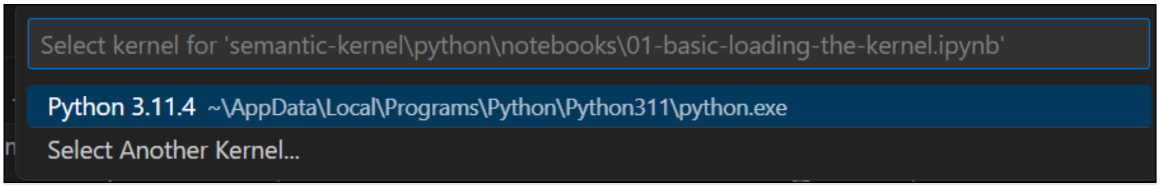
Use "keyword arguments" to instantiate an Azure OpenAI Chat Completion service and add it to the kernel:

```
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion

deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()

kernel.add_chat_service("chat_completion", AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https://[redacted]aiservice.openai.azure.com/",
```

```
<semantic_kernel.kernel.Kernel at 0x29ef5740150>
```

Run the remaining code as usual.

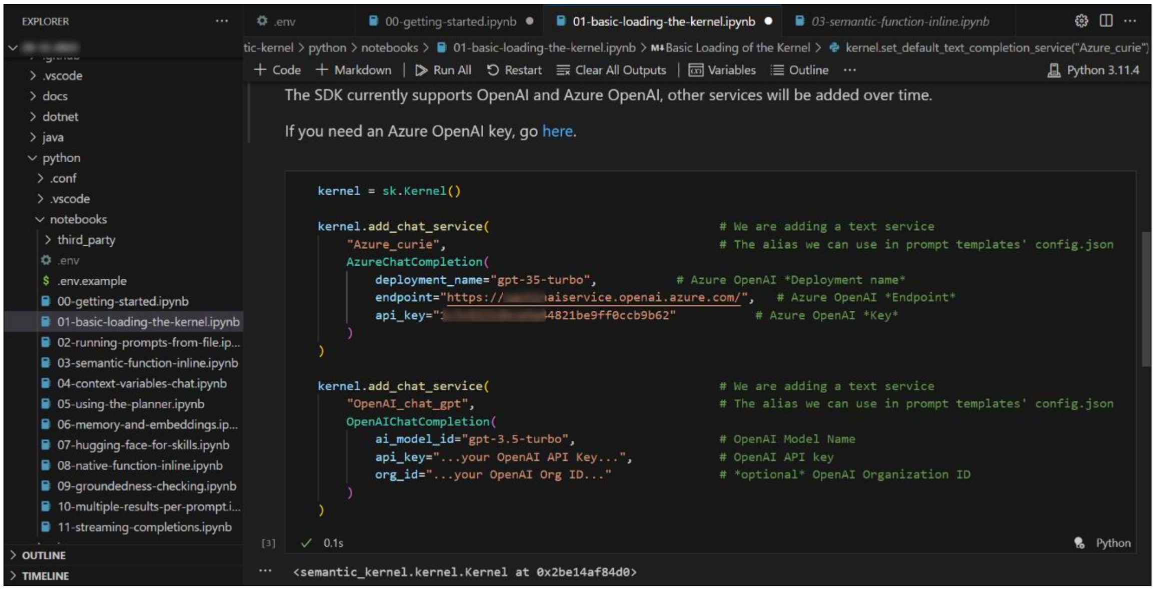
6. After successfully completing the previous notebook, navigate to `01-basic-loading-the-kernel.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen.



Select kernel for 'semantic-kernel\python\notebooks\01-basic-loading-the-kernel.ipynb'

Python 3.11.4 ~\AppData\Local\Programs\Python\Python311\python.exe
Select Another Kernel...

7. Update the values of **deployment_name, endpoint, and api_key,** which you noted earlier, in the following cell of code before executing it.



EXPLORER
[redacted]
.vscode
docs
dotnet
java
python
.conf
.vscode
notebooks
third_party
.env
.env.example
00-getting-started.ipynb
01-basic-loading-the-kernel.ipynb
02-running-prompts-from-file.ip...
03-semantic-function-inline.ipynb
04-context-variables-chat.ipynb
05-using-the-planner.ipynb
06-memory-and-embeddings.ip...
07-hugging-face-for-skills.ipynb
08-native-function-inline.ipynb
09-groundedness-checking.ipynb
10-multiple-results-per-prompt.i...
11-streaming-completions.ipynb
OUTLINE
TIMELINE
.env
00-getting-started.ipynb
01-basic-loading-the-kernel.ipynb
03-semantic-function-inline.ipynb
tic-kernel > python > notebooks > 01-basic-loading-the-kernel.ipynb > Basic Loading of the Kernel > kernel.set_default_text_completion_service("Azure_curie")
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···
Python 3.11.4

The SDK currently supports OpenAI and Azure OpenAI, other services will be added over time.

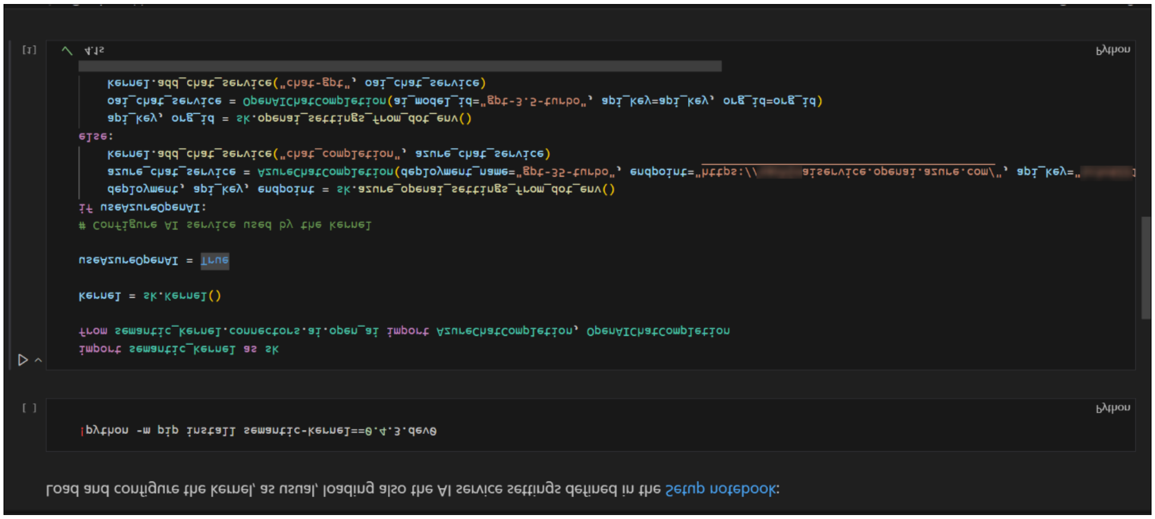
If you need an Azure OpenAI key, go here.

```
kernel = sk.Kernel()

kernel.add_chat_service(                                   # We are adding a text service
    "Azure_curie",                                         # The alias we can use in prompt templates' config.json
    AzureChatCompletion(
        deployment_name="gpt-35-turbo",             # Azure OpenAI *Deployment name*
        endpoint="https://[redacted]aiservice.openai.azure.com/",    # Azure OpenAI *Endpoint*
        api_key="[redacted]4821be9ff0ccb9b62"                # Azure OpenAI *Key*
    )
)

kernel.add_chat_service(                                   # We are adding a text service
    "OpenAI_chat_gpt",                                     # The alias we can use in prompt templates' config.json
    OpenAIChatCompletion(
        ai_model_id="gpt-3.5-turbo",                       # OpenAI Model Name
        api_key="...your OpenAI API Key...",               # OpenAI API key
        org_id="...your OpenAI Org ID..."                  # *optional* OpenAI Organization ID
    )
)
```

```
<semantic_kernel.kernel.Kernel at 0x2be14af84d0>
```

8. After successfully running the previous notebook, navigate to `02-running-prompts-from-file.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen.

9. In the following function, modify **useAzureOpenAI** to **True**, set the values of **deployment_name, endpoint, and api_key,** which you noted earlier, and execute it.



Load and configure the kernel, as usual, loading also the AI service settings defined in the Setup notebook:

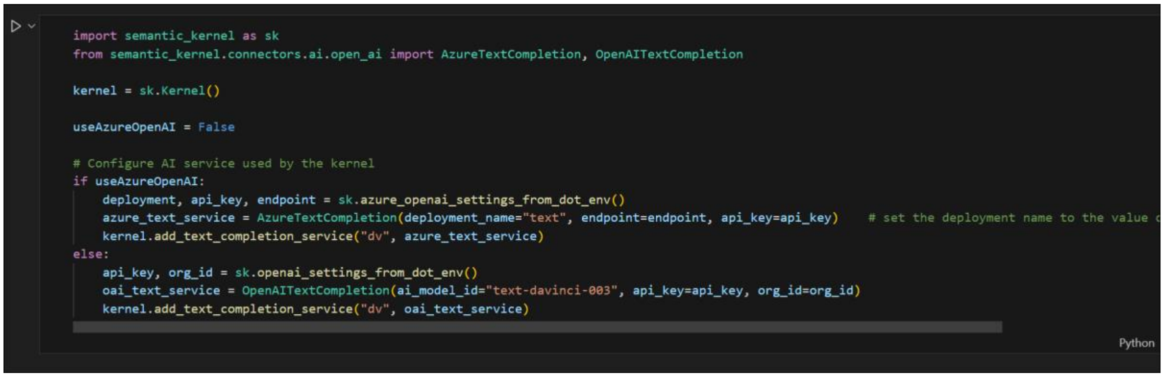
```
[ ]
!python -m pip install semantic-kernel==0.4.3.dev0
```

```
[1]
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion, OpenAIChatCompletion

kernel = sk.Kernel()

useAzureOpenAI = True

# Configure AI service used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    azure_chat_service = AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https://[redacted]aiservice.openai.azure.com/", api_key="[redacted]
    kernel.add_chat_service("chat_completion", azure_chat_service)
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    oai_chat_service = OpenAIChatCompletion(ai_model_id="gpt-3.5-turbo", api_key=api_key, org_id=org_id)
    kernel.add_chat_service("chat-gpt", oai_chat_service)
```

Run the remaining cells of code as usual.

10. After successfully completing the previous notebook, navigate to `03-semantic-function-inline.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen. Here, you need an OpenAI model that supports text completion. For that, you need to navigate to Azure OpenAI Studio and deploy a `text-davinci-003` model with a TPM capacity of 10K.

11. In the following function, set the values of **deployment_name, endpoint, and api_key** for your **text-davinci-003** model, which you noted earlier, and execute it.

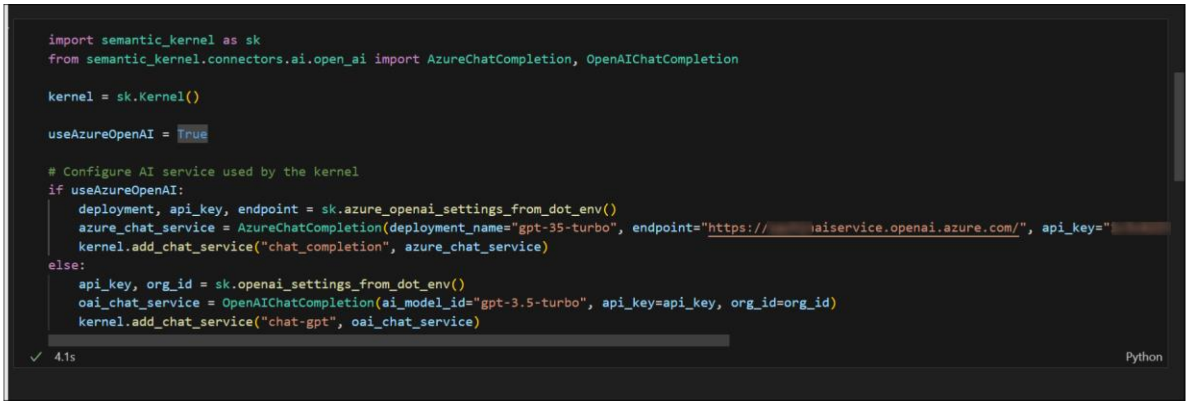


```python
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import AzureTextCompletion, OpenAITextCompletion

kernel = sk.Kernel()

useAzureOpenAI = False

# Configure AI service used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    azure_text_service = AzureTextCompletion(deployment_name="text", endpoint=endpoint, api_key=api_key)    # set the deployment name to the value 
    kernel.add_text_completion_service("dv", azure_text_service)
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    oai_text_service = OpenAITextCompletion(ai_model_id="text-davinci-003", api_key=api_key, org_id=org_id)
    kernel.add_text_completion_service("dv", oai_text_service)
```

Python

Run the remaining cells of code as usual.

12. In the following function, modify **useAzureOpenAI** to **True**, set the values of **deployment_name, endpoint, and api_key,** which you noted earlier, and execute it.



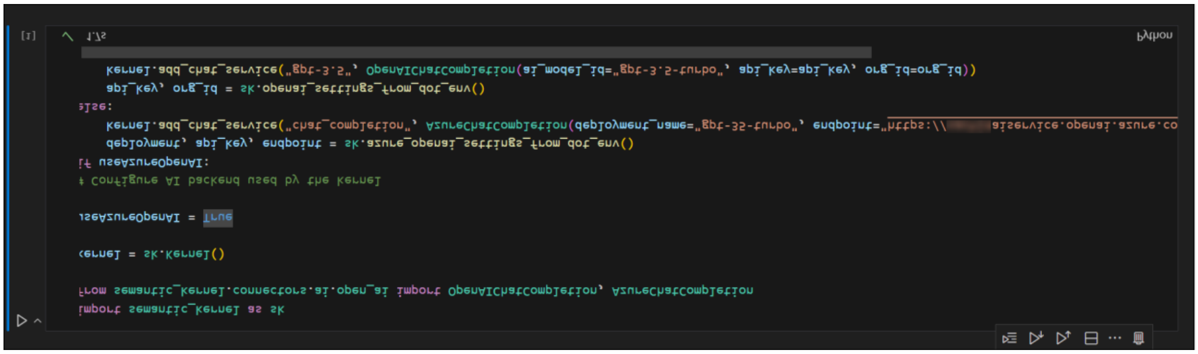
```python
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion, OpenAIChatCompletion

kernel = sk.Kernel()

useAzureOpenAI = True

# Configure AI service used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    azure_chat_service = AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https://[redacted]aiservice.openai.azure.com/", api_key="[redacted]
    kernel.add_chat_service("chat_completion", azure_chat_service)
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    oai_chat_service = OpenAIChatCompletion(ai_model_id="gpt-3.5-turbo", api_key=api_key, org_id=org_id)
    kernel.add_chat_service("chat-gpt", oai_chat_service)
```

✓ 4.1s   Python

Run the remaining cells of code as usual.

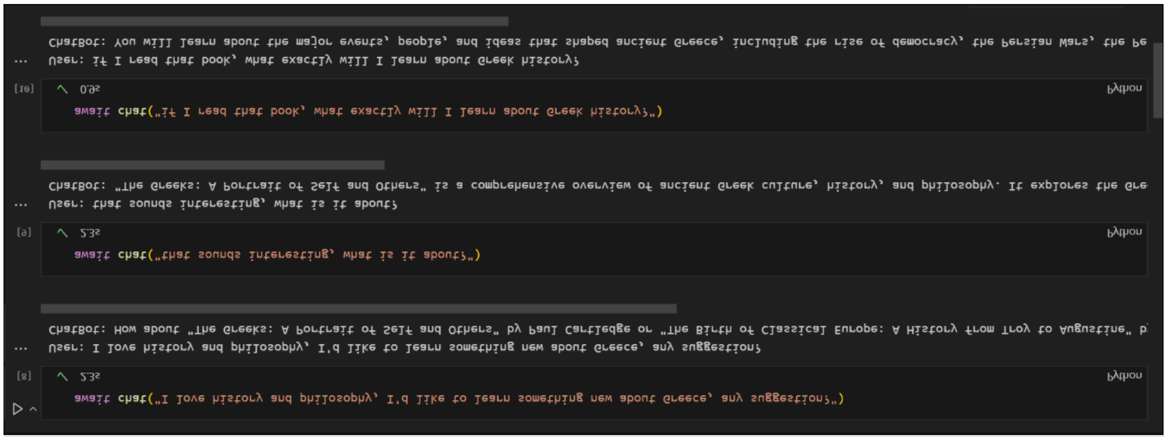
13. After successfully completing the previous notebook, navigate to `04-context-variables-chat.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen.

14. In the following function, modify **useAzureOpenAI** to **True**, set the values of **deployment_name, endpoint, and api_key,** which you noted earlier, and execute it.



```
[1]  ✓ 1.7s                                                                 Python
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import OpenAIChatCompletion, AzureChatCompletion

kernel = sk.Kernel()

useAzureOpenAI = True

# Configure AI backend used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    kernel.add_chat_service("chat_completion", AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https://[redacted]aiservice.openai.azure.co
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    kernel.add_chat_service("gpt-3.5", OpenAIChatCompletion(ai_model_id="gpt-3.5-turbo", api_key=api_key, org_id=org_id))
```

Run the remaining cells of code as usual. An example of output appearing is like this:

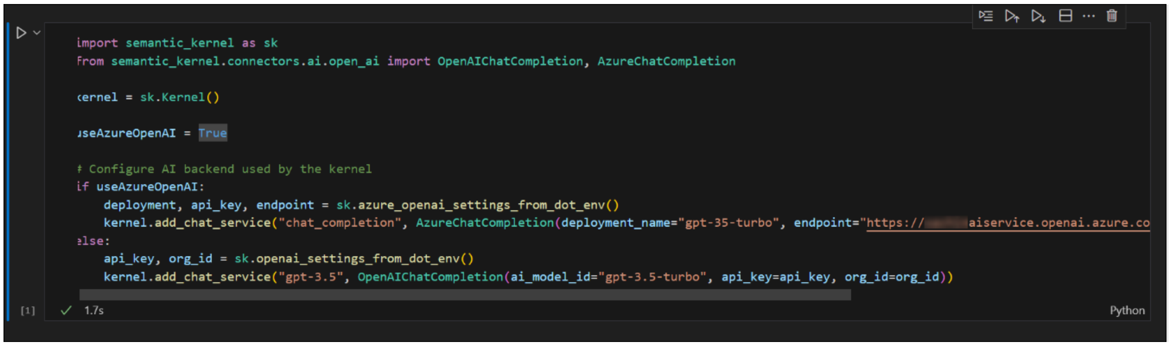


```
[8]  ✓ 2.3s                                                                 Python
  await chat("I love history and philosophy, I'd like to learn something new about Greece, any suggestion?")

... User: I love history and philosophy, I'd like to learn something new about Greece, any suggestion?
    ChatBot: How about "The Greeks: A Portrait of Self and Others" by Paul Cartledge or "The Birth of Classical Europe: A History from Troy to Augustine" b

[9]  ✓ 2.3s                                                                 Python
  await chat("that sounds interesting, what is it about?")

... User: that sounds interesting, what is it about?
    ChatBot: "The Greeks: A Portrait of Self and Others" is a comprehensive overview of ancient Greek culture, history, and philosophy. It explores the Gre

[10]  ✓ 0.9s                                                                Python
  await chat("if I read that book, what exactly will I learn about Greek history?")

... User: if I read that book, what exactly will I learn about Greek history?
    ChatBot: You will learn about the major events, people, and ideas that shaped ancient Greece, including the rise of democracy, the Persian Wars, the Pe
```

Optional Tasks

1. After the required notebooks are run successfully, you can explore and learn about Planner in Semantic Kernel by executing the Jupyter Notebook named `05-using-the-planner.ipynb` using the `Python` programming language. To do this, navigate to `05-using-the-planner.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen.

2. In the following function, modify **useAzureOpenAI** to **True**, set the values of **deployment_name, endpoint, and api_key,** which you noted earlier, and execute it.



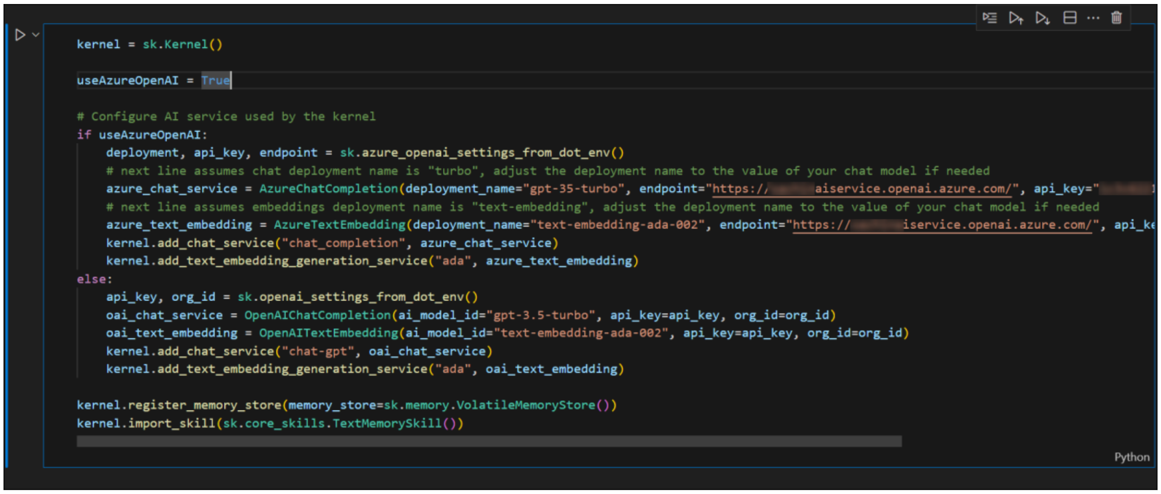
```
import semantic_kernel as sk
from semantic_kernel.connectors.ai.open_ai import OpenAIChatCompletion, AzureChatCompletion

kernel = sk.Kernel()

useAzureOpenAI = True

# Configure AI backend used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    kernel.add_chat_service("chat_completion", AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https://[redacted]aiservice.openai.azure.co
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    kernel.add_chat_service("gpt-3.5", OpenAIChatCompletion(ai_model_id="gpt-3.5-turbo", api_key=api_key, org_id=org_id))
```

[1] ✓ 1.7s Python

Run the remaining cells of code as usual.

3. After successfully completing the previous notebook, navigate to `06-memory-and-embeddings.ipynb` and run the notebook cell-by-cell. Choose **Python <your_version>** whenever a prompt appears on top of the screen.

4. In the following function, modify **useAzureOpenAI** to **True**, set the values of **deployment_name, endpoint, and api_key** for both of your models, which you noted earlier, and execute it.



```
kernel = sk.Kernel()

useAzureOpenAI = True

# Configure AI service used by the kernel
if useAzureOpenAI:
    deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
    # next line assumes chat deployment name is "turbo", adjust the deployment name to the value of your chat model if needed
    azure_chat_service = AzureChatCompletion(deployment_name="gpt-35-turbo", endpoint="https:/[redacted]aiservice.openai.azure.com/", api_key="[redacted]
    # next line assumes embeddings deployment name is "text-embedding", adjust the deployment name to the value of your chat model if needed
    azure_text_embedding = AzureTextEmbedding(deployment_name="text-embedding-ada-002", endpoint="https://[redacted]iservice.openai.azure.com/", api_ke
    kernel.add_chat_service("chat_completion", azure_chat_service)
    kernel.add_text_embedding_generation_service("ada", azure_text_embedding)
else:
    api_key, org_id = sk.openai_settings_from_dot_env()
    oai_chat_service = OpenAIChatCompletion(ai_model_id="gpt-3.5-turbo", api_key=api_key, org_id=org_id)
    oai_text_embedding = OpenAITextEmbedding(ai_model_id="text-embedding-ada-002", api_key=api_key, org_id=org_id)
    kernel.add_chat_service("chat-gpt", oai_chat_service)
    kernel.add_text_embedding_generation_service("ada", oai_text_embedding)

kernel.register_memory_store(memory_store=sk.memory.VolatileMemoryStore())
kernel.import_skill(sk.core_skills.TextMemorySkill())
```

Python

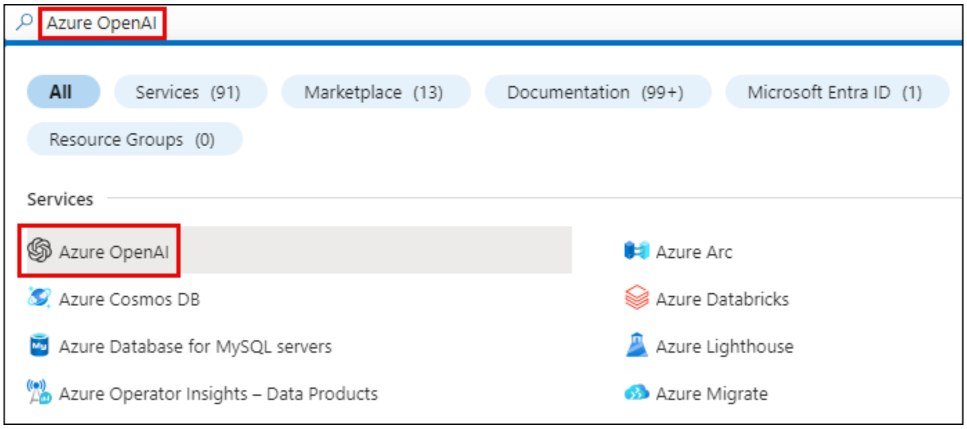
Run the remaining cells of code as usual.

Step 3: Run the Chat Copilot App Locally- Solution Guide

Task 1: Retrieving the Azure OpenAI Service values

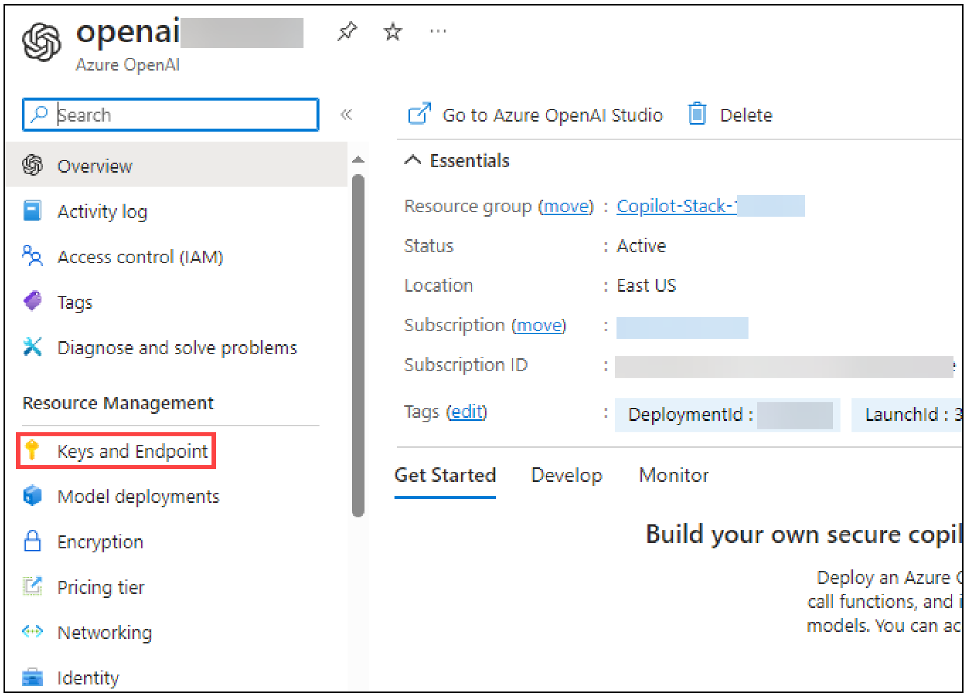
From the Azure portal, you need to retrieve the Azure OpenAI Service Key, Endpoint, and LLM model names deployed in the previous Step.

1. In the Azure Portal, search for **Azure OpenAI** and select it.



Azure OpenAI

2. Select the Azure OpenAI resource created and click on **Keys and Endpoints** from the left pane.



openai[redacted]
Azure OpenAI

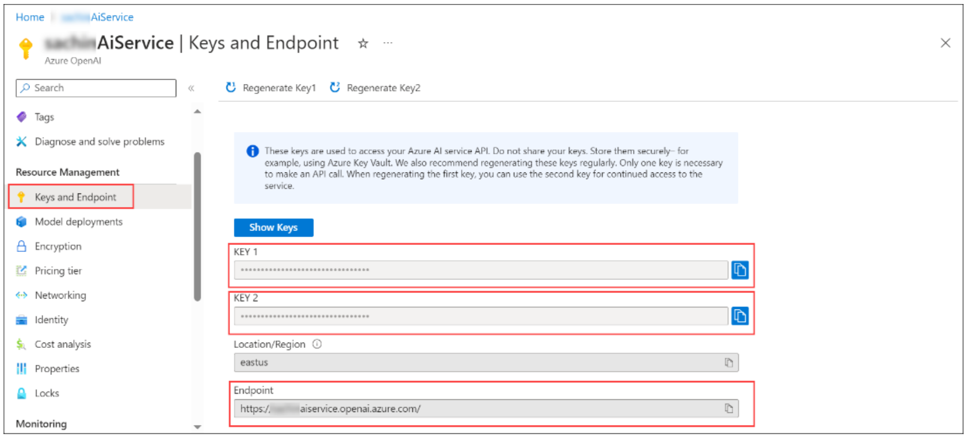
Essentials

Resource group (move) : Copilot-Stack-[redacted]
Status : Active
Location : East US
Subscription (move) : [redacted]
Subscription ID : [redacted]
Tags (edit) : DeploymentId : [redacted] LaunchId : 3

Build your own secure copil

Deploy an Azure (
call functions, and i
models. You can ac

3.Copy the Keys and the Endpoint and store them in Notepad.



These keys are used to access your Azure AI service API. Do not share your keys. Store them securely– for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

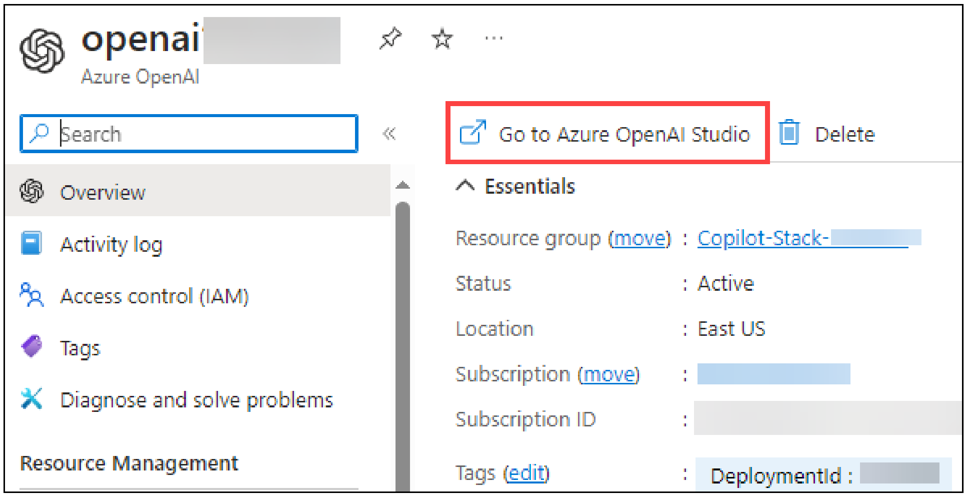
KEY 1: ••••••••••••••••••••••••••••••••

KEY 2: ••••••••••••••••••••••••••••••••

Location/Region: eastus

Endpoint: https:/[redacted]aiservice.openai.azure.com/

4.From the Overview page, click on **Go to Azure OpenAI Studio**.



Resource group (move) : Copilot-Stack-[redacted]

Status : Active

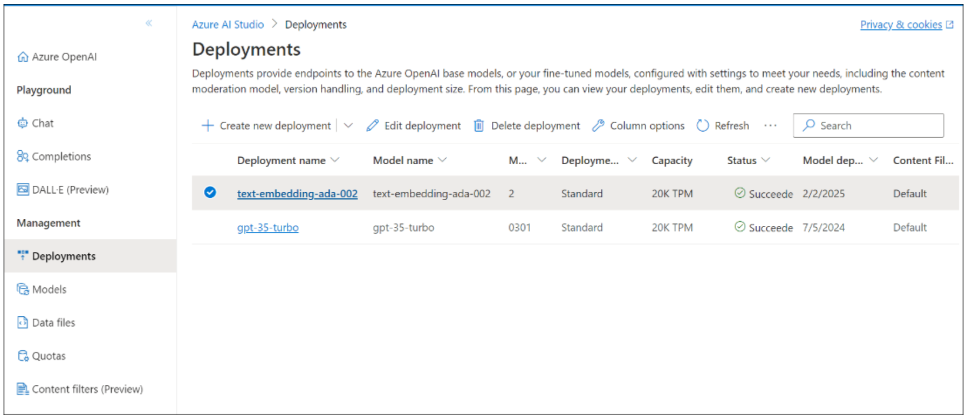
Location : East US

Subscription (move) : [redacted]

Subscription ID : [redacted]

Tags (edit) : DeploymentId : [redacted]

5. Navigate to **Deployments** in the left navigation pane, copy the deployment names of your AI model that you deployed previously, and store them in Notepad.



Deployments

Deployments provide endpoints to the Azure OpenAI base models, or your fine-tuned models, configured with settings to meet your needs, including the content moderation model, version handling, and deployment size. From this page, you can view your deployments, edit them, and create new deployments.

	Deployment name	Model name	M...	Deployme...	Capacity	Status	Model dep...	Content Fil...
✓	text-embedding-ada-002	text-embedding-ada-002	2	Standard	20K TPM	Succeede	2/2/2025	Default
	gpt-35-turbo	gpt-35-turbo	0301	Standard	20K TPM	Succeede	7/5/2024	Default

Task 2: Cloning the Chat-Copilot GitHub Repo

If you have not already cloned the Chat-Copilot GitHub Repository to the environment where you're working on this lab, follow these steps to do so. Otherwise, open the cloned folder in **Visual Studio Code**.

1. Open PowerShell as an administrator.

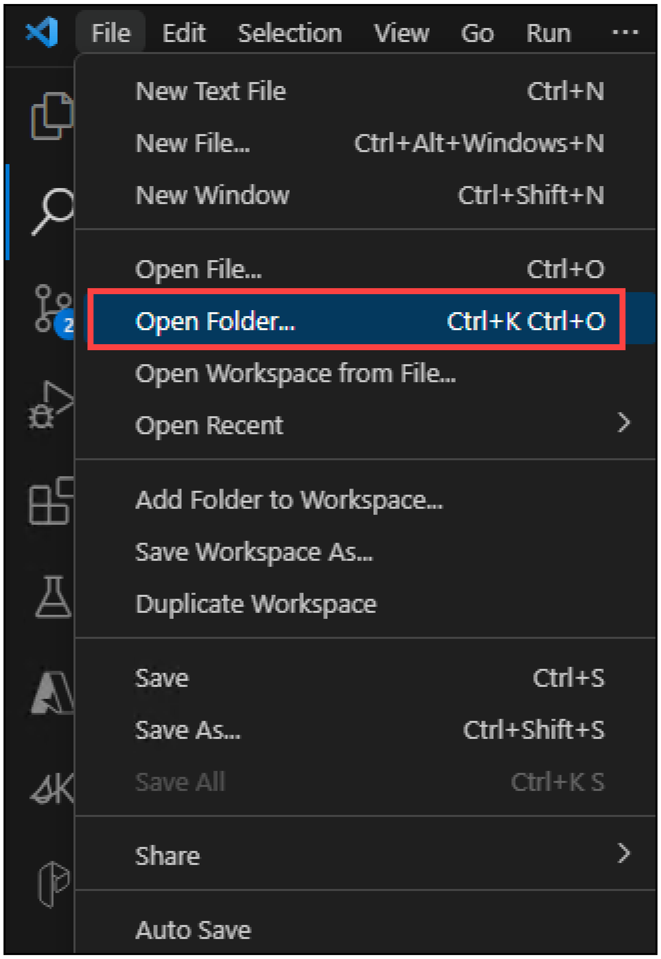
2. Navigate to the directory `C:/Users/azureuser`.

```
cd C:/Users/azureuser
```

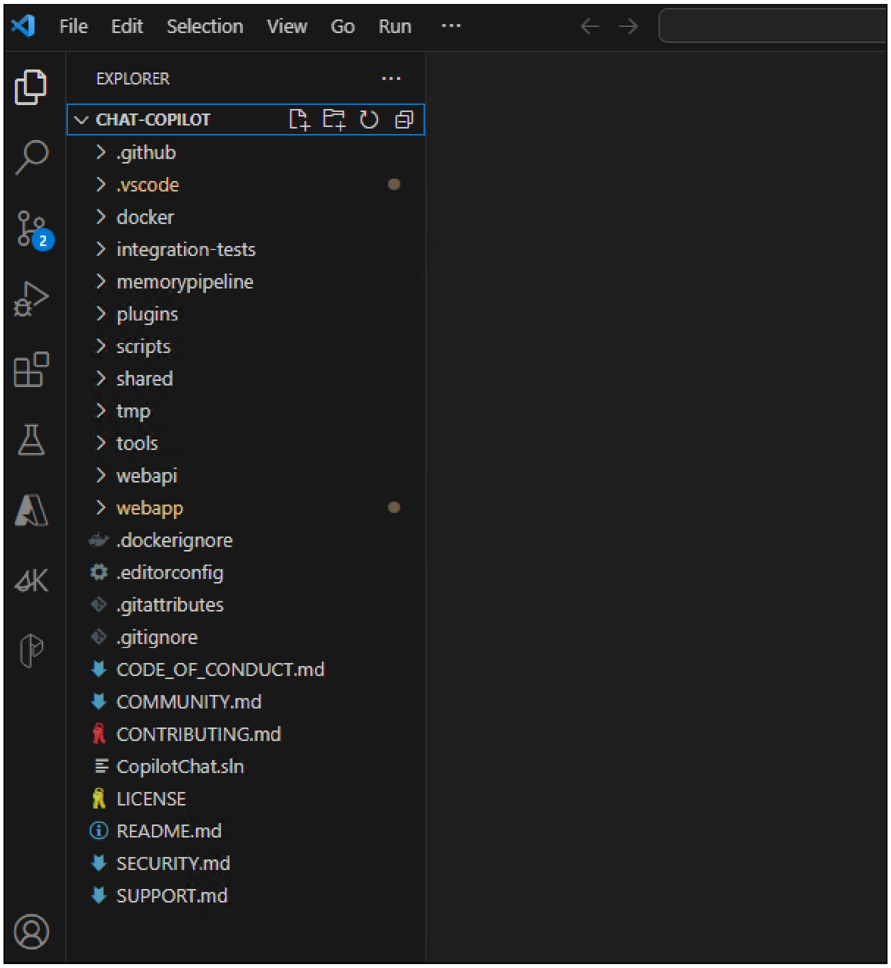
3. Clone the GitHub repository.

```
git clone https://github.com/microsoft/chat-copilot
```

4. Open Visual Studio Code and click on `File> Open folder.`



5.Select **CHAT-COPILOT** and review the files.



Task 3: Setting up the Environment

1. Open PowerShell as an administrator on your local machine. You need to have PowerShell Core 6+ installed, which is different from the default PowerShell installed on Windows.

2. Setup your environment by navigating to the scripts directory of chat-copilot using the command:

```
cd C:\Users\azureuser\chat-copilot\scripts\
```

3. Run the command below to install Chocolatey, dotnet-7.0-sdk, nodejs, and yarn:

```
.\Install.ps1
```

Note: If you receive an error that the script is not digitally signed or cannot execute on the system, you may need to change the execution policy or unblock the script.

Task 4: Configure and run the Chat Copilot App locally

1. Configure Chat Copilot by running the following command:

```
.\Configure.ps1 -AIService {AI_SERVICE} -APIKey {API_KEY} -Endpoint
{AZURE_OPENAI_ENDPOINT} -CompletionModel {DEPLOYMENT_NAME} -
EmbeddingModel {DEPLOYMENT_NAME} -PlannerModel {DEPLOYMENT_NAME}
```

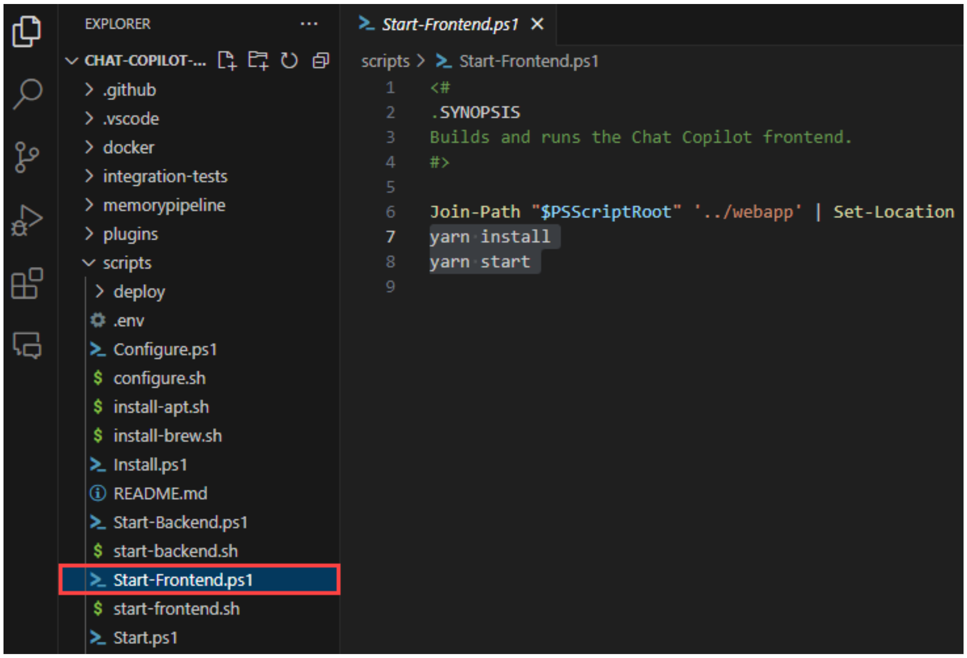
Note: Provide the Azure OpenAI Service Name, Key, Endpoint, and the already deployed model names that you noted down in the previous steps.

2. Finally, run Chat Copilot locally by executing the following command:

```
.\Start.ps1
```

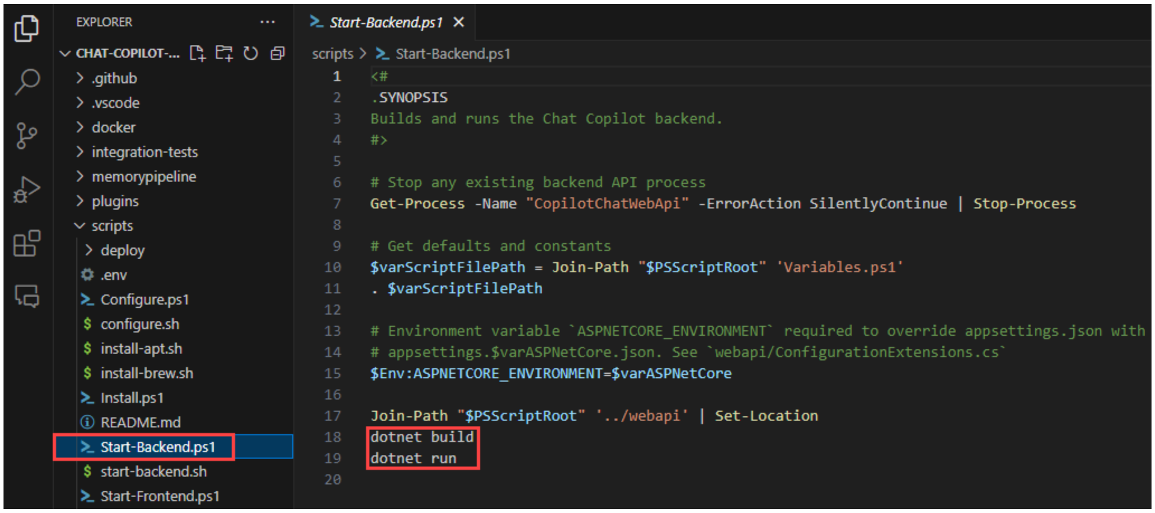
Note: This step starts both the backend API and the frontend application. It may take a few minutes for Yarn packages to install on the first run.
Note: In case of an error, follow the below steps.

3. Navigate to `chat-copilot > scripts > Start-Frontend.ps1` to run the yarn commands.



```
<#
.SYNOPSIS
Builds and runs the Chat Copilot frontend.
#>

Join-Path "$PSScriptRoot" '../webapp' | Set-Location
yarn install
yarn start
```

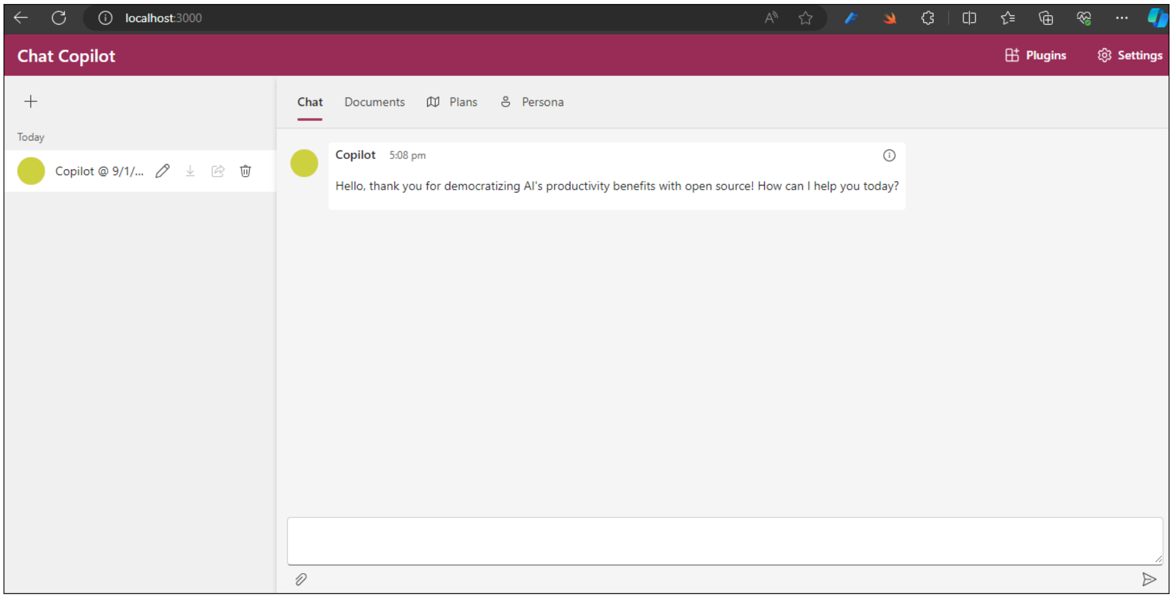
4. Navigate to `chat-copilot > scripts > Start-Backend.ps1` to run the dotnet commands.



```
<#
.SYNOPSIS
Builds and runs the Chat Copilot backend.
#>

# Stop any existing backend API process
Get-Process -Name "CopilotChatWebApi" -ErrorAction SilentlyContinue | Stop-Process

# Get defaults and constants
$varScriptFilePath = Join-Path "$PSScriptRoot" 'Variables.ps1'
. $varScriptFilePath

# Environment variable `ASPNETCORE_ENVIRONMENT` required to override appsettings.json with
# appsettings.$varASPNetCore.json. See `webapi/ConfigurationExtensions.cs`
$Env:ASPNETCORE_ENVIRONMENT=$varASPNetCore

Join-Path "$PSScriptRoot" '../webapi' | Set-Location
dotnet build
dotnet run
```

Note: Once done, navigate to the scripts directory and run the start command again.

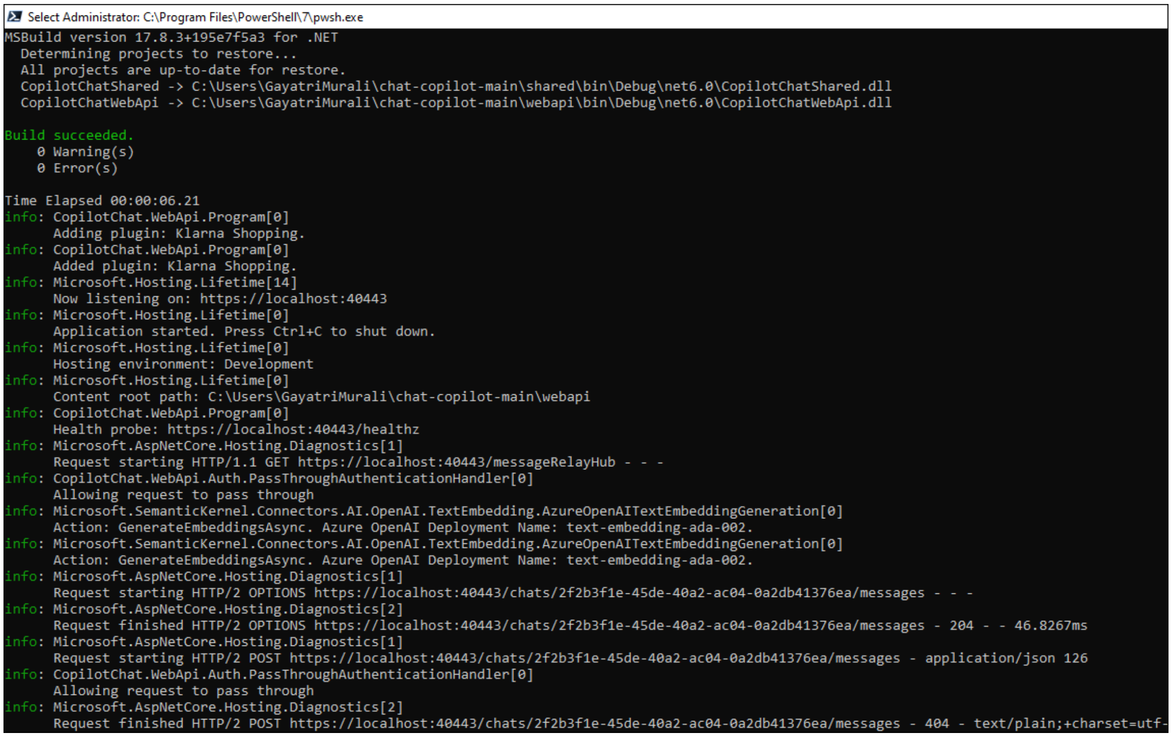
5. You will get an output similar to this for the frontend:



Copilot 5:08 pm

Hello, thank you for democratizing AI's productivity benefits with open source! How can I help you today?

6. You will get an output similar to this for the backend:



Select Administrator: C:\Program Files\PowerShell\7\pwsh.exe

```
MSBuild version 17.8.3+195e7f5a3 for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  CopilotChatShared -> C:\Users\GayatriMurali\chat-copilot-main\shared\bin\Debug\net6.0\CopilotChatShared.dll
  CopilotChatWebApi -> C:\Users\GayatriMurali\chat-copilot-main\webapi\bin\Debug\net6.0\CopilotChatWebApi.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:06.21
info: CopilotChat.WebApi.Program[0]
      Adding plugin: Klarna Shopping.
info: CopilotChat.WebApi.Program[0]
      Added plugin: Klarna Shopping.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:40443
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\GayatriMurali\chat-copilot-main\webapi
info: CopilotChat.WebApi.Program[0]
      Health probe: https://localhost:40443/healthz
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/1.1 GET https://localhost:40443/messageRelayHub - - -
info: CopilotChat.WebApi.Auth.PassThroughAuthenticationHandler[0]
      Allowing request to pass through
info: Microsoft.SemanticKernel.Connectors.AI.OpenAI.TextEmbedding.AzureOpenAITextEmbeddingGeneration[0]
      Action: GenerateEmbeddingsAsync. Azure OpenAI Deployment Name: text-embedding-ada-002.
info: Microsoft.SemanticKernel.Connectors.AI.OpenAI.TextEmbedding.AzureOpenAITextEmbeddingGeneration[0]
      Action: GenerateEmbeddingsAsync. Azure OpenAI Deployment Name: text-embedding-ada-002.
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 OPTIONS https://localhost:40443/chats/2f2b3f1e-45de-40a2-ac04-0a2db41376ea/messages - - -
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished HTTP/2 OPTIONS https://localhost:40443/chats/2f2b3f1e-45de-40a2-ac04-0a2db41376ea/messages - 204 - - 46.8267ms
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 POST https://localhost:40443/chats/2f2b3f1e-45de-40a2-ac04-0a2db41376ea/messages - application/json 126
info: CopilotChat.WebApi.Auth.PassThroughAuthenticationHandler[0]
      Allowing request to pass through
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished HTTP/2 POST https://localhost:40443/chats/2f2b3f1e-45de-40a2-ac04-0a2db41376ea/messages - 404 - text/plain;+charset=utf-
```