Generate and Improve codes

Learning Objectives

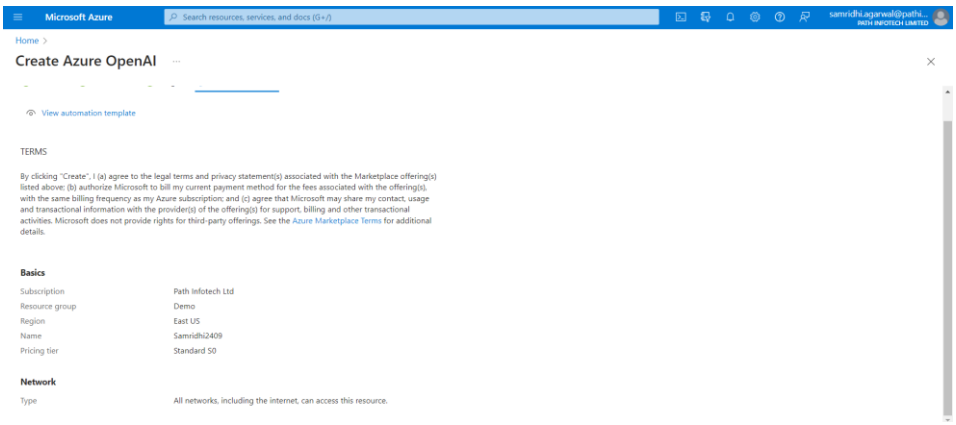
- Use natural language prompts to generate code
- Build unit tests and understand complex code with AI models
- Generate comments and documentation for existing codes

The Azure OpenAI Service provides users with the capability to use expansive language models (LLMs) for the purpose of generating various forms of content, such as computer codes. This feature enables developers to produce and enhance computer code in many languages, hence enhancing productivity and comprehension.

This session explores the utilization of Azure OpenAI for code generation and assistance in diverse developing endeavors.

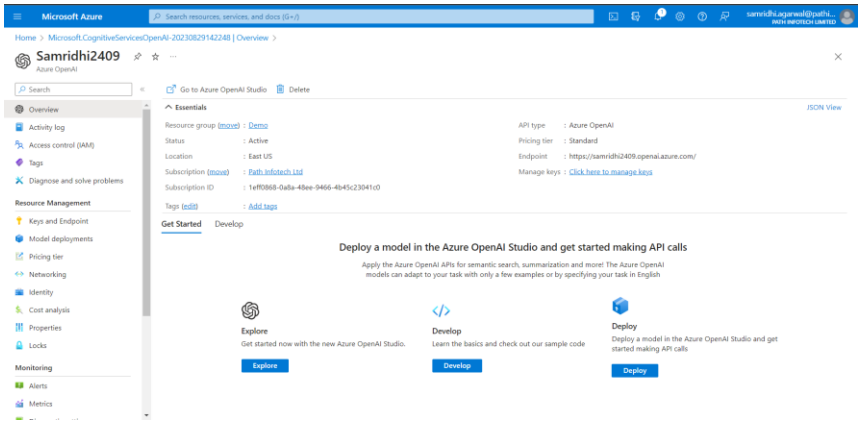
The Azure OpenAI Service models provide the capability to produce code via the use of natural language prompts. This functionality allows for the identification and resolution of errors within pre-existing code, as well as the provision of code comments. These models have the capability to elucidate and streamline pre-existing codes, hence aiding developers in comprehending the semantics, functioning, and potential enhancements of those codes.

1. Provision an Azure OpenAI resource with the following credentials:
    i. Subscription: Path InfotechLtd
    ii. Resource group: Demo
    iii. Region: EastUS
    iv. Name: Samridhi2409
    v. Pricing: Standard S0



Wait for the deployment to complete then go to the deployed Azure OpenAI resource in Azure portal.

Navigate to Keys and Endpoint page and save those to a text file to use later.

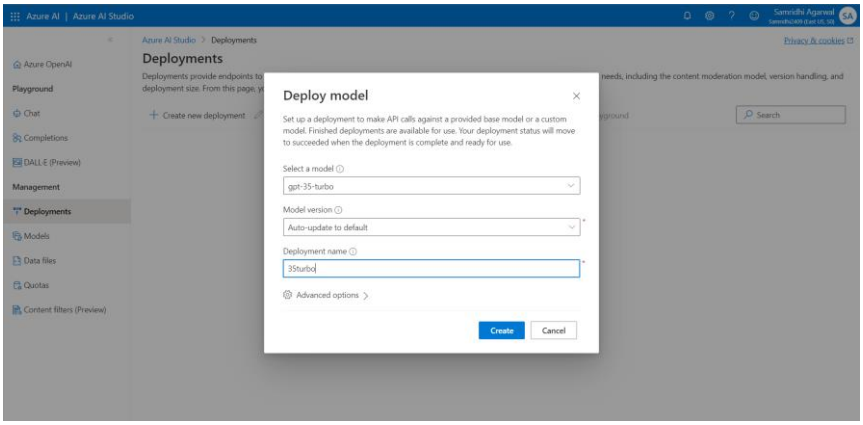


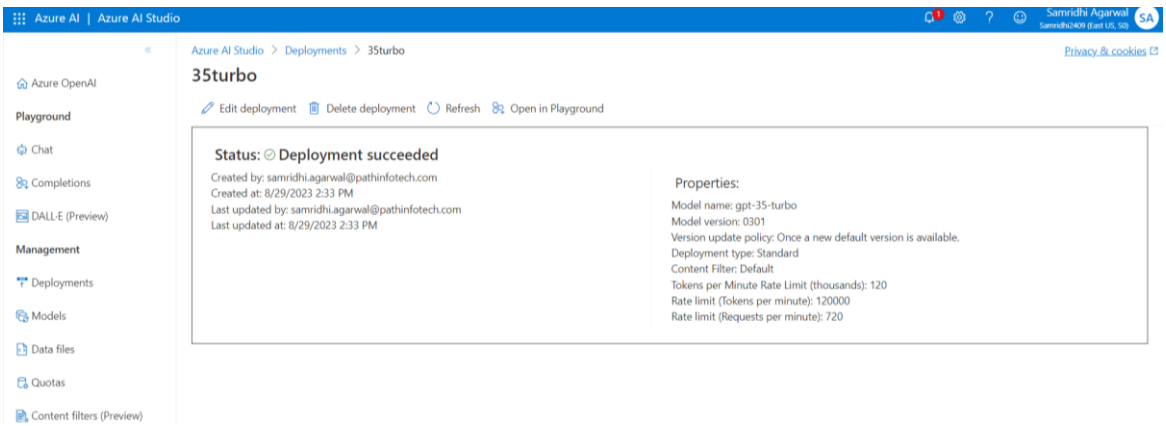
2. Deploying a model

   To use Azure OpenAI API for code generation, a model must be deployed first to use through the Azure OpenAI Studio. Once deployed, this model is used with the playground and reference that model in the app.

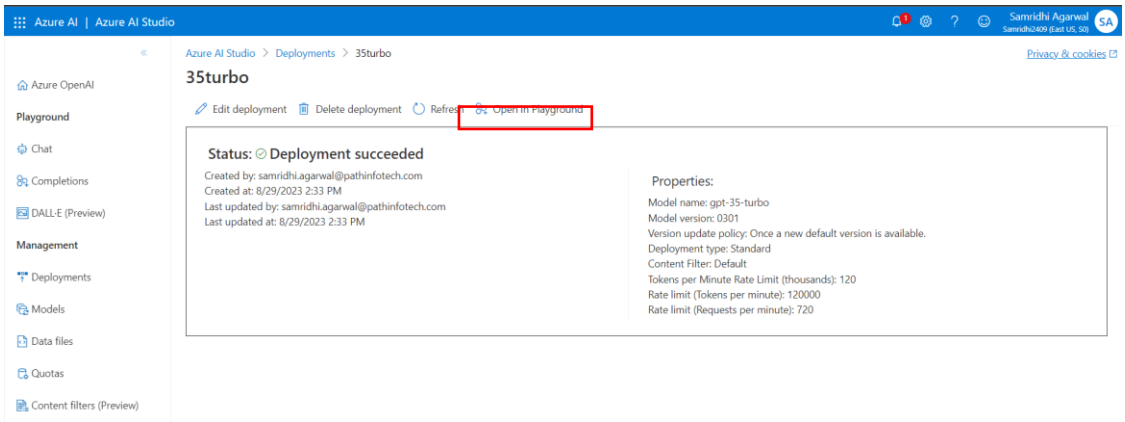
   Deploy a model with the following credentials:

   i. Model: gpt-35-turbo
   ii. Model version: default
   iii. Name: 35turbo

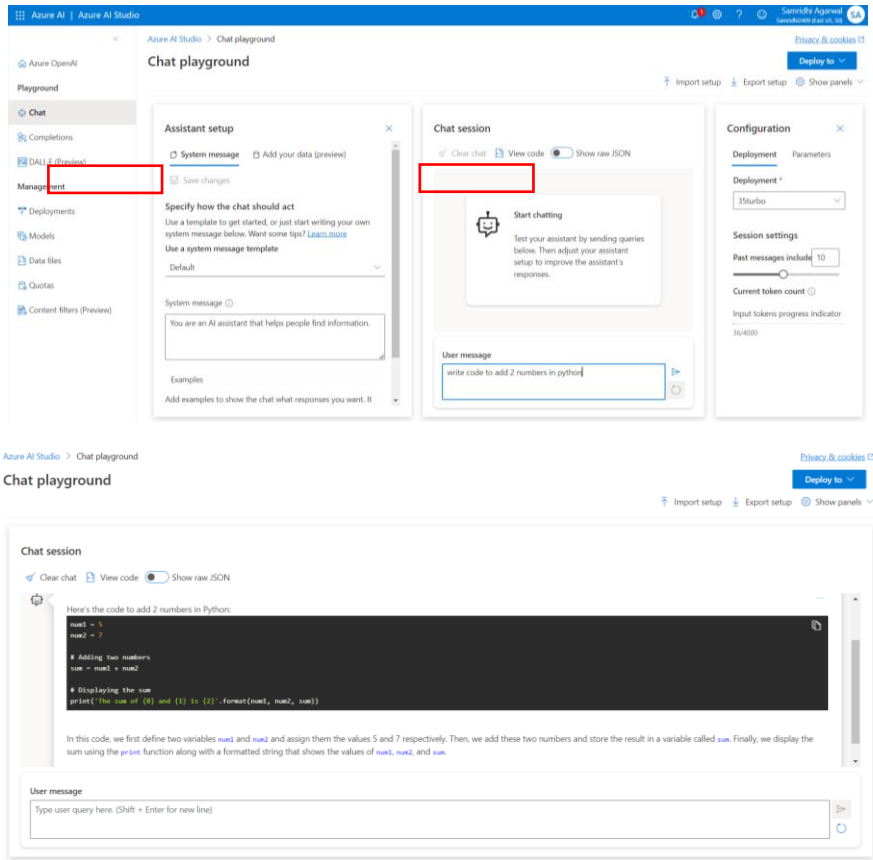




3. Generating code in chat playground
   Before using in app, examining Azure OpenAI in the chat playground



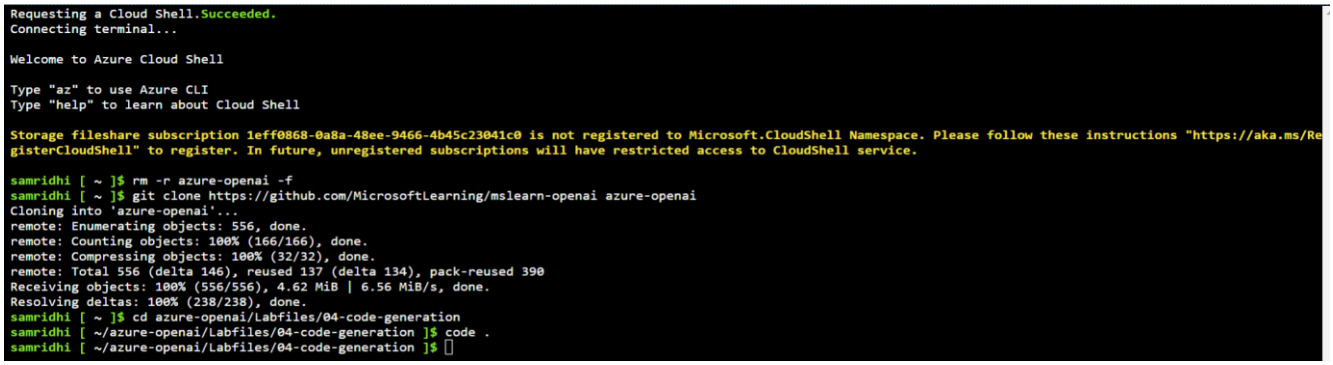
In assistant setup section, select default message template. In Chat session section, enter prompt to code for.



4. Setting up application in cloud shell
   To show integration with Azure OpenAI model, use a short command line application that runs in cloud shell on Azure.
   In the cloud shell select Bash. Then run the following command in the terminal to download the sample application and save it in a folder named azure-openai.
   - 〉 rm -r azure-openai -f
   - 〉 git clone https://github.com/MicrosoftLearning/mslearn-openai azure-openai

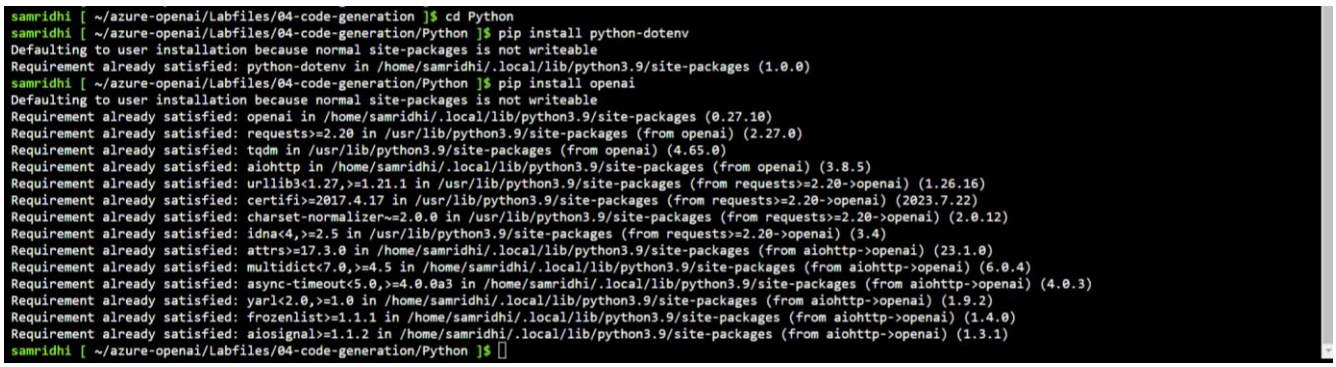
   Changing directory to Labfiles
   - 〉 cd azure-openai/Labfiles/04-code-generation

   Start coding.
   - 〉 code .



5. Configuring the application
   In the code editor, expand the preferred language folder and update the configuration values to include the endpoint, key and deployment.

6. Navigate to the preferred language folder and install the necessary packages.



7. Select the code file in the folder for preferred language and add necessary libraries as well as the necessary client configurations.
    〉 import openai

```
# Get configuration settings
load_dotenv()
azure_oai_endpoint = os.getenv("https://samridhi2409.openai.azure.com/")
azure_oai_key = os.getenv("9b614fe67f5b42188675f45fccd5d90c")
azure_oai_model = os.getenv("35turbo")

# Set OpenAI configuration settings
openai.api_type = "azure"
openai.api_base = azure_oai_endpoint
openai.api_version = "2023-08-31"
openai.api_key = azure_oai_key
```

8. In the functions that call Azure OpenAI model, add the code to format and send the request to the model.

```
# Build the messages array
messages =[
    {"role":"system", "content":system_message},
    {"role":"user", "content":user_message}
]

# Call the Azure OpenAI model
response = openai.ChatCompletion.create(
    engine=model,
    message=messages,
    temperature=0.7,
    max_tokens=1000
)
```

9. Running the application.
   In the code editor, expand the sample-code folder and briefly observe the function and the app for preferred language. These files will be used for tasks in app.
    i. In the cloud shell bash terminal, navigate to the preferred language folder.
    ii. Run the application, Python:python code-generation.py
    iii. Choose option 1 to add comments to the code. The results will be put into result/app.txt. Open the file to compare the function file in sample-code.
    iv. Choose option 2 to write unit tests for the same function. The results will replace what was in result/app.txt, and details 4-unit tests for that function.
    v. Choose option 3 to fix bugs in an app for playing Go Fish. The results will replace what was in result/app.txt and should have very similar code with a few things corrected.

       The app for Go Fish in sample-code can be run, if you replace the lines with bugs with the response from Azure OpenAI. If you run it without the fixes, it will not work correctly.

Note: a. Fixes are mad in line 18 and 31.

b. Even though the code for this Go Fish app was corrected for some syntax, it's not a strictly accurate representation of the game. If you look closely, there are issues with not checking if the deck is empty when drawing cards, not removing pairs from the players hand when they get a pair, and a few other bugs that require understanding of card games to realize. This is a great example of how useful generative AI models can be to assist with code generation but can't be trusted as correct and need to be verified by the developer.

If you would like to see the full response from Azure OpenAI, you can set the printFullResponse variable to True, and rerun the app.

```python
import os
from dotenv import load_dotenv

# Add OpenAI import
import openai

# Set to True to print the full response from OpenAI for each call
printFullResponse = False

def main():

    try:

        # Get configuration settings
        load_dotenv()
        azure_oai_endpoint = os.getenv("https://samridhi2409.openai.azure.com/")
        azure_oai_key = os.getenv("9b614fe67f5b42188675f45fccd5d90c")
        azure_oai_model = os.getenv("35turbo")

        # Set OpenAI configuration settings
        openai.api_type = "azure"
        openai.api_base = azure_oai_endpoint
        openai.api_version = "2023-08-31"
        openai.api_key = azure_oai_key

        while True:
            print('\n1: Add comments to my function\n' +
                '2: Write unit tests for my function\n' +
                '3: Fix my Go Fish game\n' +
                '\"quit\" to exit the program\n')
            command = input('Enter a number to select a task:')
            if command == '1':
                file = open(file="../sample-code/function/function.py", encoding="utf8").read()
                prompt = "Add comments to the following function. Return only the commented code.\n---\n" + file
                call_openai_model(prompt, model=azure_oai_model)
            elif command =='2':
                file = open(file="../sample-code/function/function.py", encoding="utf8").read()
                prompt = "Write four unit tests for the following function.\n---\n" + file
```

```
            call_openai_model(prompt, model=azure_oai_model)
        elif command =='3':
            file = open(file="../sample-code/go-fish/go-fish.py", encoding="utf8").read()
            prompt = "Fix the code below for an app to play Go Fish with the user. Return only the corrected code.\n---\n" + file
            call_openai_model(prompt, model=azure_oai_model)
        elif command.lower() == 'quit':
            print('Exiting program...')
            break
        else :
            print("Invalid input. Please try again.")

    except Exception as ex:
        print(ex)

def call_openai_model(prompt, model):
    # Provide a basic user message, and use the prompt content as the user message
    system_message = "You are a helpful AI assistant that helps programmers write code."
    user_message = prompt

    # Build the messages array
    messages =[
        {"role":"system", "content":system_message},
        {"role":"user", "content":user_message}
    ]

    # Call the Azure OpenAI model
    response = openai.ChatCompletion.create(
        engine=model,
        message=messages,
        temperature=0.7,
        max_tokens=1000
    )
    # Print the response to the console, if desired
    if printFullResponse:
        print(response)

```

```
        # Write the response to a file
        results_file = open(file="result/app.txt", mode="w", encoding="utf8")
        results_file.write(response.choices[0].message.content)
        print("\nResponse written to result/app.txt\n\n")

    if __name__ == '__main__':
        main()
```

REFERENCE

[Generate code with Azure OpenAI Service - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/generate-code-azure-openai/)

https://learn.microsoft.com/en-us/training/modules/generate-code-azure-openai/