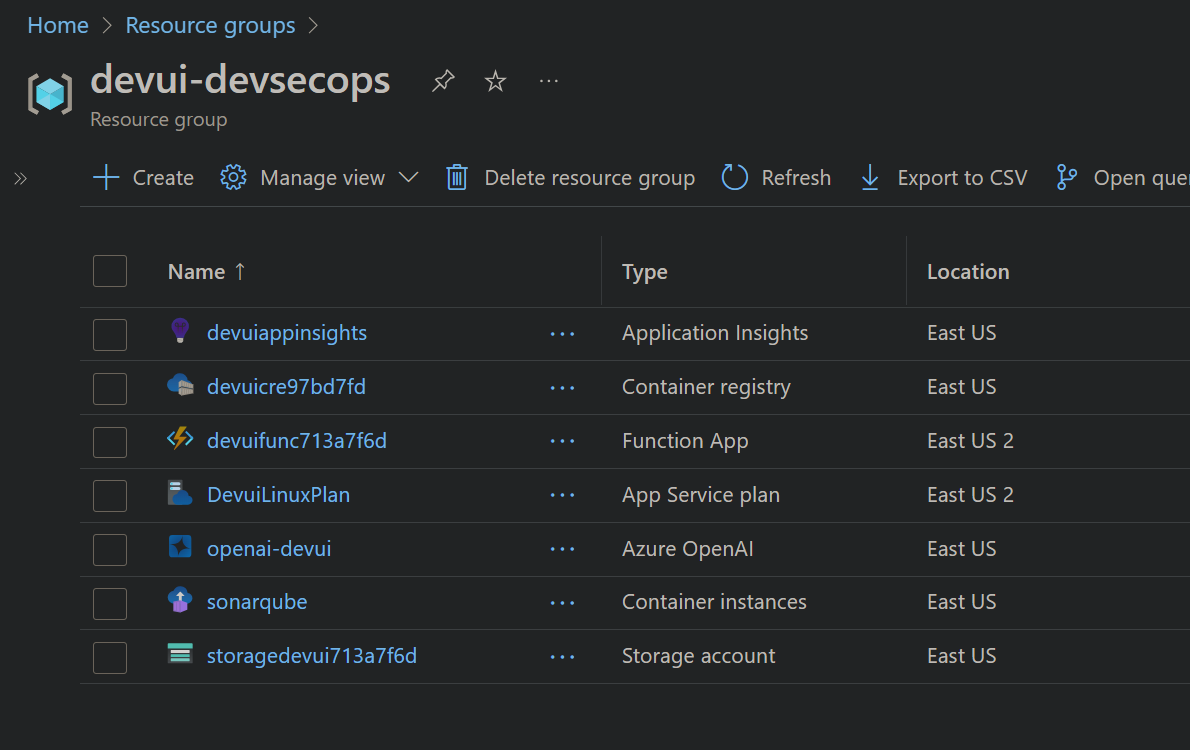
Initial:

As shown in screenshot, we've created Azure resources to facilitate pipeline to scan the repository https://github.com/stevenscwu/Devui with is only of frontend. As the previous process, we'll build YAML file to facilitate the flow start from checkout code, download SonarScanner, run SonarQube scan on self-hosted Azure VM, generate SonarQube report, beautify and upload the json report to Azure blob, reduce the sonarQube json report size, also upload the reduced report to Azure blob to trigger chatGPT summary, and store the chatGPT report also to Azure blob. First, you'll carefully review the attached sonar-gpt-pipeline.yml file to check the overall flow. Second, you'll guide me step by step implement the \_\_init\_\_.py file to realize the logic inside Azure openAI. I've also attach the \_\_init\_\_.py from previouse project for your referance.



**Step 1: Review of YAML Pipeline (sonar-gpt-pipeline.yml)**

I have reviewed your provided YAML file (sonar-gpt-pipeline.yml). The pipeline performs the following steps:

1. **Checkout Repository**: Retrieves source code from GitHub.
2. **Install .NET SDK**: Sets up the .NET environment.
3. **Install and run SonarScanner**:
   * Downloads SonarScanner.
   * Executes the scan with provided project keys and endpoints.
   * Builds the project using .NET.
4. **SonarQube Report Generation**:
   * Waits for the SonarQube report to finalize.
   * Fetches the report using the SonarQube API.
5. **JSON Report Processing**:
   * Beautifies and reduces the JSON report.
   * Uploads both full and reduced JSON reports to Azure Blob Storage.
6. **Trigger Azure Function**:
   * Uploading the reduced JSON triggers Azure Function (blob trigger).
   * Azure Function runs GPT-4 summarization on the SonarQube report.

The provided YAML structure and flow look coherent, structured, and clearly defined for your intended pipeline.

**Step 2: Implementation of \_\_init\_\_.py for Azure OpenAI Logic**

You have also provided an earlier \_\_init\_\_.py from a previous project, which I reviewed. I'll guide you step-by-step to implement the logic inside Azure OpenAI in your current scenario based on this file.

**Step-by-Step Implementation for \_\_init\_\_.py**

Follow these steps to implement Azure OpenAI logic inside your Azure Function:

**1. Ensure Required Packages**

Your Azure Function (Python) environment needs the following dependencies in requirements.txt:

pgsql

複製程式碼

azure-functions

openai

**2. Azure Function Structure**

Your directory structure for the Azure Function (BlobTriggerGPT) should look like this:

pgsql

複製程式碼

BlobTriggerGPT/

├── \_\_init\_\_.py

└── function.json

**3. Configure function.json**

Define your input and output binding clearly, for example:

json

複製程式碼

{

"scriptFile": "\_\_init\_\_.py",

"bindings": [

{

"name": "inputblob",

"type": "blobTrigger",

"direction": "in",

"path": "sonarinput/{name}",

"connection": "AzureWebJobsStorage"

},

{

"name": "outputblob",

"type": "blob",

"direction": "out",

"path": "gptsummary/{name}.md",

"connection": "AzureWebJobsStorage"

}

]

}

**4. Implementation of \_\_init\_\_.py**

Here's a refined and clarified version of your \_\_init\_\_.py tailored specifically for your current scenario:

python

複製程式碼

import logging

import os

import json

import time

import azure.functions as func

from openai import AzureOpenAI

def main(inputblob: func.InputStream, outputblob: func.Out[str]):

logging.info(f"Processing blob: {inputblob.name}, size: {inputblob.length} bytes")

data = inputblob.read().decode('utf-8')

# Guard against large files

if len(data) > 1\_000\_000:

logging.warning("Input file too large, skipping processing.")

return

# Prompt engineering for GPT summary

prompt = (

"You are an expert security engineer tasked with reviewing a SonarQube static analysis report. "

"Analyze the following JSON report and provide:\n"

"1. \*\*Top 10 Critical Issues\*\*: Severity, file, line, rule, issue description, and risk if unresolved.\n"

"2. \*\*Issue Severity Summary\*\*: Count by severity levels.\n"

"3. \*\*Type of Issues\*\*: Breakdown by bugs, vulnerabilities, and code smells.\n"

"4. \*\*Key Patterns/Hotspots\*\*: Highlight files/modules with recurring issues.\n"

"5. \*\*Actionable Recommendations\*\*: Prioritize top 3–5 actions to mitigate critical risks.\n"

"Respond concisely and format output in markdown clearly."

)

# Fetch Azure OpenAI settings from environment variables

api\_base = os.getenv("OPENAI\_API\_BASE")

api\_key = os.getenv("OPENAI\_API\_KEY")

deployment\_name = os.getenv("OPENAI\_DEPLOYMENT\_NAME")

api\_version = os.getenv("OPENAI\_API\_VERSION", "2024-12-01-preview")

if not all([api\_base, api\_key, deployment\_name]):

logging.error("Missing Azure OpenAI configuration.")

return

# Create Azure OpenAI client

client = AzureOpenAI(api\_key=api\_key, azure\_endpoint=api\_base, api\_version=api\_version)

# Call Azure OpenAI with provided prompt and report

summary = ""

start = time.time()

try:

response = client.chat.completions.create(

model=deployment\_name,

messages=[

{"role": "system", "content": prompt},

{"role": "user", "content": data}

],

temperature=0.2,

max\_tokens=2000

)

summary = response.choices[0].message.content

logging.info(f"Token usage: {response.usage}")

except Exception as e:

logging.error(f"OpenAI API error: {e}")

summary = f"OpenAI API encountered an error: {e}"

latency\_ms = (time.time() - start) \* 1000

logging.info(f"OpenAI latency: {latency\_ms:.0f} ms")

# Write GPT summary to blob output

outputblob.set(summary)

**5. Set Up Azure OpenAI Environment Variables**

Configure these environment variables on your Azure Function App settings (portal):

* OPENAI\_API\_BASE: URL endpoint of your Azure OpenAI instance.
* OPENAI\_API\_KEY: API key from your Azure OpenAI resource.
* OPENAI\_DEPLOYMENT\_NAME: Name of your GPT-4 deployment in Azure OpenAI.
* OPENAI\_API\_VERSION: API version (default "2024-12-01-preview").

Example values from your current setup:

* OPENAI\_API\_BASE: e.g., https://<your-resource-name>.openai.azure.com/
* OPENAI\_API\_KEY: from Azure portal
* OPENAI\_DEPLOYMENT\_NAME: e.g., gpt-4.1

**6. Configure Blob Containers**

From your Azure screenshot, your Blob Storage account storagedevui713a7f6d should have:

* **Input container**: sonarinput (triggers Azure Function).
* **Output container**: gptsummary (GPT-4 markdown summaries).

Make sure these containers exist in your storage account.

**Final Flow Overview (For Your Reference):**

pgsql

複製程式碼

GitHub Repo Checkout

└── .NET SDK Setup

└── SonarScanner Execution (self-hosted Azure VM)

└── Generate SonarQube JSON Report

└── Upload Full Report to Blob (fulljson)

└── Reduce JSON size and Upload Reduced Report (sonarinput)

└── Blob Trigger Azure Function (GPT Summary)

└── Save GPT Summary to Blob (gptsummary)

You are now ready to integrate and run your pipeline. Let me know if you need further clarification or troubleshooting at any step!