Chatbot documentation

# Set up

## Establish the directory of the python interpreter

In VSCode

1. **View** -> **Command Palette**
2. *Python: Select Interpreter*
3. **Enter interpreter path**
4. **Browse your file system to find a Python interpreter**
5. Browse to: *C:\Users\andrew.dilley\development\chatbot3\venv\Scripts\python.exe*

Note: can use this Python interpreter across multiple development directory structures

## Creating a new chatbot development environment

### Establish the directory structure

1. Copy the last *chatbot* directory and paste it into the *development* directory
2. Rename it to *chatbot<next number in the sequence>*
3. Delete the *venv* directory

### Create and activate the environment

1. Open a command prompt (cmd)
2. Go to the newly created development directory
3. Create the environment with python -m venv venv
4. deactivate
5. Activate the environment with *venv\Scripts\activate*

### reset the python interpreter

VSCode

View

Command palette

Python: select interpreter

Enter interpreter path

Browse

Eg C:\Users\andrew.dilley\development\chatbot11\venv\Scripts\python.exe

Select interpreter

### Verify the correct interpreter

In powershell

where python

(should return the current virtual environment)

In python shell

import sys

print(sys.executable)

### Install the dependencies

pip install -r requirements.txt

### check for any specific packages

pip show PyPDF2

### Committing a change to the code

#### Stage all modified files

In the CMD prompt and in the development directory type:

git add .

#### Commit the changes

In the CMD prompt and in the development directory type:

git commit -m "Your commit message"

data privacy on the OpenAI API Platform

**Services for businesses, such as ChatGPT Team, ChatGPT Enterprise, and our API Platform**

By default, we do not train on any inputs or outputs from our products for business users, including ChatGPT Team, ChatGPT Enterprise, and the API. We offer API customers a way to opt-in to share data with us, such as by [providing feedback in the Playground](https://help.openai.com/en/articles/9883556-providing-feedback-in-the-api-playground), which we then use to improve our models. Unless they explicitly opt-in, organizations are opted out of data-sharing by default.

Please see our [Enterprise Privacy page](https://openai.com/enterprise-privacy) for information on how we handle business data.

[How your data is used to improve model performance | OpenAI Help Center](https://help.openai.com/en/articles/5722486-how-your-data-is-used-to-improve-model-performance)

OpenAI encrypts all data at rest (AES-256) and in transit between our customers and us and between us and our service providers (TLS 1.2+), and uses strict access controls to limit who can access data. Our security team has an on-call rotation that has 24/7/365 coverage and is paged in case of any potential security incident. We offer a [Bug Bounty Program⁠](https://openai.com/blog/bug-bounty-program) for responsible disclosure of vulnerabilities discovered on our platform and products. Please visit our [Trust Portal⁠(opens in a new window)](https://trust.openai.com/) for more details.

[Enterprise privacy at OpenAI | OpenAI](https://openai.com/enterprise-privacy/)

[Usage policies | OpenAI](https://openai.com/policies/usage-policies/)

**Azure OpenAI Service**

But – OpenAI only has data centres in the US so we need to utilise **Azure OpenAI Service which has Australian based data centres**

**Your prompts (inputs) and completions (outputs), your embeddings, and your training data:**

* **are NOT available to other customers.**
* **are NOT available to OpenAI.**
* **are NOT used to improve OpenAI models.**
* **are NOT used to train, retrain, or improve Azure OpenAI Service foundation models.**
* **are NOT used to improve any Microsoft or 3rd party products or services without your permission or instruction.**
* **Your fine-tuned Azure OpenAI models are available exclusively for your use.**

**The Azure OpenAI Service is operated by Microsoft as an Azure service; Microsoft hosts the OpenAI models in Microsoft's Azure environment and the Service does NOT interact with any services operated by OpenAI (e.g. ChatGPT, or the OpenAI API).**

[**Data, privacy, and security for Azure OpenAI Service - Azure AI services | Microsoft Learn**](https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy?utm_source=chatgpt.com&tabs=azure-portal)

* **Your organization’s data is not used to train foundation models**. Microsoft’s generative AI solutions, including Azure OpenAI Service and Copilot services and capabilities, do not use your organization’s data to train foundation models without your permission. Your data is not available to OpenAI or used to train OpenAI models.

[How-to: Create and deploy an Azure OpenAI Service resource - Azure OpenAI | Microsoft Learn](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/create-resource?pivots=web-portal)

[Protecting the data of our commercial and public sector customers in the AI era - Microsoft On the Issues](https://blogs.microsoft.com/on-the-issues/2024/03/28/data-protection-responsible-ai-azure-copilot/?utm_source=chatgpt.com)

Docker

Build the image

cmd

docker build --tag hello-world .

view the available images

docker images

check if any docker images are running

docker ps

docker ps -a

docker start <NAME>

docker stop <NAME>

docker rm <NAME>

clear

docker logs hello

**Verify File Presence**

After rebuilding your Docker image, run the container interactively to check if the documents folder and its files exist:

bash

Copy code

docker run -it hello-world bash

Inside the container, navigate to /app/documents:

bash

Copy code

cd /app/documents

ls

Ensure Acceptable.docx and other required files are present.

**Force Rebuild Without Using Cache**

To ensure a completely fresh build (not reusing intermediate layers), you can use the --no-cache option:

bash

Copy code

docker build --no-cache --tag hello-world .

run the image

docker run -p 8080:80 –name hello -d hello-world

(8080 refers to the port onf the machine and 80 refers to the port inside of the container)

Deployment

New repository created in docker hub

cmd

docker images

docker tag hello-world andrewdilley494/hello-world

Hub

docker push andrewdilley494/hello-world

Heroku

docker build --tag chatbot8 .

docker run -p 5000:5000 --name chatbot8 -d chatbot8

docker ps -a

docker rm chatbot8

docker build -t chatbot8

git init

git add .

git commit -m "Updated chatbot using WW policy documents"

git remote add origin https://github.com/AndrewDilley/chatbot.git

git push -u origin master

OPENAI\_API\_KEY=sk-proj-aM3UrXvPa4siuqVuQCxUT5B8n\_F23Fl-9zk\_MHn6kEHlcFFKWXjmwQGWcBVkXKg4\_K0sDCd5giT3BlbkFJbsfKTNbGh9IfOldBqSSF6bAU3g7xj0iJiNCIIcrzeywe-L8UjrF0NOOrmLkpQ4OfPkOZjY0AwA

FLASK\_SECRET\_KEY=2ffc1c8235fc9e3a91d86311ef2fc1d8a370008211186f390b215d94dc7df325

FILES = ['Alcohol and Drugs in the Workplace Procedure.DOCX',

         'Consequence Of Employee Misconduct.DOCX',

         'Contractor Management Procedure.DOCX',

         'Cyber Security Incident Response Plan Framework.DOCX',

         'Flexible Working Arrangements Procedure.DOCX',

         'Gifts Benefits and Hospitality Policy - BOARD.DOCX',

         'Hazard Reporting Procedure.DOCX',

         'Incident Reporting and Response Procedure.DOCX',

         'Information Technology Security Procedure.DOCX',

         'Mobile Phone Procedure.DOCX',

         'Motor Vehicle Operational Procedure.DOCX',

         'Personal Protective Equipment and Field Uniform.DOCX',

         'Physical Security Policy.docx',

         'Use of text based Generative Artificial Intelligence (AI).DOCX',

         'Vehicle Logbook Procedure.DOCX',

         'Vehicle Safety System Alarm Procedure.DOCX',

         'Vehicle Safety System Manual.DOCX',

         'Zero Harm Policy.DOCX']

SHAREPOINT\_LINKS = {

"Alcohol and Drugs in the Workplace Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/508",

"Consequence Of Employee Misconduct.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/286",

"Contractor Management Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/417",

"Cyber Security Incident Response Plan Framework.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/885",

"Flexible Working Arrangements Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/640",

"Gifts Benefits and Hospitality Policy - BOARD.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/822",

"Hazard Reporting Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/293",

"Incident Reporting and Response Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/665",

"Information Technology Security Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/815",

"Mobile Phone Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/896",

"Motor Vehicle Operational Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/240",

"Personal Protective Equipment and Field Uniform.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/230",

"Vehicle Logbook Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/1321",

"Physical Security Policy.docx": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/1355",

"Use of text based Generative Artificial Intelligence (AI).DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/1373",

"Vehicle Safety System Alarm Procedure.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/883",

"Vehicle Safety System Manual.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/1317",

"Zero Harm Policy.DOCX": "https://wannonwater.sharepoint.com/sites/cdms/SitePages/Homepage.aspx#/PublishedDocumentView/722"

# Add other file names and links here

}

FILES = list(SHAREPOINT\_LINKS.keys())

# Update the generate\_response function

def generate\_response(user\_input):

try:

relevant\_text, file\_name = search\_relevant\_text(user\_input)

# Build the prompt

prompt = f"Use the following document text to answer the question:\n\n{relevant\_text}\n\nQuestion: {user\_input}"

# Generate a response from OpenAI

completion = client.chat.completions.create(

model="gpt-4o-mini",

messages=[

{"role": "system", "content": "You are a helpful assistant."},

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

]

)

answer = completion.choices[0].message.content

# Add hyperlink to the reference

sharepoint\_link = SHAREPOINT\_LINKS.get(file\_name, "#")

formatted\_answer = (

f"{answer}<br><br>"

f"<span style='color:purple; font-weight:bold;'>Reference:</span> "

f"<a href='{sharepoint\_link}' target='\_blank'>{file\_name}</a>"

)

return formatted\_answer

except Exception as e:

return f"Error: {str(e)}"

**index**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Wannon Water Policy and Procedure Chatbot</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

</head>

<body>

<h1 class="title">Wannon Water Policy and Procedure Chatbot</h1>

<div class="chat-container">

<div id="chat-box">

<!-- Chat messages will appear here -->

</div>

<form id="chat-form">

<input type="text" id="user-input" placeholder="Type your message here..." autocomplete="off" required>

<button type="submit">Send</button>

</form>

</div>

<script src="{{ url\_for('static', filename='js/script.js') }}"></script>

</body>

</html>

**Style**

.chat-container {

width: 800px; /\* Doubled width \*/

background: white;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

overflow: hidden;

display: flex;

flex-direction: column;

}

#chat-box {

height: 600px; /\* Doubled height \*/

overflow-y: auto;

padding: 10px;

border-bottom: 1px solid #ddd;

}

.title {

text-align: center;

font-size: 2em;

color: #007bff;

margin-bottom: 20px;

}

git status

git add .

git commit -m "Updated chatbot10: linked SharePoint references"

git push origin master

chatbot11

PDF processing

Specifically the EBA

**Requirements.txt**

PyPDF2==3.0.1

**cmd**

pip install -r requirements.txt

python -m pip install setuptools

pip install --upgrade pip setuptools wheel

**dockerfile**

COPY requirements.txt /app/requirements.txt

RUN pip install -r /app/requirements.txt

**cmd**

docker build -t your-app-name .

docker run -p 5000:5000 your-app-name

pip uninstall PyPDF2

pip install PyPDF2==3.0.1

chatbot12

pre-processing the documents

should I create a new environment each time I make changes to the code?

No, you **don't need to create a new environment each time you make changes to your code**. Instead, you should:

1. **Use the Same Environment for Ongoing Development:**  
   Keep working within the same virtual environment as long as you’re making changes to the code. The environment serves as a sandbox for managing dependencies, and there’s no need to recreate it unless something major changes, such as:
   * Updating Python to a new version.
   * Major dependency changes that cause conflicts or issues.
   * The environment gets corrupted.
2. **Update the Environment When Needed:**  
   If you add new dependencies or update existing ones, install them in the current environment and update requirements.txt:
3. pip install <new-package>
4. pip freeze > requirements.txt
5. **Recreate the Environment Only When Necessary:**  
   Create a new environment if:
   * The current one becomes corrupted.
   * You switch to a new Python version.
   * You want to test a fresh setup (e.g., replicating the environment on a new machine).

**When to Create a New Environment**

* **New Project:**  
  Always create a fresh environment for new projects.
* **Major Updates:**  
  When testing a major version upgrade for Python or dependencies.
* **Corrupted Environment:**  
  If the current environment breaks, it's easier to recreate it than debug extensively.

**Good Practices**

1. **Regularly Update Dependencies:**  
   Keep your environment up-to-date with the latest versions of dependencies, ensuring compatibility and security.
2. pip list --outdated
3. pip install --upgrade <package-name>
4. **Use Version Control (Git):**  
   Keep your code changes in version control (e.g., Git). This separates code management from dependency management.
5. **Document Changes:**  
   If you add or update dependencies, make sure to document them in requirements.txt using:
6. pip freeze > requirements.txt
7. **Testing in a Fresh Environment:**  
   Occasionally test by recreating the environment from requirements.txt to ensure reproducibility:
8. python -m venv venv
9. .\venv\Scripts\activate # Activate the new environment
10. pip install -r requirements.txt

In summary, stick with your current environment for day-to-day development. Create a new environment only when testing, debugging, or starting something significantly new. Let me know if you need help managing environments! 😊

Notes:

Freezing the dependencies

The command:

bash

Copy code

pip freeze > requirements.txt

**What It Does**

This command generates a list of all the installed Python packages and their exact versions in the current environment and saves it to a file named requirements.txt.

1. **Update Regularly:**  
   Whenever you install or upgrade packages, update requirements.txt:

bash

Copy code

pip freeze > requirements.txt

1. **Avoid Unnecessary Packages:**  
   Clean up unused packages before freezing dependencies:

bash

Copy code

pip uninstall <unnecessary-package>

1. **Use a Virtual Environment:**  
   Always run this command within a virtual environment to ensure only project-specific dependencies are captured.

Check for outdated packages

The command:

pip list --outdated

**What It Does**

This command lists all the Python packages in your current environment that have newer versions available on the Python Package Index (PyPI).

**Detailed Breakdown**

1. **Installed Packages:**  
   The command examines the currently installed packages in your environment.
2. **Checks for Updates:**  
   It queries PyPI to see if newer versions are available for any installed packages.
3. **Outputs Information:**  
   The output includes:
   * The installed version of the package.
   * The latest available version.
   * The source of the installation (e.g., PyPI).

**Example Output**

Package Version Latest Type

---------- --------- --------- -----

Flask 2.1.2 2.2.2 sdist

numpy 1.24.2 1.24.3 wheel

requests 2.28.1 2.30.0 wheel

**What Each Column Means**

* **Package:** The name of the installed package.
* **Version:** The currently installed version.
* **Latest:** The newest version available on PyPI.
* **Type:** The distribution type (e.g., sdist for source distribution, wheel for binary distribution).

**Why Use It?**

1. **Keep Dependencies Up-to-Date:**
   * Identifies packages that have newer versions available to stay current with bug fixes, security patches, and new features.
2. **Resolve Compatibility Issues:**
   * If you're facing issues with a package, upgrading to the latest version might resolve them.
3. **Proactively Maintain Your Environment:**
   * Reduces technical debt by updating dependencies regularly.

**How to Update Packages**

Once you identify outdated packages, you can update them using:

**Single Package Update**

pip install --upgrade <package-name>

Example:

pip install --upgrade Flask

**Update All Outdated Packages**

To update all outdated packages at once:

pip list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip install -U

Note: Use this carefully, as it updates all packages and may introduce breaking changes.

**Best Practices**

1. **Test Updates:**  
   Before updating packages, test them in a staging environment to avoid breaking your application.
2. **Update Regularly:**  
   Run pip list --outdated periodically to ensure you’re aware of new versions.
3. **Freeze Updates:**  
   After updating, update your requirements.txt file:
4. pip freeze > requirements.txt

.reference {

    text-align: left;

    font-style: italic; /\* Italicized for distinction \*/

    margin-top: 5px;

    font-size: 0.9em; /\* Slightly smaller font \*/

}

.reference .reference-label {

    color: #333; /\* Dark gray for label \*/

    font-weight: normal;

}

.reference .reference-text {

    color: #333; /\* Forest green for data \*/

}

/\* Ensure Content enquiries label and text are styled consistently with reference \*/

.reference .content-enquiries-label {

    color: #333; /\* Match reference label color \*/

    font-weight: normal;

}

.reference .content-enquiries-text {

    color: #333; /\* Match reference text color \*/

}

Got it! To adapt the setup so the **shared volume for JSON files** is /preprocessed\_data and the **documents** are in /documents, we’ll ensure that these directories are properly mounted and used in the containers.

Here’s how the updated configuration will look:

**1. Updated preprocess\_documents.py Docker Setup**

**1.1 Updated Dockerfile**

# Dockerfile for preprocess\_documents

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the required files

COPY preprocess\_documents.py requirements.txt /app/

COPY documents /documents # Copy the documents folder to the container

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Define the shared volume

VOLUME /preprocessed\_data

# Run the script

CMD ["python", "preprocess\_documents.py", "--output-dir", "/preprocessed\_data", "--documents-dir", "/documents"]

**1.2 Command-Line Arguments in preprocess\_documents.py**

Modify preprocess\_documents.py to accept command-line arguments for directories:

import argparse

# Parse command-line arguments

parser = argparse.ArgumentParser(description="Preprocess documents to generate JSON and FAISS files.")

parser.add\_argument("--output-dir", type=str, required=True, help="Directory to save preprocessed files.")

parser.add\_argument("--documents-dir", type=str, required=True, help="Directory containing documents.")

args = parser.parse\_args()

# Use args.output\_dir and args.documents\_dir in your script logic

output\_dir = args.output\_dir

documents\_dir = args.documents\_dir

**1.3 Build and Run the Preprocessing Container**

1. Build the Docker image:
2. docker build -t preprocess-documents -f Dockerfile.preprocess .
3. Run the container:
4. docker run -v $(pwd)/preprocessed\_data:/preprocessed\_data -v $(pwd)/documents:/documents preprocess-documents

**2. Updated app.py Docker Setup**

**2.1 Updated Dockerfile**

# Dockerfile for app.py

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the required files

COPY app.py requirements.txt /app/

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Define the shared volume

VOLUME /preprocessed\_data

# Expose the application port

EXPOSE 5000

# Start the application

CMD ["python", "app.py", "--data-dir", "/preprocessed\_data"]

**2.2 Command-Line Arguments in app.py**

Modify app.py to accept a command-line argument for the data directory:

import argparse

# Parse command-line arguments

parser = argparse.ArgumentParser(description="Run the chatbot application.")

parser.add\_argument("--data-dir", type=str, required=True, help="Directory containing preprocessed files.")

args = parser.parse\_args()

# Use args.data\_dir in your app logic

data\_dir = args.data\_dir

**2.3 Build and Run the Application Container**

1. Build the Docker image:
2. docker build -t chatbot-app -f Dockerfile.app .
3. Run the container:
4. docker run -v $(pwd)/preprocessed\_data:/preprocessed\_data -p 5000:5000 chatbot-app

**3. Updated docker-compose.yml**

To orchestrate the two containers with shared volumes for /preprocessed\_data and /documents:

version: "3.8"

services:

preprocess:

build:

context: .

dockerfile: Dockerfile.preprocess

volumes:

- ./preprocessed\_data:/preprocessed\_data

- ./documents:/documents

command: ["python", "preprocess\_documents.py", "--output-dir", "/preprocessed\_data", "--documents-dir", "/documents"]

app:

build:

context: .

dockerfile: Dockerfile.app

volumes:

- ./preprocessed\_data:/preprocessed\_data

ports:

- "5000:5000"

command: ["python", "app.py", "--data-dir", "/preprocessed\_data"]

depends\_on:

- preprocess

**4. Directory Structure**

Ensure your project has the following structure:

/project

├── app.py

├── preprocess\_documents.py

├── requirements.txt

├── Dockerfile.preprocess

├── Dockerfile.app

├── docker-compose.yml

├── /documents

│ └── (Your input documents here)

└── /preprocessed\_data

└── (Generated files like text\_map.json and faiss\_index.bin)

**5. Build and Run with Docker Compose**

1. Build and run the containers:
2. docker-compose up --build
3. Check the logs to ensure:
   * preprocess\_documents.py runs and generates the files in /preprocessed\_data.
   * app.py successfully loads these files.

**6. Verify in Azure**

* **Push Docker Images**: Follow the steps outlined earlier to push images to Azure Container Registry.
* **Persistent Volume Setup**:
  + Create persistent volumes in Azure Kubernetes Service (AKS) for /preprocessed\_data and /documents.
  + Mount these volumes in both containers to ensure data sharing.

Let me know if you need assistance with the deployment to Azure or further clarification! 😊

docker run -e OPENAI\_API\_KEY="sk-proj-HdOgtUz\_C-suiU8igWdSLhhH8mpqjr4pnra3fOt7qCBBlwB9gKdeiu\_oLqZTPJTzkOW7PRVSxhT3BlbkFJu1mj9TlIDIctsI\_wC5s5hj2bMLqLSxZL1WJGtaW\_rC7omH4iVUwbUjTu7MSlHzE5kvUAhKFEIA" \

-v C:/Users/andrew.dilley/development/chatbot12/preprocessed\_data:/preprocessed\_data \

-v C:/Users/andrew.dilley/development/chatbot12/documents:/documents \

preprocess-documents

docker run -p 5000:5000 --name chatbot8 -d chatbot8

preprocess-documents

chatbot-app

docker images

docker ps -a

docker rm chatbot8

docker rm preprocess-documents

docker build --no-cache -t preprocess-documents -f Dockerfile.preprocess .

docker images

docker run -p 5000:5000 --name preprocess-documents -d preprocess-documents

docker ps -a

docker logs preprocess-documents

docker build --no-cache -t chatbot-app -f Dockerfile.app .

rm chatbot-app

run -p 5000:5000 --name chatbot-app -d chatbot-app

docker logs chatbot-app

<https://docs.docker.com/engine/storage/volumes/#use-a-volume-with-docker-compose>

**Create Volumes**

Run the following commands to create the Docker volumes:

docker volume create documents\_volume

docker volume create preprocessed\_data\_volume

**Verify Output**: Inspect the preprocessed\_data\_volume volume:

docker run --rm -v preprocessed\_data\_volume:/data busybox ls /data

**Populate the documents\_volume**

Copy your documents into the documents\_volume using the following command (update the path to your actual files):

docker run --rm -v documents\_volume:/data -v C:\Users\andrew.dilley\development\chatbot12\documents:/host\_data busybox sh -c "cp -r /host\_data/\* /data/"

**Inspect the volume contents**

docker run --rm -v documents\_volume:/data busybox ls /data

**Build the Docker Image**

Build the Docker image with the name preprocess-documents:

docker build --no-cache -t preprocess-documents -f Dockerfile.preprocess .

1 warning found (use docker --debug to expand):

- SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV "OPENAI\_API\_KEY") (line 8)

**Check the image was created**

docker images

**run the container**

docker run --rm -v documents\_volume:/documents -v preprocessed\_data\_volume:/preprocessed\_data preprocess-documents

**Verify the Output**

After the container has run, you can inspect the preprocessed\_data\_volume to ensure the output files were generated:

docker run --rm -v preprocessed\_data\_volume:/data busybox ls /data

if os.getenv("DOCKER\_ENV") == "true":

DOCUMENTS\_PATH = "/app/documents"

PREPROCESSED\_PATH = "/app/preprocessed\_data"

else:

DOCUMENTS\_PATH = "C:/Users/andrew.dilley/development/chatbot12/documents"

PREPROCESSED\_PATH = "C:/Users/andrew.dilley/development/chatbot12/preprocessed\_data"

 **Run the Container with Interactive Shell**:

bash

Copy code

docker run --rm -it -v preprocessed\_data\_volume:/app/preprocessed\_data preprocess-documents sh

 **List Files with Timestamps**: Inside the container, navigate to the directory and use ls -l:

sh

Copy code

ls -l /app/preprocessed\_data

(venv) PS C:\Users\andrew.dilley\development\chatbot12> docker run --rm -it -v preprocessed\_data\_volume:/app/preprocessed\_data preprocess-documents sh

# ls -l /app/preprocessed\_data

total 552

-rw-r--r-- 1 root root 116781 Jan 14 06:55 faiss\_index.bin

-rw-r--r-- 1 root root 18 Jan 14 07:02 test\_file.txt

-rw-r--r-- 1 root root 441620 Jan 14 06:55 text\_map.json

**2. Debug the Script Execution**

Run the container interactively and manually check if the output directory /app/preprocessed\_data has any files:

docker run --rm -it -v documents\_volume:/app/documents -v preprocessed\_data\_volume:/app/preprocessed\_data preprocess-documents sh

at the # prompt (ie inside the container

**Check Mounted Volumes**:

* Verify the contents of the documents\_volume (mounted to /app/documents):

ls /app/documents

This should display the files in your documents\_volume.

* Yes it does!
* 

Verify the preprocessed\_data\_volume (mounted to /app/preprocessed\_data):

ls /app/preprocessed\_data

* Nothing returnedf

**Test Writing to the Preprocessed Directory**: Check if you can write files to the preprocessed\_data\_volume:

echo "test file content" > /app/preprocessed\_data/test\_file.txt

ls /app/preprocessed\_data

If this works, the volume is mounted and writable.

* This worked
* **Run Your Script Manually**: Execute your script inside the container to debug:
* sh
* Copy code
* python preprocess\_documents.py

docker build --no-cache -t app -f Dockerfile.app .

docker run --rm -p 80:80 app