

Automating Dockerfile Generation with Ollama API

Introduction

- Manually writing Dockerfiles for different programming languages can be time-consuming and error-prone. So, I built a Python script that automatically generates a best-practice Dockerfile using the Ollama API powered by the TinyLlama model. 🔥
- This project takes a programming language as input and generates a custom Dockerfile with comments, saving it for later use.

Step-by-Step Guide Outline

1 Install Dependencies

- Install Python and required libraries (requests).
- Install and set up Ollama.

2 Create Project Structure

- Set up project folder with main.py, config.py, and dockerfiles/.

3 Configure API

- Define OLLAMA_API_URL and MODEL_NAME in config.py.

4 Write Python Script

- Accept programming language as input.
- Send request to TinyLlama model via Ollama API.
- Receive and save the generated Dockerfile.

5 Run the Script

- Start Ollama API (ollama serve).
- Execute python main.py.
- Enter programming language.

6 Check Output

- View generated Dockerfile in dockerfiles/ folder.

7 Troubleshooting & Enhancements

- Fix connection errors by ensuring Ollama is running.

Tools Used

1 Windows Laptop

- The project was developed and tested on a Windows laptop for easy local execution.

2 ChatGPT

- Used for brainstorming ideas, debugging, and improving code efficiency.

3 Ollama (TinyLlama Model)

- A local LLM (Large Language Model) for generating Dockerfiles.

4 Python

- Core programming language used for scripting and API communication.

Why Use a Local LLM?

Local LLM vs. Hosted LLM		
Feature	Local LLM (TinyLlama)	Hosted LLM (API-based)
Latency	Faster, no internet dependency	May experience network delays
Privacy	Fully private, runs on own device	Data is sent to external servers
Cost	One-time setup, no API costs	Often requires paid API access
Customization	Can fine-tune models locally	Limited customization

Advantages of Local LLMs

- ✓ No Internet Required – Works offline.
- ✓ No API Costs – No need to pay for API usage.
- ✓ Better Data Privacy – Keeps all processing on the local machine.

Disadvantages of Local LLMs

- ✗ Hardware Dependent – Requires a good CPU/GPU.
- ✗ Limited Model Size – Cannot use very large models.
- ✗ Manual Setup – Needs installation and configuration.

This project leverages a local LLM to ensure fast, cost-effective, and private Dockerfile generation without relying on external APIs. 🚀

Step 1: Install Required Software

Open CMD and check Python and Docker are installed correctly and working or not
By typing

1. python --version
2. docker --version

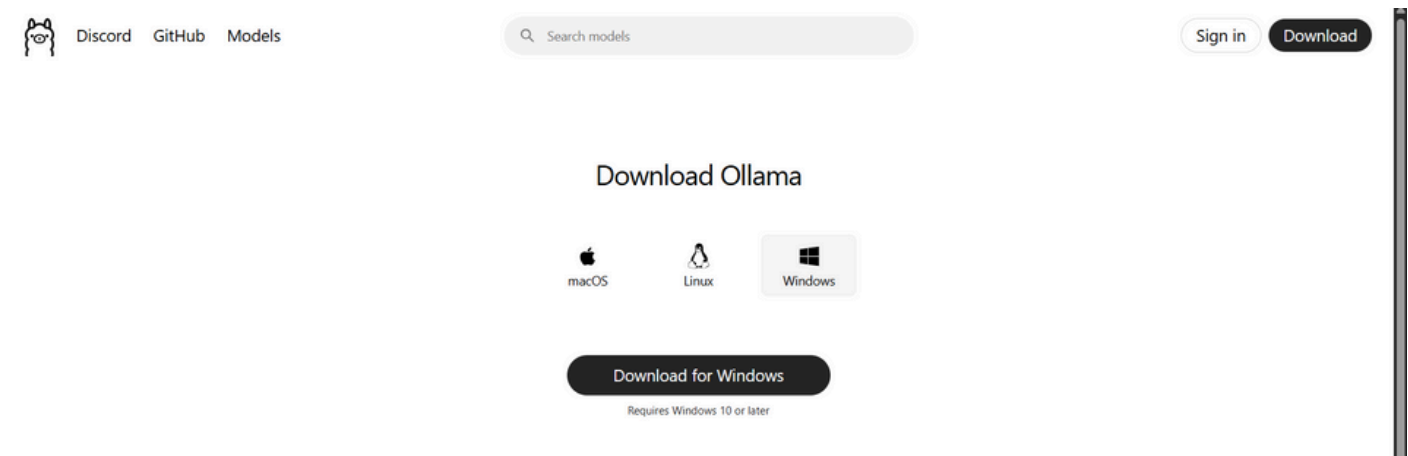
```
Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>docker --version
Docker version 27.5.1, build 9f9e405

C:\Users\Admin>python --version
Python 3.13.2

C:\Users\Admin>|
```

Download Ollama windows version from the its website and install it



Check if its correctly instaled or not by typing “ollama” on cmd. and check version by “ollama --version”

```
Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>ollama
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve    Start ollama
  create   Create a model from a Modelfile
  show     Show information for a model
  run      Run a model
  stop     Stop a running model
  pull     Pull a model from a registry
  push     Push a model to a registry
  list     List models
  ps       List running models
  cp       Copy a model
  rm       Remove a model
  help     Help about any command

Flags:
  -h, --help    help for ollama
  -v, --version  Show version information

Use "ollama [command] --help" for more information about a command.

C:\Users\Admin>ollama -v
ollama version is 0.6.2

C:\Users\Admin>|
```

Step 2: Setup Your Project Folder

Create a new folder for your project by typing the below commands

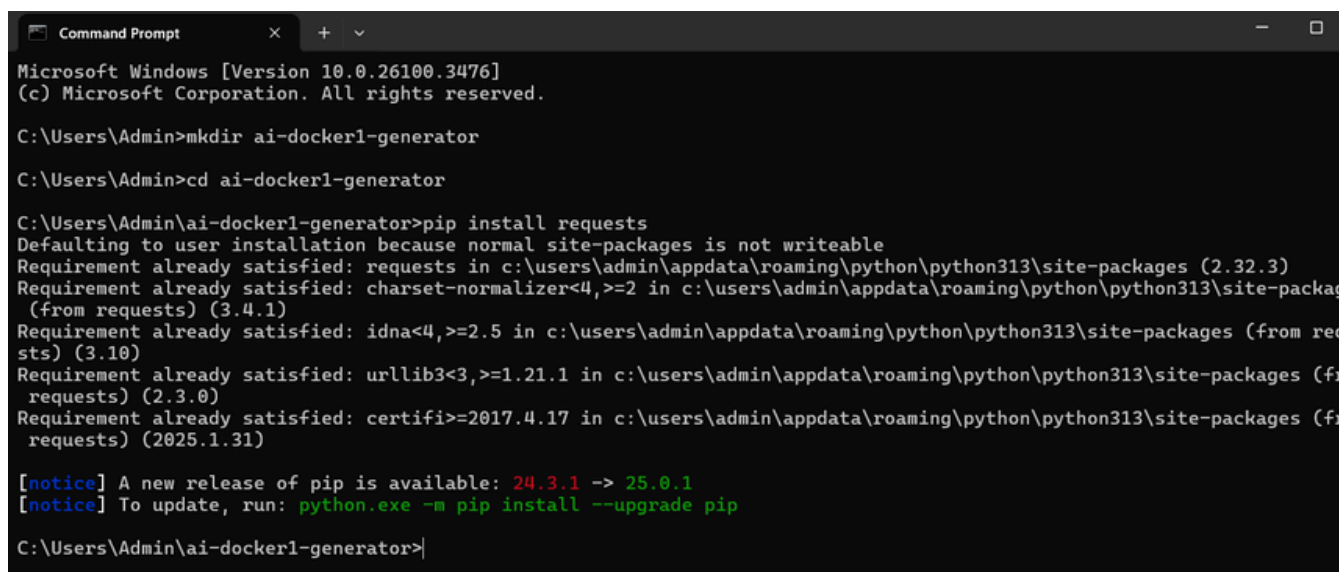
```
"mkdir ai-docker-generator"
```

```
"cd ai-docker1-generator"
```

Step 3: Install Python Dependencies

Inside the project folder, install required libraries:

```
"pip install requests"
```



```
Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>mkdir ai-docker1-generator

C:\Users\Admin>cd ai-docker1-generator

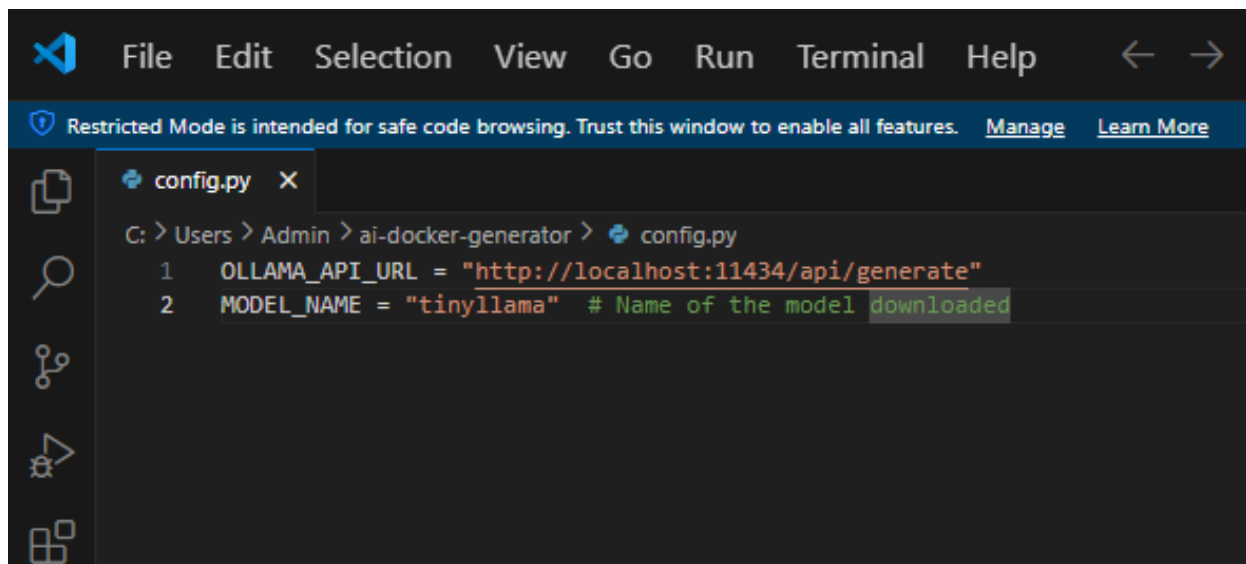
C:\Users\Admin\ai-docker1-generator>pip install requests
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: requests in c:\users\admin\appdata\roaming\python\python313\site-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\admin\appdata\roaming\python\python313\site-packages (from requests) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\admin\appdata\roaming\python\python313\site-packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\admin\appdata\roaming\python\python313\site-packages (from requests) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\admin\appdata\roaming\python\python313\site-packages (from requests) (2025.1.31)

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Admin\ai-docker1-generator>
```

Step 4: Create Configuration File

Inside your project folder, create a file named config.py and add the Ollama API URL:

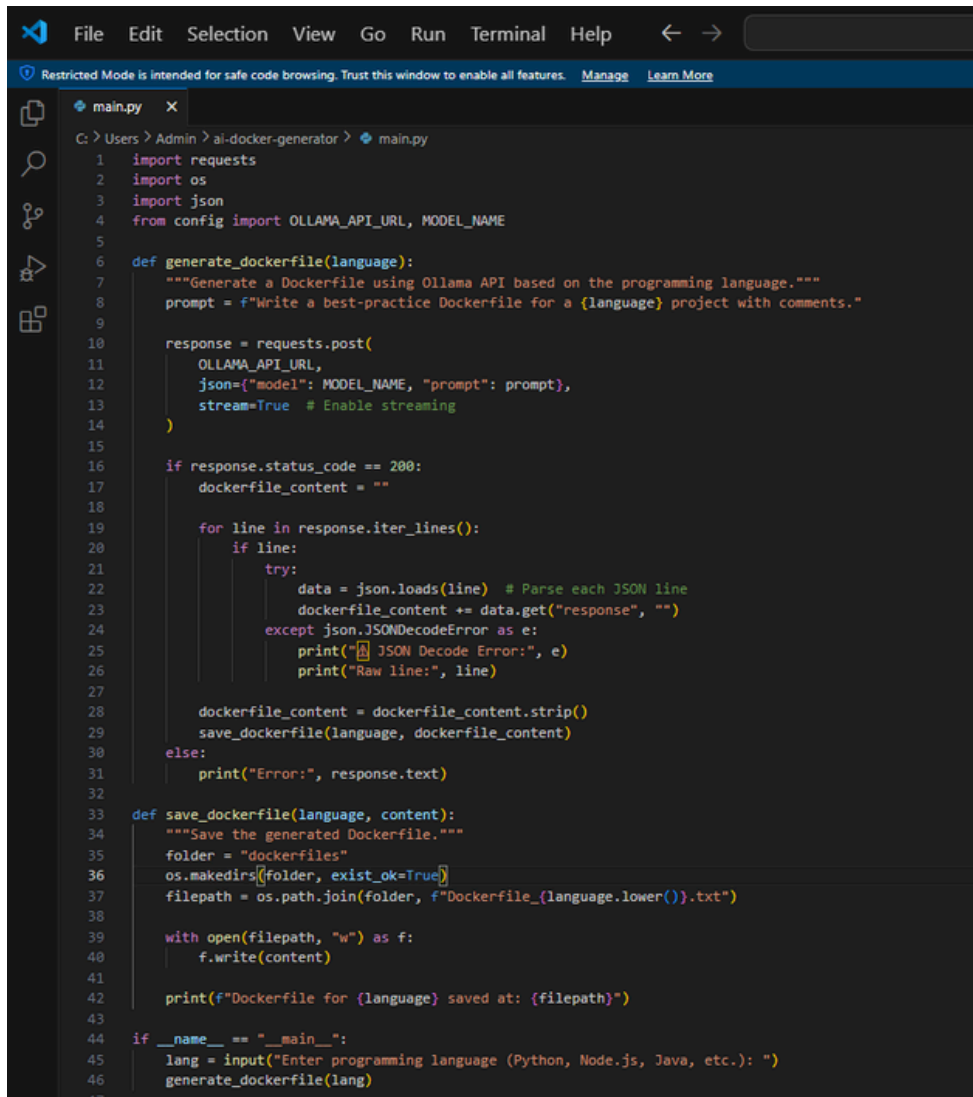


```
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

config.py
C: > Users > Admin > ai-docker-generator > config.py
1 OLLAMA_API_URL = "http://localhost:11434/api/generate"
2 MODEL_NAME = "tinyllama" # Name of the model downloaded
```

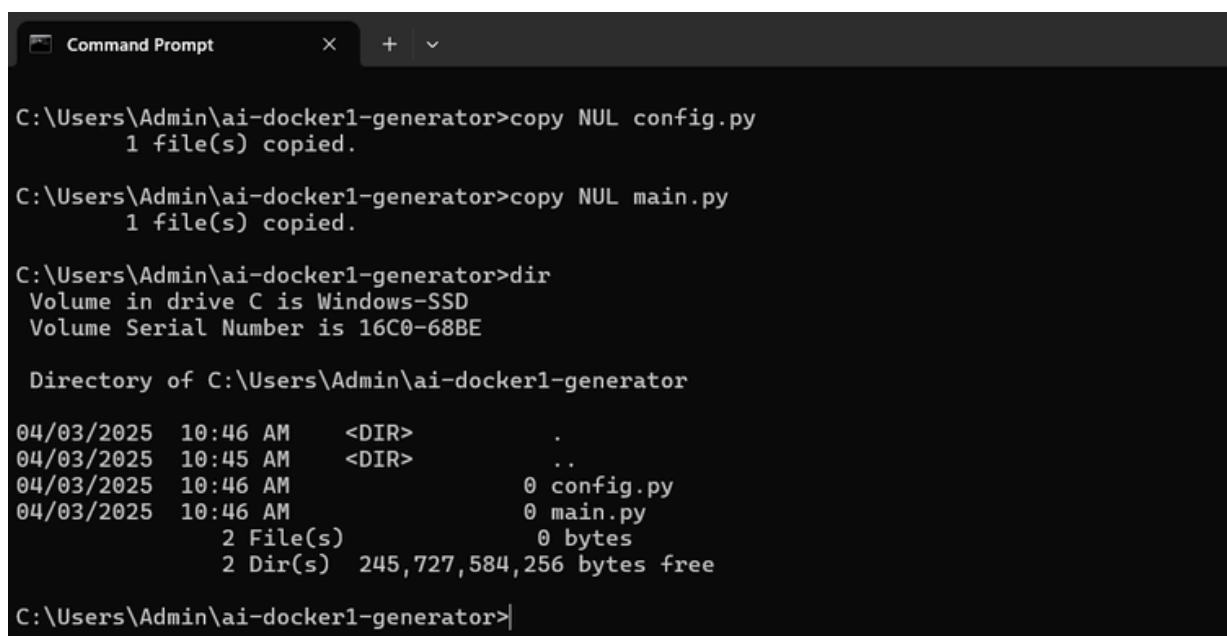
Step 5: Create the Main Script

Create a file main.py inside the folder.

A screenshot of a code editor window titled 'main.py'. The editor shows a Python script for generating Dockerfiles using the Ollama API. The script includes imports for requests, os, and json, and a config module for OLLAMA_API_URL and MODEL_NAME. It defines two functions: generate_dockerfile(language) and save_dockerfile(language, content). The generate_dockerfile function sends a POST request to the Ollama API with a prompt to generate a Dockerfile for a specific language. It then iterates through the response lines, parsing JSON to extract the Dockerfile content. The save_dockerfile function creates a 'dockerfiles' folder, constructs a filepath, and writes the generated Dockerfile content to the file. The script concludes with a main block that prompts the user for a programming language and calls the generate_dockerfile function.

```
1 import requests
2 import os
3 import json
4 from config import OLLAMA_API_URL, MODEL_NAME
5
6 def generate_dockerfile(language):
7     """Generate a Dockerfile using Ollama API based on the programming language."""
8     prompt = f"Write a best-practice Dockerfile for a {language} project with comments."
9
10    response = requests.post(
11        OLLAMA_API_URL,
12        json={"model": MODEL_NAME, "prompt": prompt},
13        stream=True # Enable streaming
14    )
15
16    if response.status_code == 200:
17        dockerfile_content = ""
18
19        for line in response.iter_lines():
20            if line:
21                try:
22                    data = json.loads(line) # Parse each JSON line
23                    dockerfile_content += data.get("response", "")
24                except json.JSONDecodeError as e:
25                    print(f"JSON Decode Error:", e)
26                    print("Raw line:", line)
27
28        dockerfile_content = dockerfile_content.strip()
29        save_dockerfile(language, dockerfile_content)
30    else:
31        print("Error:", response.text)
32
33 def save_dockerfile(language, content):
34     """Save the generated Dockerfile."""
35     folder = "dockerfiles"
36     os.makedirs(folder, exist_ok=True)
37     filepath = os.path.join(folder, f"Dockerfile_{language.lower()}.txt")
38
39     with open(filepath, "w") as f:
40         f.write(content)
41
42     print(f"Dockerfile for {language} saved at: {filepath}")
43
44 if __name__ == "__main__":
45     lang = input("Enter programming language (Python, Node.js, Java, etc.): ")
46     generate_dockerfile(lang)
47
```

Here we can see both the file are present in the folder ai-docker1-generator

A screenshot of a Windows Command Prompt window. The user has navigated to the directory C:\Users\Admin\ai-docker1-generator. They have executed two 'copy' commands to copy 'NUL config.py' and 'NUL main.py' into the directory, both of which were successful. Then, they executed the 'dir' command, which displayed the directory's contents. The output shows two files, config.py and main.py, each 0 bytes in size, and two subdirectories, . and .., each 0 bytes in size. The total free space in the directory is 245,727,584,256 bytes.

```
C:\Users\Admin\ai-docker1-generator>copy NUL config.py
1 file(s) copied.

C:\Users\Admin\ai-docker1-generator>copy NUL main.py
1 file(s) copied.

C:\Users\Admin\ai-docker1-generator>dir
Volume in drive C is Windows-SSD
Volume Serial Number is 16C0-68BE

Directory of C:\Users\Admin\ai-docker1-generator

04/03/2025  10:46 AM    <DIR>          .
04/03/2025  10:45 AM    <DIR>          ..
04/03/2025  10:46 AM                0 config.py
04/03/2025  10:46 AM                0 main.py
                2 File(s)                0 bytes
                2 Dir(s)  245,727,584,256 bytes free

C:\Users\Admin\ai-docker1-generator>
```

Step 6: Pull a Ollama Model and start Ollama API server

For my project I have downloaded Ollama Model - “Tinyllama” Since it requires less ram for the process and then start the ollama api server

“ollama pull tinyllama”

“ollama serve”

since i have alredy downloaded i can check my model by typing “ollama list”

```
Command Prompt

C:\Users\Admin\ai-docker1-generator>ollama list
NAME                ID                SIZE      MODIFIED
tinyllama:latest    2644915ede35      637 MB    14 hours ago

C:\Users\Admin\ai-docker1-generator>|
```

After “ollama serve” command we can see the api server begins

```
C:\Users\Admin\ai-docker1-generator>ollama serve
2025/04/03 10:58:45 routes.go:1230: INFO server config env="map[CUDA_VISIBLE_DEVICES: GPU_DEVICE_ORDINAL: HIP_VISIBLE_DEVICES: HSA_OVERRIDE_GFX_VERSION: HTTP_PROXY: HTTP_PROXY: NO_PROXY: OLLAMA_CONTEXT_LENGTH:2048 OLLAMA_DEBUG:false OLLAMA_FLASH_ATTENTION:false OLLAMA_GPU_OVERHEAD:0 OLLAMA_HOST:http://127.0.0.1:11434 OLLAMA_INTEL_GPU:false OLLAMA_KEEP_ALIVE:5m0s OLLAMA_KV_CACHE_TYPE: OLLAMA_LLM_LIBRARY: OLLAMA_LOAD_TIMEOUT:5m0s OLLAMA_MAX_LOADED_MODELS:0 OLLAMA_MAX_QUEUE:512 OLLAMA_MODELS:C:\Users\Admin\ai-docker1-generator\models OLLAMA_MULTIUSER_CACHE:false OLLAMA_NEW_ENGINE:false OLLAMA_NOHISTORY:false OLLAMA_NOPRUNE:false OLLAMA_NUM_PARALLEL:0 OLLAMA_ORIGINS:[http://localhost https://localhost http://localhost:* https://localhost:* http://127.0.0.1 https://127.0.0.1 http://127.0.0.1:* https://127.0.0.1:* http://0.0.0.0 https://0.0.0.0 http://0.0.0.0:* https://0.0.0.0:* app://* file://* tauri://* vscode-webview://* vscode-file://*] OLLAMA_SCHED_SPREAD:false ROCR_VISIBLE_DEVICES:]"
time=2025-04-03T10:58:45.602+05:30 level=INFO source=images.go:432 msg="total blobs: 5"
time=2025-04-03T10:58:45.602+05:30 level=INFO source=images.go:439 msg="total unused blobs removed: 0"
time=2025-04-03T10:58:45.603+05:30 level=INFO source=routes.go:1297 msg="Listening on 127.0.0.1:11434 (version 0.6.2)"
time=2025-04-03T10:58:45.604+05:30 level=INFO source=gpu.go:217 msg="looking for compatible GPUs"
time=2025-04-03T10:58:45.604+05:30 level=INFO source=gpu_windows.go:167 msg="packages count=1"
time=2025-04-03T10:58:45.604+05:30 level=INFO source=gpu_windows.go:214 msg="" package=0 cores=4 efficiency=0 threads=8
time=2025-04-03T10:58:45.612+05:30 level=INFO source=gpu.go:377 msg="no compatible GPUs were discovered"
time=2025-04-03T10:58:45.614+05:30 level=INFO source=types.go:130 msg="inference compute" id=0 library=ccpu variant="" compute="" driver=0.0 name="" total="7.8 GiB" available="1.3 GiB"
[GIN] 2025/04/03 - 11:01:06 | 200 | 0s | 127.0.0.1 | HEAD | "/"
[GIN] 2025/04/03 - 11:01:06 | 200 | 1.2046ms | 127.0.0.1 | GET | "/api/tags"
```

check weather the model is working and repoding properly or not by simply asking a question

“ollama run tinyllama "Answer in one sentence: What is AI?"

```
C:\Users\Admin\ai-docker-generator>ollama run tinyllama "Answer in one sentence: What is AI?"
AI stands for Artificial Intelligence, and it's a complex set of technologies designed to allow machines or software to emulate the cognitive abilities of human beings. This includes things like problem-solving, decision making, and learning from experience. AI is often used in areas such as healthcare, finance, and manufacturing, but its applications are wide-ranging and varied depending on a particular industry's needs.

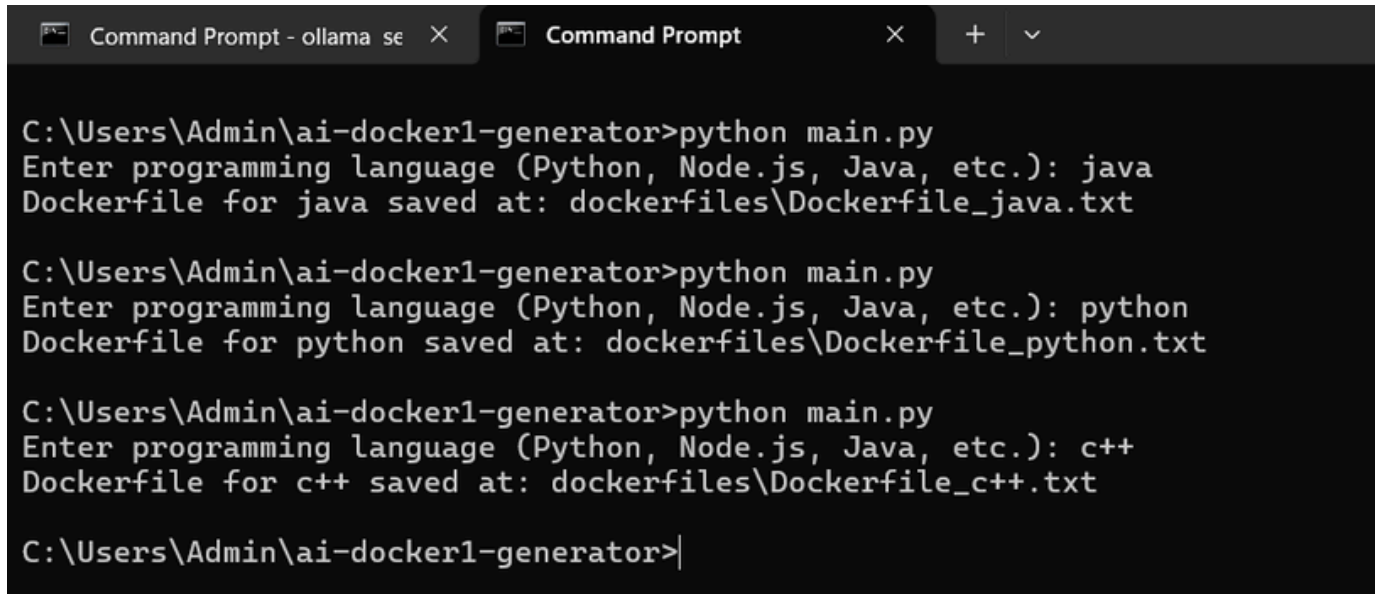
C:\Users\Admin\ai-docker-generator>|
```

Respond correctly. 👍

Step 7: Running the Script main.py

Run the script from the terminal:

“python main.py”



```
C:\Users\Admin\ai-docker1-generator>python main.py
Enter programming language (Python, Node.js, Java, etc.): java
Dockerfile for java saved at: dockerfiles\Dockerfile_java.txt

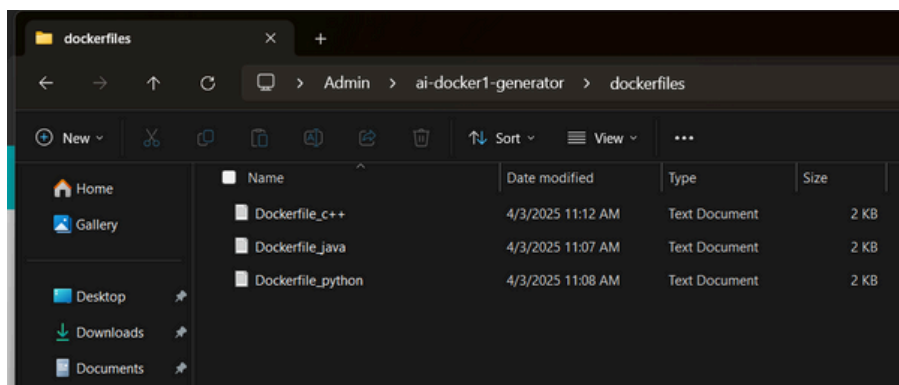
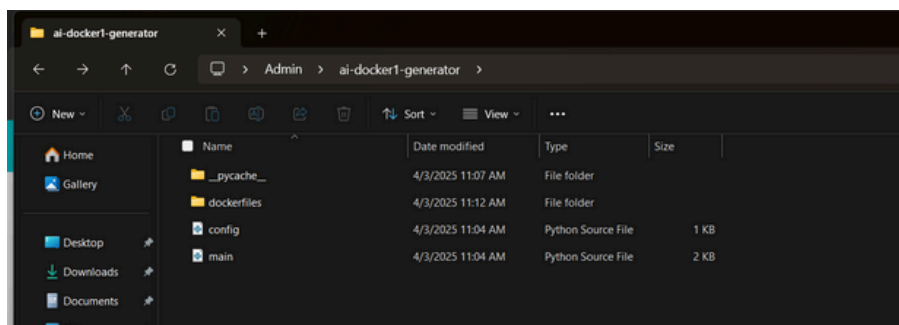
C:\Users\Admin\ai-docker1-generator>python main.py
Enter programming language (Python, Node.js, Java, etc.): python
Dockerfile for python saved at: dockerfiles\Dockerfile_python.txt

C:\Users\Admin\ai-docker1-generator>python main.py
Enter programming language (Python, Node.js, Java, etc.): c++
Dockerfile for c++ saved at: dockerfiles\Dockerfile_c++.txt

C:\Users\Admin\ai-docker1-generator>
```

After running the main file we get a question to choose a language to generate the docker file.

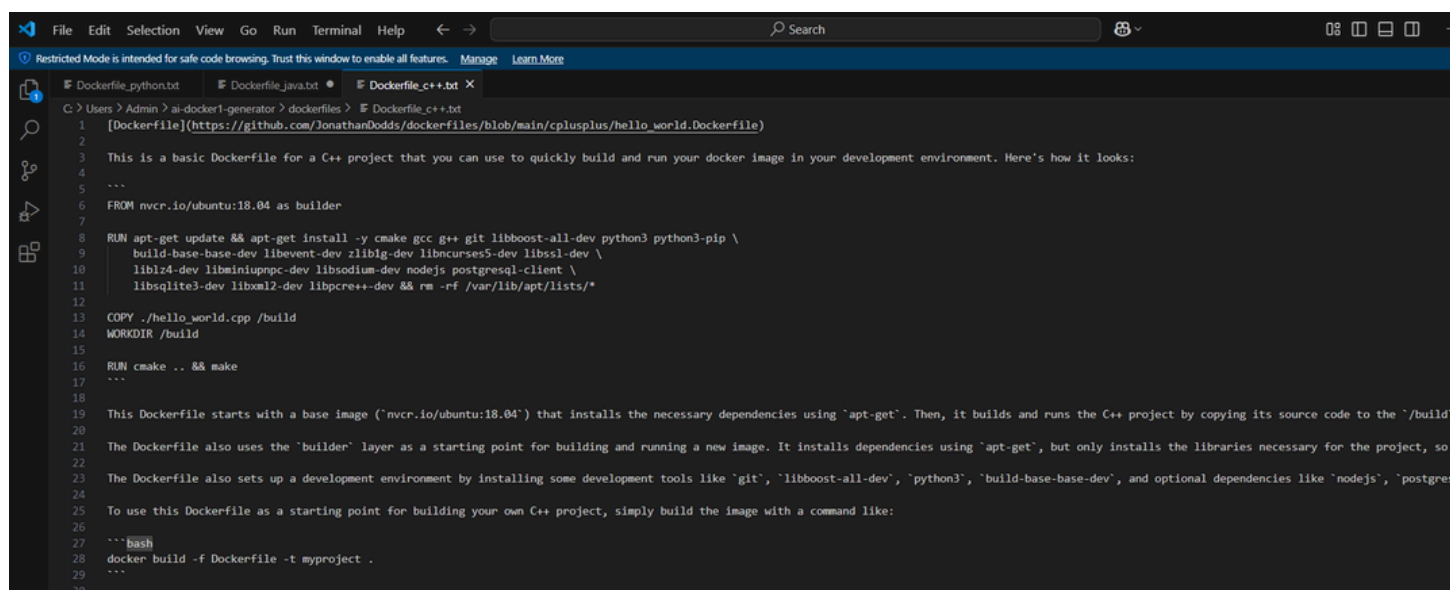
for reference I have done for Python, Java, C++ Languages



We can see that the model had automatically created a Docker folder and started adding files after creating them

Below are the Docker files created By Ollama Tinyllama Model

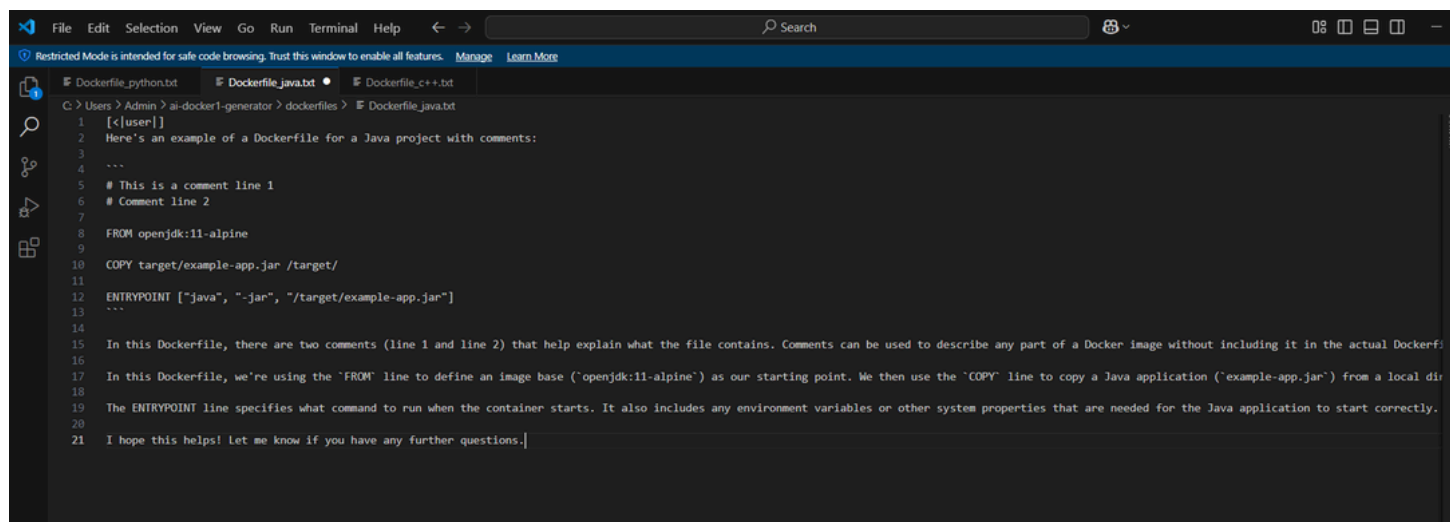
For C++



The screenshot shows a code editor with a Dockerfile for C++. The file is named 'Dockerfile_c++.txt'. The content of the Dockerfile is as follows:

```
1 [Dockerfile](https://github.com/JonathanDodds/dockerfiles/blob/main/cplusplus/hello_world.Dockerfile)
2
3 This is a basic Dockerfile for a C++ project that you can use to quickly build and run your docker image in your development environment. Here's how it looks:
4
5 ---
6 FROM nvcr.io/ubuntu:18.04 as builder
7
8 RUN apt-get update && apt-get install -y cmake gcc g++ git libboost-all-dev python3 python3-pip \
9     build-base-base-dev libevent-dev zlib1g-dev libncurses5-dev libssl-dev \
10     libbz2-dev libminiupnpc-dev libsodium-dev nodejs postgresql-client \
11     libsqlite3-dev libxml2-dev libpcre++-dev && rm -rf /var/lib/apt/lists/*
12
13 COPY ./hello_world.cpp /build
14 WORKDIR /build
15
16 RUN cmake .. && make
17
18 ---
19 This Dockerfile starts with a base image ('nvcr.io/ubuntu:18.04') that installs the necessary dependencies using 'apt-get'. Then, it builds and runs the C++ project by copying its source code to the '/build'
20
21 The Dockerfile also uses the 'builder' layer as a starting point for building and running a new image. It installs dependencies using 'apt-get', but only installs the libraries necessary for the project, so
22
23 The Dockerfile also sets up a development environment by installing some development tools like 'git', 'libboost-all-dev', 'python3', 'build-base-base-dev', and optional dependencies like 'nodejs', 'postgres
24
25 To use this Dockerfile as a starting point for building your own C++ project, simply build the image with a command like:
26
27 ---bash
28 docker build -f Dockerfile -t myproject .
29
30 ---
```

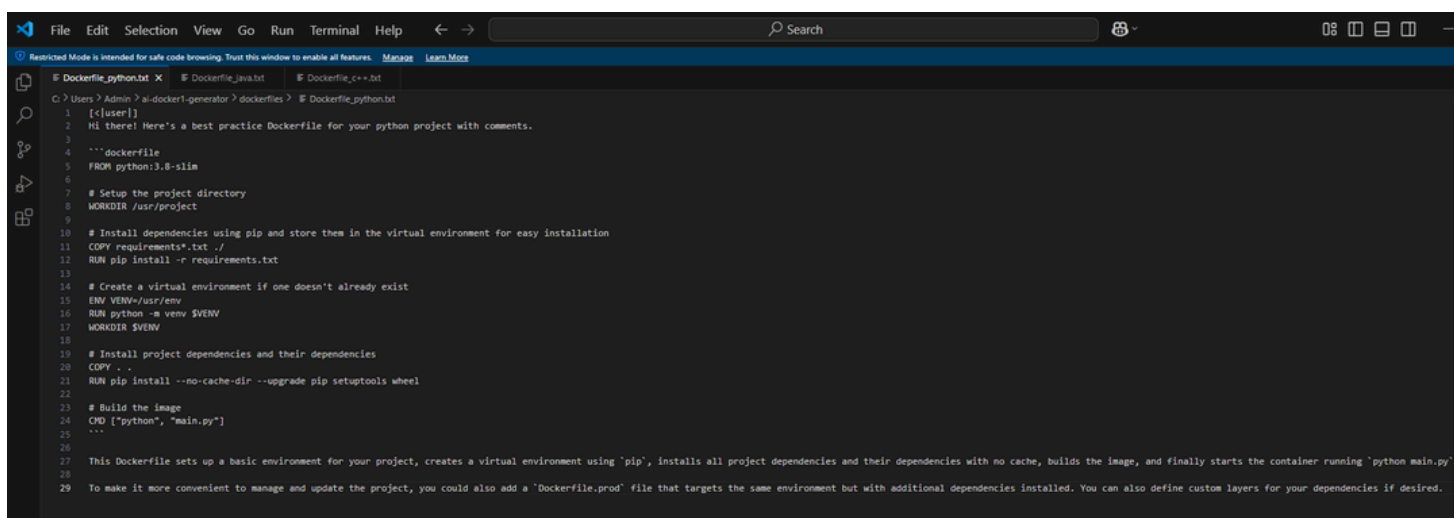
For JAVA



The screenshot shows a code editor with a Dockerfile for Java. The file is named 'Dockerfile_java.txt'. The content of the Dockerfile is as follows:

```
1 [user]
2 Here's an example of a Dockerfile for a Java project with comments:
3
4 ---
5 # This is a comment line 1
6 # Comment line 2
7
8 FROM openjdk:11-alpine
9
10 COPY target/example-app.jar /target/
11
12 ENTRYPOINT ["java", "-jar", "/target/example-app.jar"]
13
14 ---
15 In this Dockerfile, there are two comments (line 1 and line 2) that help explain what the file contains. Comments can be used to describe any part of a Docker image without including it in the actual Dockerfile.
16
17 In this Dockerfile, we're using the 'FROM' line to define an image base ('openjdk:11-alpine') as our starting point. We then use the 'COPY' line to copy a Java application ('example-app.jar') from a local dir
18
19 The ENTRYPOINT line specifies what command to run when the container starts. It also includes any environment variables or other system properties that are needed for the Java application to start correctly.
20
21 I hope this helps! Let me know if you have any further questions.
```

For Python



The screenshot shows a code editor with a Dockerfile for Python. The file is named 'Dockerfile_python.txt'. The content of the Dockerfile is as follows:

```
1 [user]
2 Hi there! Here's a best practice Dockerfile for your python project with comments.
3
4 ---dockerfile
5 FROM python:3.8-slim
6
7 # Setup the project directory
8 WORKDIR /usr/project
9
10 # Install dependencies using pip and store them in the virtual environment for easy installation
11 COPY requirements*.txt ./
12 RUN pip install -r requirements.txt
13
14 # Create a virtual environment if one doesn't already exist
15 ENV VENV=/usr/env
16 RUN python -m venv $VENV
17 WORKDIR $VENV
18
19 # Install project dependencies and their dependencies
20 COPY . .
21 RUN pip install --no-cache-dir --upgrade pip setuptools wheel
22
23 # Build the image
24 CMD ["python", "main.py"]
25
26 ---
27 This Dockerfile sets up a basic environment for your project, creates a virtual environment using 'pip', installs all project dependencies and their dependencies with no cache, builds the image, and finally starts the container running 'python main.py'
28
29 To make it more convenient to manage and update the project, you could also add a 'Dockerfile.prod' file that targets the same environment but with additional dependencies installed. You can also define custom layers for your dependencies if desired.
```


Conclusion

DevOps is all about automation, and integrating AI into the process makes automation even more efficient, reliable, and faster. By using Ollama's TinyLlama model, I was able to generate Dockerfiles dynamically, reducing manual effort and improving consistency in containerization.

While the generated Dockerfile may not be perfect, it can be significantly improved with better model training and fine-tuning. The potential of AI in DevOps automation is vast, and with continuous improvements, it can become an essential tool for streamlining infrastructure management.

For scripting, I leveraged ChatGPT to assist in writing the Python script that interacts with the Ollama API. While the project itself is simple, setting up Ollama, troubleshooting installation issues, and fine-tuning model responses required patience and problem-solving skills.

This project was a great learning experience, demonstrating how AI-powered automation can optimize DevOps workflows while also highlighting the challenges of working with local AI models. 🚀



OLLAMA