**Lab 6: CLI AI Assistant for Template Suggestion**

**🛠️ Step-by-Step Implementation**

**✅ Step 1: Install Dependencies**

You can use a venv or Conda. Then install:

pip install -U langchain langchain-community langchain-huggingface huggingface\_hub

Use openai or huggingface\_hub as LLM wrapper depending on source.

**✅ Step 2: Set Up LangChain LLM Wrapper (HuggingFace Free Model)**

We’ll use a free open-source LLM from HuggingFace (like HuggingFaceHub wrapper with mistralai/Mistral-7B-Instruct-v0.1).

Create a file: cli\_ai\_assistant.py

(venv) root@ip-172-31-14-172:~# cat assistant.py

from langchain\_huggingface import HuggingFaceEndpoint

from langchain\_core.prompts import PromptTemplate

import os

# Set your HF token (it must have access to the Inference endpoint)

os.environ["HUGGINGFACEHUB\_API\_TOKEN"] = "hf\_czNLRvUJBMgu"

# Initialize endpoint (NO use\_cache)

llm = HuggingFaceEndpoint(

repo\_id="mistralai/Mixtral-8x7B-Instruct-v0.1",

temperature=0.5,

max\_new\_tokens=300,

)

prompt = PromptTemplate(

input\_variables=["project\_description"],

template="""

You are a Platform Engineering expert. Given the following description:

"{project\_description}"

Suggest a suitable microservice starter template that includes:

- framework

- observability stack

- testing tools

- containerization (Docker)

Respond in a single paragraph.

"""

)

# Runnable chain

chain = prompt | llm

def main():

print("Developer Template Recommender Assistant")

while True:

user\_input = input("\nDescribe your project (or type 'exit'): ")

if user\_input.lower() == 'exit':

break

response = chain.invoke({"project\_description": user\_input})

print(f"\nSuggested Template:\n{response}\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**🧪 Example Run**

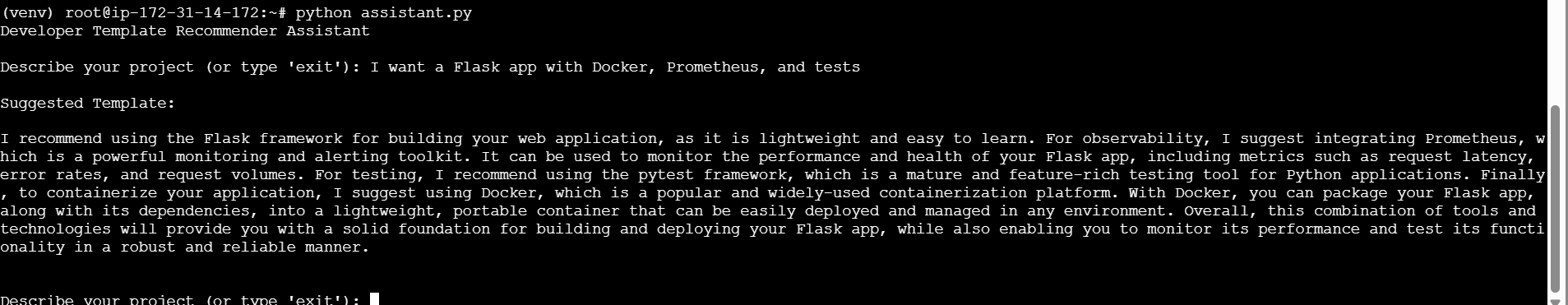
$ python cli\_ai\_assistant.py

Developer Template Recommender Assistant

Describe your project (or type 'exit'): I want a Flask app with Docker, Prometheus, and tests

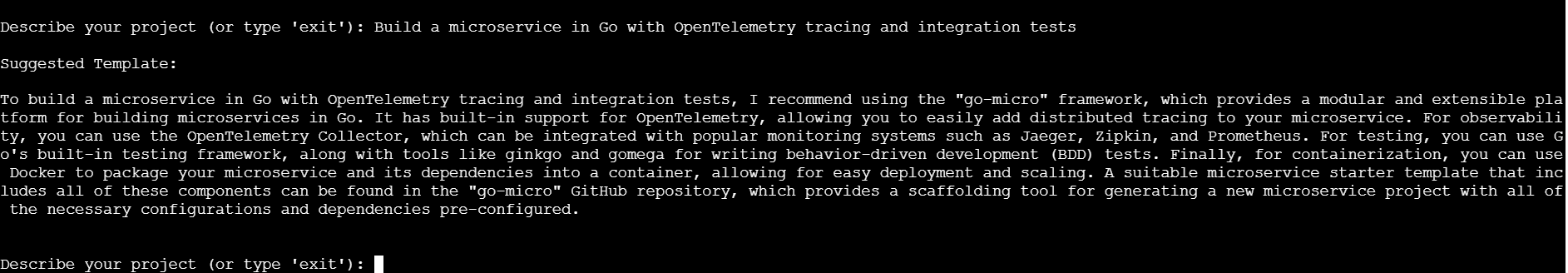
Suggested Template:

Use the `cookiecutter-flask-prometheus-docker` template. It supports Flask scaffolding, Dockerfile, Prometheus metrics and Pytest.

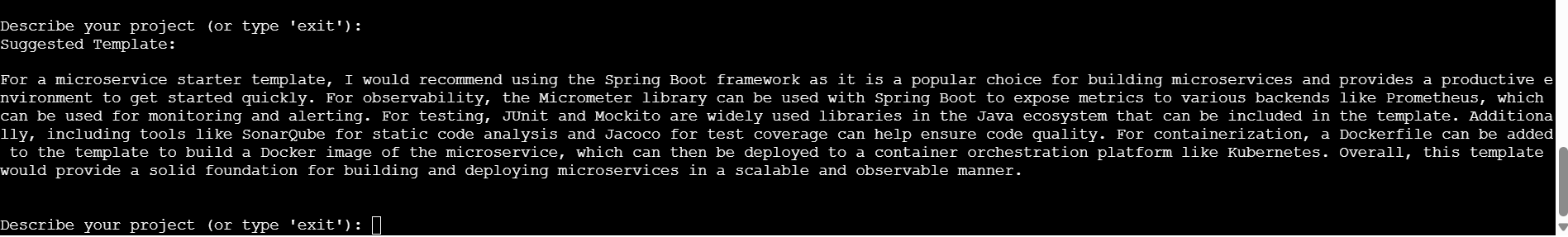


**More tests to get responses from :**

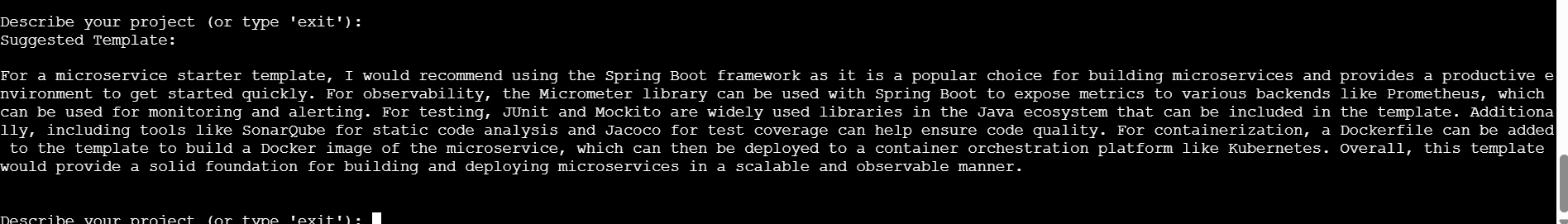
* Build a microservice in Go with OpenTelemetry tracing and integration tests



* Create a data pipeline with Apache Airflow, Docker, and monitoring alerts



* Develop a machine learning service in Python with FastAPI, logging, and Docker



* Below complete code generation applicable only with paide models : gpt 4 , other larger good models.
* **How to enable your assistant to generate complete project templates (code + config + structure)**

**Use a more detailed prompt with project scaffolding instructions**

Your prompt to the LLM should explicitly ask it to **generate the full project structure and key files**, for example:

1.

Generate a complete starter Flask microservice project that includes:

- A `app.py` file with basic Flask routes,

- A Dockerfile to containerize the app,

- Prometheus metrics instrumentation integrated in the app,

- A `tests` folder with pytest test files,

- A `requirements.txt` with necessary dependencies,

- A README.md with instructions.

Output the full content of each file clearly, including directory structure.

2.

You are a Platform Engineering expert. Given the following description:

"{project\_description}"

Suggest a suitable microservice starter template that includes:

- framework

- observability stack

- testing tools

- containerization (Docker)

Respond and Output the full content of each file clearly, including directory structure.

**Lab 7: MVP Golden Path Integration – Link Templates into Platform Design**

**🎯 Objective**

Build and validate a CLI-based AI assistant that:

* Recommends microservice templates based on user prompts
* Uses a HuggingFace-hosted LLM (Mixtral)
* Returns structured JSON responses
* Automatically scaffolds projects via Cookiecutter

It demonstrates **Golden Path Integration** where reusable templates are dynamically suggested and consumed during internal developer workflows.

pip install -U langchain langchain-community langchain-huggingface huggingface\_hub

export HUGGINGFACEHUB\_API\_TOKEN="hf\_czNLRvUJ"

(venv) root@ip-172-31-87-197:~/raman\_micro\_test# cat cli\_ai\_assistant23.py

import os

import json

import traceback

from dotenv import load\_dotenv

from langchain\_core.prompts import PromptTemplate

from langchain\_huggingface import HuggingFaceEndpoint

load\_dotenv()

# ✅ List of Available Golden Path Templates

GOLDEN\_PATH\_TEMPLATES = [

{

"name": "Golden Path Microservice (Flask, Python)",

"stack": ["python", "flask"],

"framework": "Flask",

"docker": True,

"observability": ["Prometheus", "Grafana"],

"test\_frameworks": ["pytest"],

"repo\_url": "https://github.com/ramannkhanna2/cookiecutter-golden-path.git"

}

]

# 🧠 Prompt for Stack Detection

STACK\_DETECT\_PROMPT = PromptTemplate(

input\_variables=["project\_description"],

template="""

Extract the technology stack from this project description.

Respond ONLY in this JSON format:

{{

"stack": ["<tech1>", "<tech2>", ...]

}}

Project Description: {project\_description}

"""

)

# 🔍 Match stack with available templates

def find\_matching\_template(detected\_stack):

for template in GOLDEN\_PATH\_TEMPLATES:

if any(tech.lower() in detected\_stack for tech in template["stack"]):

return template

return None

# 🏗️ Generate fallback scaffold JSON

def fallback\_scaffold(detected\_stack):

stack\_label = ", ".join(detected\_stack)

return {

"error": "❌ No matching template found for the given stack.",

"suggestion": "You can create a new Cookiecutter template for this stack. Here's a sample structure to get you started.",

"template\_scaffold": {

"cookiecutter.json": {

"project\_name": f"My {stack\_label} Microservice",

"project\_slug": f"{'\_'.join(detected\_stack)}\_microservice",

"author\_name": "Your Name",

"description": f"A simple {stack\_label} microservice template with basic observability, Docker, and tests",

"port": "3000"

},

"folder\_structure": [

"{{cookiecutter.project\_slug}}/",

"├── app.main", # generic

"├── Dockerfile",

"├── observability.yaml",

"├── test/",

"│ └── app.test"

]

}

}

def main():

print("🔧 Developer Template Recommender Assistant\n")

print("Hugging Face Token Present:", bool(os.getenv("HUGGINGFACEHUB\_API\_TOKEN")))

try:

llm = HuggingFaceEndpoint(

repo\_id="mistralai/Mixtral-8x7B-Instruct-v0.1",

temperature=0.3,

huggingfacehub\_api\_token=os.getenv("HUGGINGFACEHUB\_API\_TOKEN")

)

except Exception as e:

print("❌ Error initializing LLM:", e)

return

chain = STACK\_DETECT\_PROMPT | llm

while True:

user\_input = input("\n📝 Describe your project (or type 'exit'): ").strip()

if user\_input.lower() == "exit":

break

if not user\_input:

print("⚠️ Please enter a valid description.")

continue

try:

response = chain.invoke({"project\_description": user\_input})

print("\n📦 Raw LLM Output:\n", response)

try:

parsed = json.loads(response)

detected\_stack = [s.lower().strip() for s in parsed.get("stack", [])]

print(f"\n🔍 Detected Stack: {detected\_stack}")

template = find\_matching\_template(detected\_stack)

if not template:

fallback = fallback\_scaffold(detected\_stack)

print(f"\n❌ {fallback['error']}")

print(f"\n💡 {fallback['suggestion']}")

print("\n📁 Suggested Template Structure:")

print(json.dumps(fallback["template\_scaffold"], indent=2))

else:

print("\n✅ Recommended Template:\n")

for k, v in template.items():

print(f"{k}: {v}")

confirm = input("\n➡️ Do you want to scaffold this project using the suggested template? (y/n): ").strip().lower()

if confirm == 'y':

print(f"\n➡️ Scaffolding project using: {template['repo\_url']}")

os.system(f"cookiecutter {template['repo\_url']} --output-dir ./generated-projects")

print("\n✅ Project scaffolded successfully into: ./generated-projects")

except json.JSONDecodeError:

print("❌ Failed to parse LLM response as JSON. Raw output:")

print(response)

print("\n" + "-"\*60 + "\n")

except Exception as e:

print("Exception traceback:")

traceback.print\_exc()

print(f"Error message: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

* You can now test it by entering a project description like:

I want a Go microservice using Fiber, containerized with Docker and instrumented via Prometheus

* Above template will end in a suggestion as its not in the git repository which is referenced in prompt template.

I want a Go microservice using Fiber, containerized with Docker and instrumented via Prometheus

Build a Rust microservice using Actix-Web with Docker support and Prometheus metrics

A Java microservice using Spring Boot, Dockerized, and supports distributed tracing using Zipkin

A Node.js API using Express.js, containerized with Docker, and logs exported via Prometheus

**🧱 Next Step (optional) : Add a Golden Path Template for Go + Fiber**

Update your GOLDEN\_PATH\_TEMPLATES like this:

GOLDEN\_PATH\_TEMPLATES = [

{

"name": "Golden Path Microservice (Flask, Python)",

"stack": ["python", "flask"],

"framework": "Flask",

"docker": true,

"observability": ["Prometheus", "Grafana"],

"test\_frameworks": ["pytest"],

"repo\_url": "https://github.com/ramannkhanna2/cookiecutter-golden-path.git"

},

{

"name": "Golden Path Microservice (Go, Fiber)",

"stack": ["go", "fiber"],

"framework": "Fiber",

"docker": true,

"observability": ["Prometheus"],

"test\_frameworks": ["go test"],

"repo\_url": "https://github.com/YOUR\_USERNAME/cookiecutter-go-fiber-template.git"

},

{

"name": "Golden Path Microservice (Node.js, Express)",

"stack": ["node.js", "express"],

"framework": "Express",

"docker": true,

"observability": ["Prometheus"],

"test\_frameworks": ["jest"],

"repo\_url": "https://github.com/YOUR\_USERNAME/cookiecutter-node-express-template.git"

}

]

**OR CREATE A templates.json file as well and put all templates over ther to make the setup more scalable.**

**🧩 Recap: What You’ve Done Till Now**

**✅ Day 1 & Day 2 Summary**

| **Area** | **What You Did** |
| --- | --- |
| **Golden Paths** | Built a cookiecutter-based scaffold template (raman-micro) for microservices |
| **App Stack** | Flask app + Prometheus metrics + Dockerfile + pytest |
| **CLI Assistant** | Created a LangChain-based recommender to generate/scaffold projects |
| **Reusable CI/CD** | Generated starter GitHub Actions pipelines |
| **MVP Golden Path Integration** | Validated Golden Path + pipeline wiring in the platform setup |

✅ At this point: you have a solid service bootstrap + automation layer ready for **developer workflows**.

**🔐 Where Day 3 – Lab 1 Fits In**

Now you’re moving **from developer workflows → secure infrastructure foundation**.

**🧱 Day 3 Lab 1 is about Platform Layer Infra:**

| **Component** | **Role in Platform Engineering** |
| --- | --- |
| **VPC** | Isolates microservices in private network zones |
| **S3 (secure)** | Stores artifacts (state files, logs, build cache, backups) securely |
| **IAM (least privilege)** | Allows fine-grained access control for platform tools (Vault, runners, apps) |
| **Terraform Modules** | Provide repeatable, compliant IaC for every team/app/env |

**🔄 Integration Between Days 1–3**

Here’s how everything fits:

plaintext

CopyEdit

┌─────────────────────────────┐

│ Golden Path (Day 1–2) │

│ • Cookiecutter Scaffold │

│ • Docker, Prometheus, CI │

└────────────┬────────────────┘

│

▼

┌──────────────────────────────┐

│ Terraform Infra Modules │ (Day 3–Lab 1)

│ • VPC + Subnets │

│ • Secure S3 for State/Apps │

│ • IAM Roles for App & Tools │

└────────────┬─────────────────┘

│

▼

┌──────────────────────────────┐

│ Secrets + Guardrails │ (Day 3–Labs 2–4)

│ • Vault, OPA, Gatekeeper │

└────────────┬─────────────────┘

▼

┌──────────────────────────────┐

│ Observability + Portal │ (Day 4)

│ • Register in Backstage │

│ • Grafana Dashboards │

│ • Docs + Deploy via Portal │

└──────────────────────────────┘