

Working with LangChain and LCEL



Demo: Interaction with Deployed Chains on LangServe

Overview

In this project, a joke generation service has been demonstrated using LangChain, FastAPI, and Google's Gemini 1.5 Flash model. A LangChain pipeline has been hosted as an API using LangServe, and a command-line interface (CLI) has been provided for consuming the deployed service. The purpose of this Demonstration is to familiarize with the deployment environment offered by LangServe.

Two core components are included:

- A server that serves the LangChain pipeline via FastAPI.
- A client that connects to the server and retrieves AI-generated jokes based on a given topic.

Scenario

It is often desirable for AI applications to provide contextual, humorous responses in real-time. In this scenario, a joke generator has been designed to produce clean and relevant jokes on a given topic. The solution has been architected to run locally, be easily deployable, and allow interaction through both web-based and CLI interfaces.

Problem Statement

Traditional joke generation systems often rely on static templates or predefined joke lists, which can result in repetitive or generic output. Additionally, integrating powerful AI-based language models into applications often requires complex infrastructure.

To address these challenges:

- A modern LLM (Google Gemini 1.5 Flash) is utilized.
- LangChain is employed to manage prompts and model outputs.
- LangServe is used to expose the pipeline via a FastAPI server.
- A simple CLI is implemented to demonstrate external API consumption.

Approach for the Solution

A modular and lightweight system was developed by combining the following components:

- **Prompting:** A structured prompt was crafted using LangChain's ChatPromptTemplate.
- **Language Model:** The Gemini 1.5 Flash model was selected for fast, context-aware generation.
- **Serving:** The chain was served using LangServe over a FastAPI application.
- **Client Access:** A CLI interface was developed using RemoteRunnable to invoke the API.

Implementation Steps

Project Structure

joke-generator/

```
|— server.py      # LangServe-based FastAPI server
|— client.py     # CLI to invoke joke generator
|— .env          # API key configuration
|— requirements.txt # Dependencies list
```




Stepwise Solution

Step 1: Setting up the Environment

- **Objective:** Set up a virtual environment, install necessary dependencies, and load the environment variables.


Requirements.txt



```
1  langchain
2  langchain-google-genai
3  fastapi
4  uvicorn
5  langserve
6  sse_starlette
7  python-dotenv
```

- Dependencies must be installed using the following command.

Code:



```
1  pip install requirements.txt
```

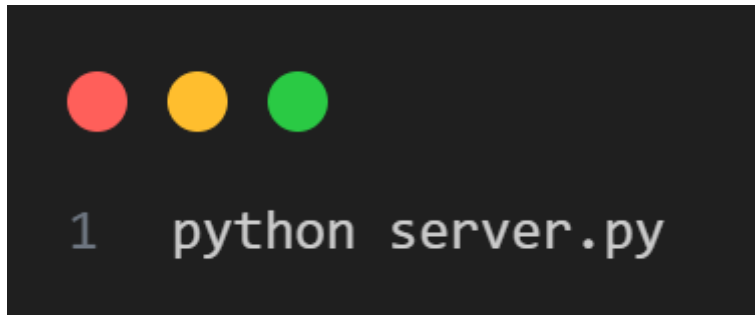
- Create a `.env` file to store API keys (like `GOOGLE_API_KEY`) for the AI model.

Step 2: Server Setup (server.py)

The server has been configured to perform the following:

- Construct a prompt to instruct the model to generate topic-based jokes.
- Configure and invoke the Google Gemini model.
- Parse and return the joke text.
- Expose the chain using LangServe on the /joke-generator route.

To start the server, the following command should be run:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The command `1 python server.py` is entered at the prompt.

```
1 python server.py
```

The API can then be explored via:

- Swagger UI: <http://localhost:8000/docs>
- LangChain Playground: <http://localhost:8000/joke-generator/playground/>

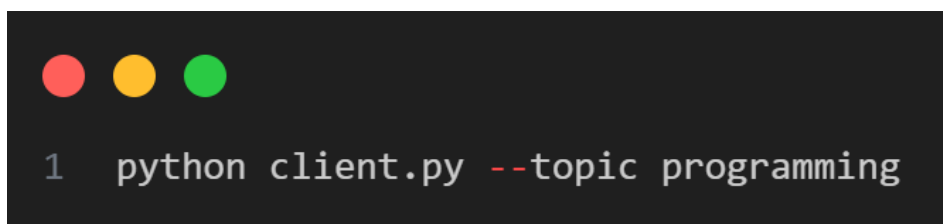
Step 3: Client Setup (client.py)



The client script has been designed to:

- Accept a topic from the command line.
- Connect to the remote API endpoint.
- Receive and display the joke generated by the server.

The client can be executed as follows:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The command `1 python client.py --topic programming` is entered at the prompt.

```
1 python client.py --topic programming
```

Other topics may be tested:

- `python client.py --topic sports`
- `python client.py --topic space`

Stepwise Solution

Core Components

Component	Description
ChatPromptTemplate	A LangChain utility used to define prompt structure
ChatGoogleGenerativeAI	Gemini 1.5 Flash model integrated with LangChain
StrOutputParser	Extracts text output from model responses
LangServe	Hosts the chain over FastAPI
RemoteRunnable	Enables the client to invoke the chain remotely via HTTP

Code Explanation

- server.py – FastAPI Server : This file defines the backend API.

```
1 from fastapi import FastAPI
2 from langchain_core.prompts import ChatPromptTemplate
3 from langchain_core.output_parsers import StrOutputParser
4 from langchain_google_genai import ChatGoogleGenerativeAI
5 from langserve import add_routes
6 from dotenv import load_dotenv
7 import os
```

- Imports: LangChain, FastAPI, and Gemini components are imported.
- Environment loading: .env is loaded using dotenv.

```
1 system_template = "You are a helpful assistant that generates jokes about {topic}."
2 prompt_template = ChatPromptTemplate.from_messages([
3     ('system', system_template),
4     ('user', 'Tell me a short, clean joke about {topic}.')
5 ])
```

- A prompt template is configured using LangChain's ChatPromptTemplate.
- It includes system-level instruction and a user-level query.

```
1 model = ChatGoogleGenerativeAI(
2     model="gemini-1.5-flash",
3     google_api_key=os.environ.get("GOOGLE_API_KEY")
4 )
```

- The Gemini model is instantiated using an API key from the .env.

```
1 parser = StrOutputParser()
2 chain = prompt_template | model | parser
```

- A FastAPI app is initialized with metadata.
- The chain is exposed through LangServe at /joke-generator.
- A basic homepage route is created for API discovery.

```
1 @app.get("/")
2 async def root():
3     return {
4         "message": "Welcome to the Joke Generator API",
5         "endpoints": {
6             "joke_generator": "/joke-generator",
7             "docs": "/docs"
8         }
9     }
```

client.py – Command-Line Client: This script allows the user to query the API from a terminal.

- RemoteRunnable is used to invoke chains over HTTP.
- argparse handles command-line argument parsing.



```
1 parser = argparse.ArgumentParser(description='Joke Generator Client')
2 parser.add_argument('--topic', type=str, default='programming',
3                       help='Topic to generate a joke about')
4 args = parser.parse_args()
```

- The topic is retrieved via CLI input (default: "programming").
`remote_chain = RemoteRunnable("http://localhost:8000/joke-generator/")`
- A connection is made to the remote LangServe-hosted chain.
`response = remote_chain.invoke({"topic": args.topic})`
- The API is called by passing the topic.
- The joke is received and printed.

Usage Instructions

- Ensure the .env file contains a valid API key.
- Launch the API server: `python server.py`
- In a new terminal, run the client with a topic: `python client.py --topic food`
- Jokes should be returned and displayed in the terminal.

Testing and Validation

- The server.py must be running on localhost:8000 for the client to function.
- The API may be tested through the /docs endpoint.
- Any connectivity issues or missing key errors will be raised by the client.

Conclusion

The Joke Generator API serves as a practical example of how LangChain's modular pipeline can be deployed as a microservice using FastAPI and LangServe. By abstracting the complexities of prompt engineering, model invocation, and output parsing into reusable chains, a highly maintainable and scalable system has been demonstrated. The inclusion of a command-line client further exemplifies how such APIs can be consumed in lightweight interfaces without requiring full-stack development.

This project not only highlights the ease of integrating powerful language models like Gemini 1.5 Flash into custom applications but also illustrates key principles such as chain deployment, remote invocation, and interactive NLP tooling. With minimal setup, a developer is able to generate intelligent and creative content tailored to user-specified topics, opening doors for a wide range of use cases—from chatbots and entertainment tools to education and beyond.

This architecture can be extended for more advanced AI-powered services, demonstrating a clear path from prototype to production-ready systems using LangChain.

