

## LangSmith Setup Guide: Tracing Your LangGraph Applications

LangSmith is LangChain's developer platform that lets you trace, debug, evaluate, and monitor your LLM applications—including LangGraph workflows—automatically.

This guide walks you through:

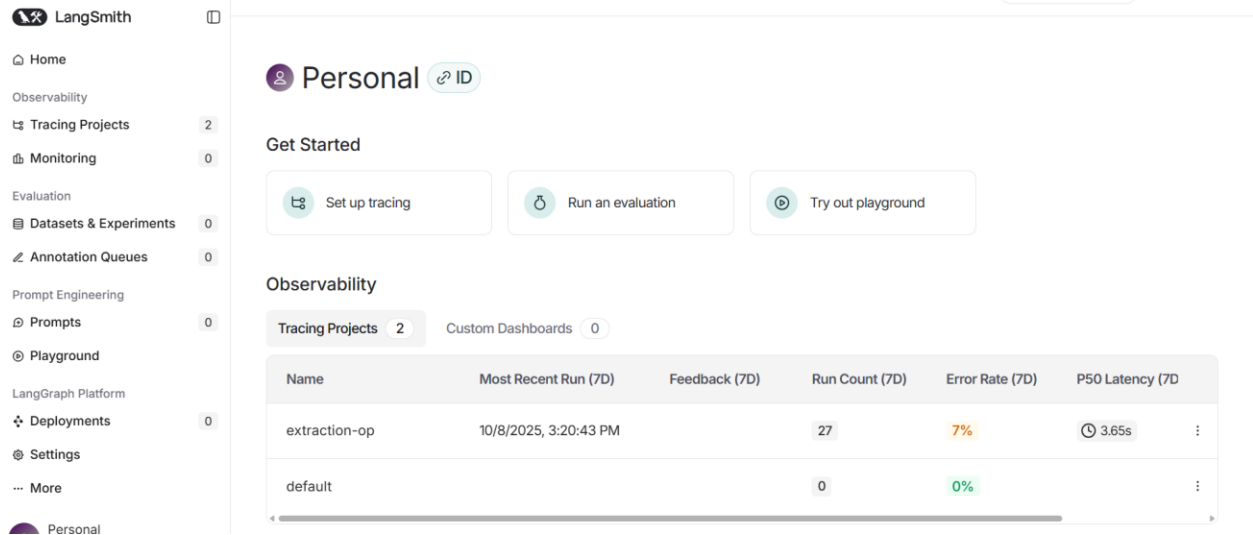
You can go to this link for more details on how to generate API key:

*<https://docs.langchain.com/langsmith/create-account-api-key>*

1. Creating a LangSmith account.
2. Getting your **\*\*API key\*\***.
3. Configuring your .env file.
4. Enabling automatic tracing for your LangGraph app.

### ### Step 1: Create a LangSmith Account

1. Go to <https://smith.langchain.com>
2. Click "Sign Up" (you can use Google, GitHub, or email)
3. Complete registration and verify your email
4. You'll be redirected to the LangSmith dashboard



### ### Step 2: Get Your API Key

In the LangSmith dashboard, click your profile icon (top-right) → "Settings"

1. Go to the "API Keys" tab in settings
2. Click "Create API Key"
3. Give it a name (e.g., my-langgraph-dev)
4. Click "Create"
5. Copy your API Key **\*\*(you won't see it again!)\*\***

< Back

Organizations

Org ID

Personal

Workspaces

PLUS

Members and roles

PLUS

API Keys

OAuth Providers

Models

Shared

Secrets

Feedback tags

Billing and usage

Documentation

API Keys ⓘ

PersonalService

API docs+ API Key

Key	Expires	Description	Created At	Last Used At	Actions
lsv2_pt_c ... 3495	Never	Auto created during onboarding	9/24/2025, 11...	10/8/2025, 3:...	

### ### Step 3: Set Up Your .env File

In your project root create a **\*\*.env\*\*** file and paste your credentials

#### # LangSmith Configuration

LANGCHAIN\_API\_KEY=your\_actual\_api\_key\_here

LANGCHAIN\_PROJECT=your\_project\_name\_here

LANGCHAIN\_TRACING\_V2=true

LANGCHAIN\_ENDPOINT=<https://api.smith.langchain.com>

### ### Step 4: Automatic Tracing in LangGraph (Zero Code Changes!)

**Once you set the environment variables above, LangChain (and LangGraph) will automatically trace every run—no code changes needed!**

How It Works:

LangChain detects `LANGCHAIN_TRACING_V2=true`

Every LLM call, tool use, and graph node execution is sent to LangSmith

Your LangGraph state transitions, messages, and outputs appear in the dashboard

### **### Now**

Go to LangSmith Dashboard → "Traces"

The screenshot displays the LangSmith interface, which is used for monitoring and debugging AI applications. The top section shows a project overview for 'extraction-op' with various tabs like Runs, Threads, Evaluators, and Automations. A table lists several runs, including 'default' and 'LangGraph', with columns for Name, Input, Output, Error, and Start Time. A 'Stats' panel on the right provides summary metrics such as Run Count (27), Total Tokens (81,398 / \$0.44), Median Tokens (444), Error Rate (7%), % Streaming (6%), and First Token (P50: 0.85s).

The bottom section shows a detailed 'TRACE' of a specific run. It lists the sequence of nodes in the LangGraph: 'default', 'extract\_chunks' (0.01s), 'convert\_to\_...' (1.86s), 'gpt-4o' (1.84s, 298 tokens), 'extract\_struc...' (1.10s, 443 tokens), 'gpt-4o' (1.09s), and 'save\_json' (0.01s). The 'Input' section on the right shows the state of the graph, including 'Output Json Path' (madhu.json), 'Output Md Path' (madhu.md), 'Pdf Path' (Fail-6.pdf), 'Markdown Chunks' (0), and 'Text Chunks' (0).

You'll see your full LangGraph execution with:

Each node (extract\_chunks, convert\_to\_markdown, etc.)

Input/output state

LLM calls with prompts & responses

Timing and tokens used

**### For this you can run main.py or directly run the run\_pipeline function in the main.graph.py with parameters and you can check the tracing for test purpose.**

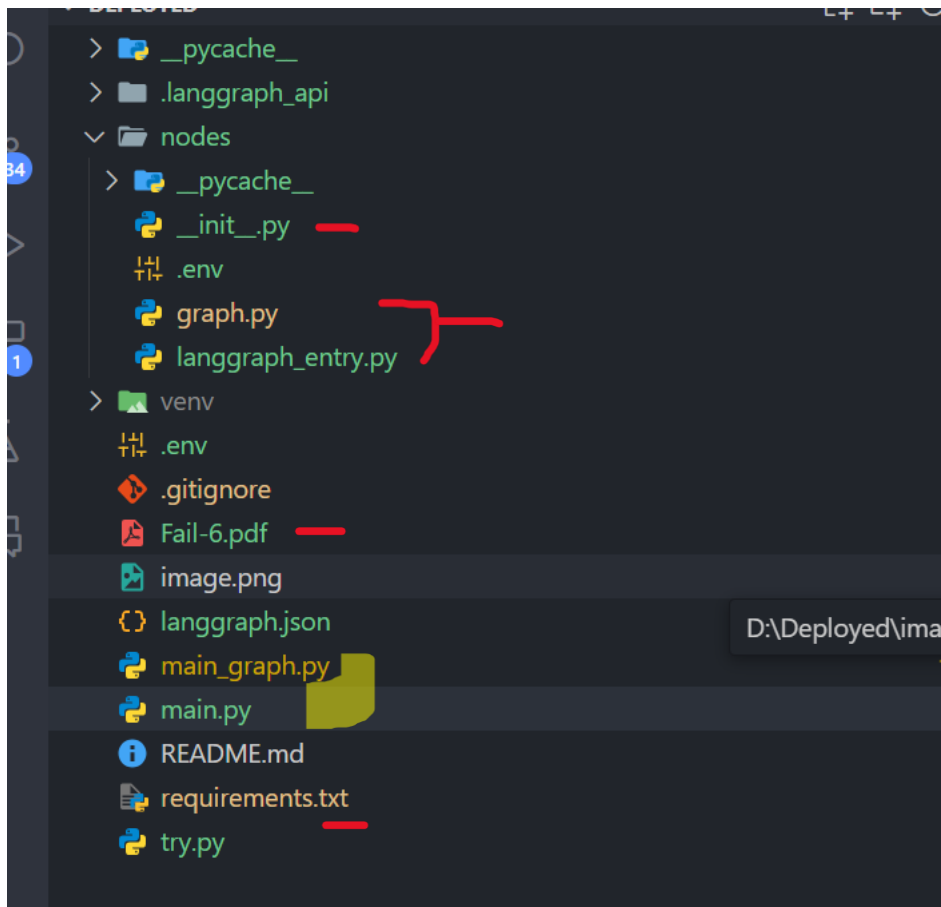
**### Now we would want to deploy the graph in LangGraph Studio in order to visualize it and make it accessible through an endpoint.**

## **## Deploying Your LangGraph App Locally with LangGraph Studio**

**\*\*Organize your folder structure like this\*\***

Deployed/

- |—— .env
- |—— langgraph.json ← LangGraph deployment config
- |—— nodes/
  - |   |—— langgraph\_entry.py ← Entry point for the server
- |   |—— graph.py ← Contains your graph
- |   |—— run\_server.py ← manual server script



# langgraph.json -> tells the LangGraph server where to find the Graph.

```
{  
  "graphs": {  
    "default": {  
      "path": "nodes.langgraph_entry:graph"  
    }  
  },  
  "dependencies": [  
    "langgraph>=0.1.0",  
  ]  
}
```

```
"langchain-openai",  
"python-dotenv",  
"pdfplumber",  
"pymupdf",  
"pillow"  
]  
}
```

It is also having the dependencies that are required to run the graph.

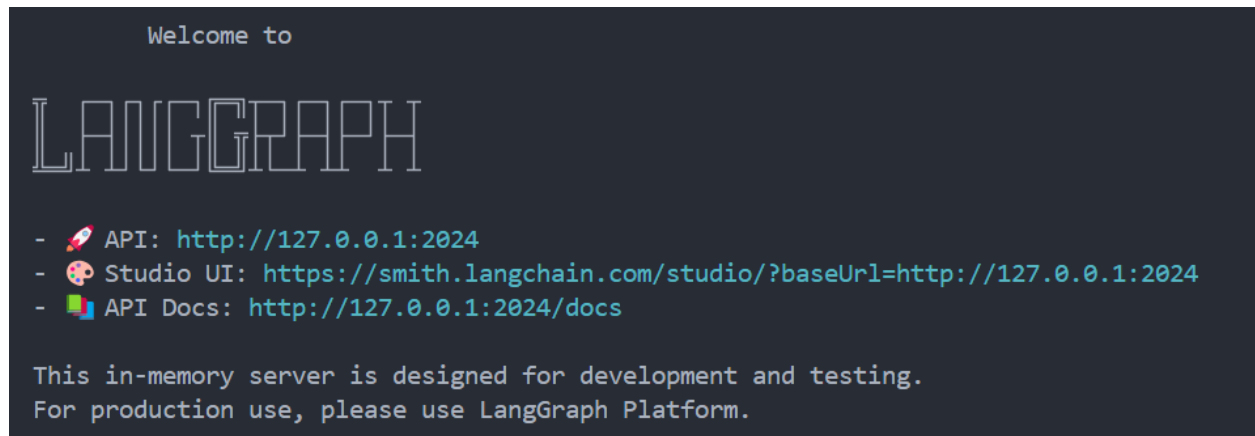
Now we need to have **\*\*langgraph-cli\*\*** installed in the environment.

**### use langgraph dev to run from the root folder**

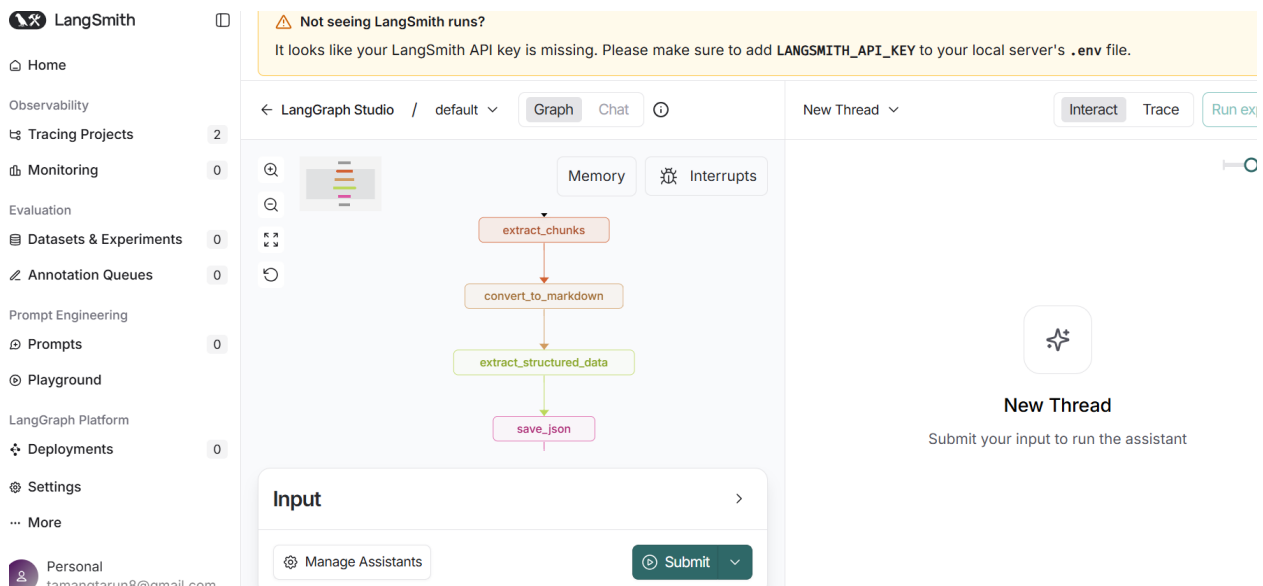
```
langgraph dev
```

It will start the development server and take you to Langgraph studio hosted on  
**\*\*<http://127.0.0.1:2024>\*\***

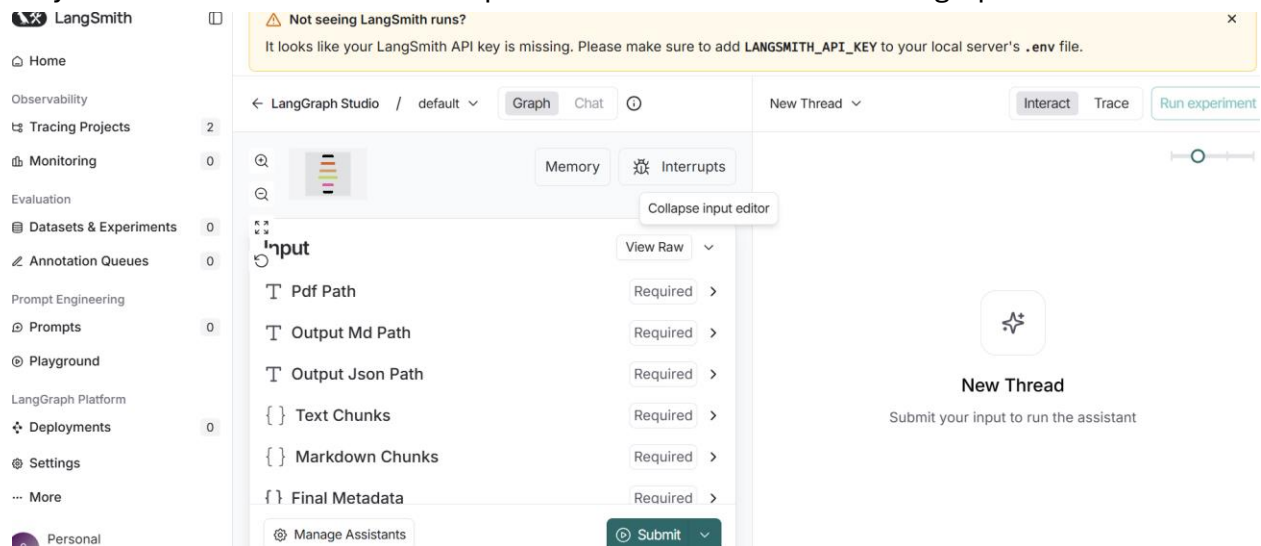




**\*\*We can check on the Langgraph Studio to run it and it's like a playground where we can see our graphs and agents working \*\***



As you can see below we have the parameter or the data of state of graph



**\*\*Now using try.py we will access the above endpoint and send our files and run trace it in langsmith server\*\***

When running locally give assistant\_id as default

LangSmith