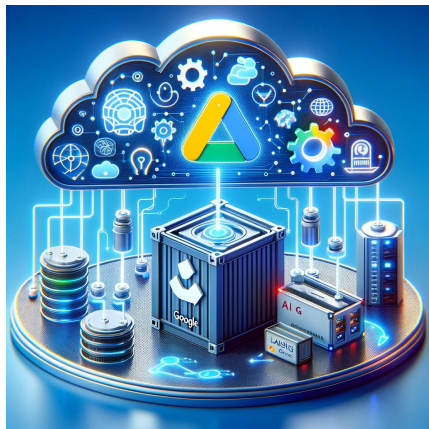


Pulling Google Drive Data into RAG with LangServe and Docker

Deploying a RAG toolchain using LangServe in your local Docker environment



Purpose



- Learn to leverage LangChain Templates and Docker to build scalable microservices for your chains with ease



- See how you can interact with LangServe to create FastAPI endpoints for dynamically create and invoke your chains
- Streamlit as frontend to interact with your chains
- Docker to containerize and deploy your AI microservices



- See the app in action performing RAG on your own data



- Try it out and create your own with your preferred document loader!

Use Cases for RAG on your own Data

- Retrieving engineering standards / procedures
- Querying meeting transcripts
- Getting insights on project info that would otherwise be overlooked

Lab Deep Dive

Technologies Used

Backend

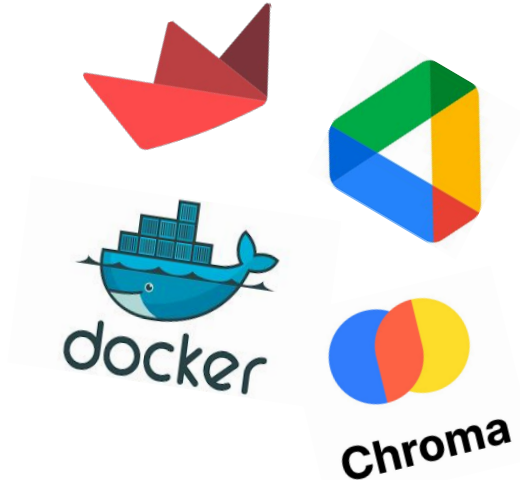
- LangChain [rag-chroma](#) template
- [Fast API](#) + [LangServe](#)
- Open AI - [ChatOpenAI](#) + [OpenAIEmbeddings](#)
- [Chroma](#)
- [Google Drive API](#)

Frontend

- [Streamlit](#) - [link to tutorial](#)

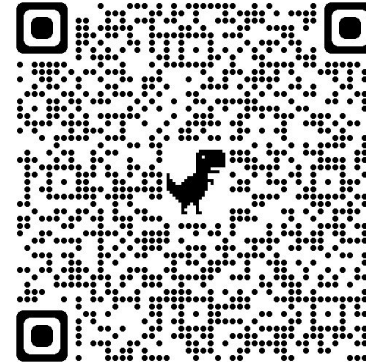
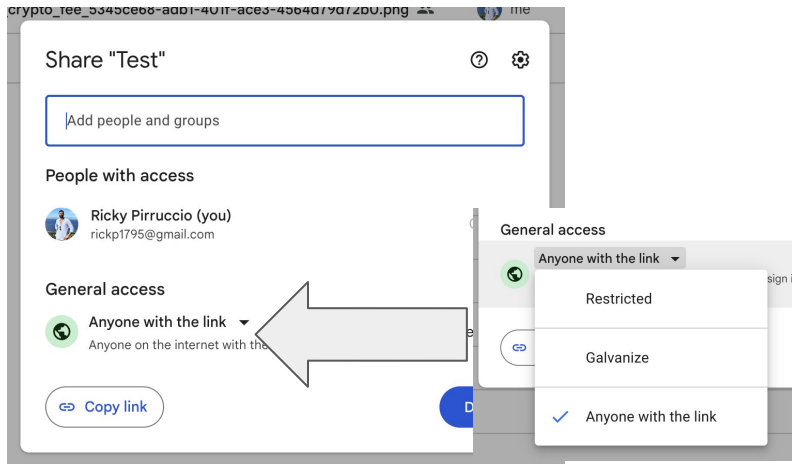
Deployment and Development Pipeline

- [Docker](#) - [link to tutorial](#)



Google Credentials Prerequisites

- You must create a google service account before you can access the Google Drive API on your own folders
- Karim Lalani made a great tutorial on this [here](#)
- Important: Set your folder sharing permissions to “Anyone with the link”



Prerequisites

Clone the repo

- `git clone https://github.com/colinmcnamara/austin_langchain`
- `cd austin_langchain/labs/LangChain_103/rag_chroma_from_google_drive`

Installs

- [Docker Desktop](#)

Environment Variables

- `GOOGLE_APPLICATION_CREDENTIALS`
- `OPENAI_API_KEY`

Live Demo

Docker setup

Chain Microservice
(Backend)


Streamlit Microservice
(Frontend)

Compose

```
Dockerfile > ...
1 FROM python:3.11-slim
2
3 RUN pip install poetry==1.6.1
4
5 RUN pip install google-api-python-client google-auth-oauthlib && \
6     pip install pypdf2
7
8 RUN poetry config virtualenvs.create false
9
10 ENV GOOGLE_APPLICATION_CREDENTIALS=/app/.credentials/keys.json
11
12 WORKDIR /code
13
14 COPY ./pyproject.toml ./langchain_template_README.md ./poetry.lock* ./
15
16 COPY ./packages ./packages
17
18 RUN poetry install --no-interaction --no-ansi --no-root
19
20 COPY ./app ./app
21
22 RUN poetry install --no-interaction --no-ansi
23
24 EXPOSE 8000
25
26 CMD exec uvicorn app.server:app --host 0.0.0.0 --port 8000
27
```

```
Dockerfile.streamlit > ...
1 # Use an official Python runtime as a parent image
2 FROM python:3.11-slim
3
4 # Set the working directory in the container
5 WORKDIR /usr/src/app
6
7 # Install Streamlit
8 RUN pip install streamlit
9
10 # Copy the Streamlit app file into the container
11 COPY streamlit_chat.py .
12
13 # Make port 8501 available to the world outside this container
14 EXPOSE 8501
15
16 # Run Streamlit when the container launches
17 CMD ["streamlit", "run", "streamlit_chat.py"]
```

```
docker-compose.yml
1 version: '3.8'
2
3 services:
4   api:
5     build:
6       context: .
7       dockerfile: Dockerfile
8     environment:
9       - OPENAI_API_KEY=${OPENAI_API_KEY}
10    volumes:
11      - ${GOOGLE_APPLICATION_CREDENTIALS}:/app/.credentials/keys.json
12    ports:
13      - "8000:8000"
14
15   frontend:
16     build:
17       context: .
18       dockerfile: Dockerfile.streamlit
19     depends_on:
20       - api
21    ports:
22      - "8501:8501"
```

 docker-compose.yml

 Dockerfile

 Dockerfile.streamlit

Basic Backend Setup

```
server.py 4 X
app > server.py > ...

11 @app.get("/")
12 async def redirect_root_to_docs():
13     return RedirectResponse("/docs")
14
15
16 @app.get("/list-folder-routes")
17 def list_folder_routes():
18     routes = set()
19     for route in app.routes:
20         if isinstance(route, APIRoute):
21             if route.name == "invoke" and route.path.startswith(
22                 "/folders/{folder_id}/"
23             ):
24                 routes.add(route.path)
25     return list(routes)
26
27
28 @app.post("/initialize-chain")
29 async def initialize_chain_endpoint(
30     folder_id: str = Body(..., embed=True), name: str = Body(..., embed=True)
31 ):
32     try:
33         chain = create_chain(folder_id)
34         new_path = f"/folders/{folder_id}/{name}"
35         add_routes(app, chain, path=new_path)
36         return {
37             "message": f"Chain initialized successfully at {new_path}",
38             "path": f"/{new_path}/invoke",
39         }
40     except Exception as e:
41         raise HTTPException(status_code=500, detail=str(e))
42
43
44 if __name__ == "__main__":
45     import uvicorn
46
47     uvicorn.run(app, host="0.0.0.0", port=8000)
48
```

Adds a new chain

Retrieves a chain

```
chain.py 9+ X
packages > rag-chroma > rag_chroma > chain.py > create_chain.py

13 def create_chain(folder_id):
14
15
16     loader = GoogleDriveLoader(
17         folder_id=folder_id,
18         recursive=False,
19         # we need to use service_account_key to get google credentials because we're using data
20         # issues/8755
21         file_types=["document", "sheet", "pdf"],
22         service_account_key=os.environ["GOOGLE_APPLICATION_CREDENTIALS"],
23     )
24     data = loader.load()
25
26     # Split
27     text_splitter = RecursiveCharacterTextSplitter(
28         chunk_size=1000, chunk_overlap=200, separators=[" ", ",", "\n"]
29     )
30     all_splits = text_splitter.split_documents(data)
31
32     # Add to vectorDB
33     vectorstore = Chroma.from_documents(
34         documents=all_splits,
35         collection_name="rag-chroma",
36         embedding=OpenAIEmbeddings(),
37     )
38     retriever = vectorstore.as_retriever()
39
40     # RAG prompt
41     template = """
42     Answer the question based only on the following context:
43     {context}
44
45     Question: {question}
46     """
47     prompt = ChatPromptTemplate.from_template(template)
48
49     # LLM
50     model = ChatOpenAI()
51
52     # RAG chain
53     chain = (
54         RunnableParallel({"context": retriever, "question": RunnablePassthrough()})
55         | prompt
56         | model
57         | StrOutputParser()
58     )
59
60     # Add typing for input
61     class Question(BaseModel):
62         __root__: str
63
64     chain = chain.with_types(input_type=Question)
65
66     return chain
```

rap chain creation in a function. Use to create FastAPI chain endpoint

Doc loader

Text splitter

Vector store layer

Prompt

Model

Chain

Basic Frontend Setup

Define vars

```
import requests
import streamlit as st

API_BASE = "http://api:8000"
state = st.session_state
```

Initializes state

```
def initialize_state():
    if "chat_histories" not in state:
        state.chat_histories = {}
    if "selected_path" not in state:
        state.selected_path = ""
    if "folder_routes" not in state:
        state.folder_routes = []
```

Handles saving /
retrieving chat
history

```
class ChatManager:
    def add_message(self, role, content):
        if state.selected_path not in state.chat_histories:
            state.chat_histories[state.selected_path] = []

        message = {
            "role": role,
            "content": content,
        }
        state.chat_histories[state.selected_path].append(message)

    with st.chat_message(role):
        st.write(content)

    def display_chat_history(self):
        # Display chat messages from history for the selected path
        if state.selected_path in state.chat_histories:
            for message in state.chat_histories[state.selected_path]:
                with st.chat_message(message["role"]):
                    st.write(message["content"])
```

Handles selecting and
displaying chains in
the sidebar

```
def handle_folder_configs():
    if state.folder_routes == []:
        get_folder_routes()

    with st.sidebar:
        st.header("Folder Configurations")
        folder_id = st.text_input("Enter Google Drive Folder ID")
        config_name = st.text_input("Enter a name for this configuration")
        if st.button("Add New Configuration",
            ):
            add_new_folder_config(config_name, folder_id)

        if state.folder_routes:
            st.sidebar.header("Select Drive to Chat With")
            for path in state.folder_routes:
                if st.button(path):
                    select_config(path)
```

Our condensed
app

```
st.title("Chat With Google Drive Files - Rag Chroma")

initialize_state()
chat_manager = ChatManager()

handle_folder_configs()

if state.selected_path:
    chat_manager.display_chat_history()

if prompt := st.chat_input("What is your query?"):
    chat_manager.add_message("user", prompt)

    response = get_response_from_llm(prompt, state.selected_path)

    chat_manager.add_message("assistant", response)
```

Connecting Backend with Frontend

Query a chain



```
@st.cache_data
def get_response_from_llm(query, path):
    payload = {"input": query, "config": {}, "kwargs": {}}
    res = requests.post(f"{API_BASE}/{path}", json=payload)
    if res.status_code == 200:
        return res.json()["output"]
    else:
        return f"Error: Received status code {res.status_code}"
```

Get list of chain routes
(GET request endpoint)



```
def get_folder_routes():
    response = requests.get(f"{API_BASE}/list-folder-routes")
    if response.status_code == 200:
        folder_routes = response.json()
        state.folder_routes = folder_routes
    else:
        st.error(f"{response.status_code} Error: Failed to retrieve folder routes")
```

Create new chain / chain
endpoint
(POST request endpoint)



```
def add_new_folder_config(name, folder_id):
    response = requests.post(
        f"{API_BASE}/initialize-chain", json={"folder_id": folder_id, "name": name}
    )
    if response.status_code == 200:
        path = response.json().get("path")
        state.folder_routes.append(path)
        select_config(path)
    else:
        error_detail = response.json().get("detail")
        if "File not found" in error_detail:
            user_friendly_error = "Invalid folder ID, please try again."
        else:
            user_friendly_error = error_detail
        st.error(f"{response.status_code} Error: {user_friendly_error}")
```