LangChain 🤝 MinIO

基于 GenAI 的数据管理可行性探索

张海立

KubeSphere Ambassador, CNCF OpenFunction TOC Member

《LangChain 实战》作者，开源爱好者和布道师

云原生专注领域：Kubernetes, DevOps, FaaS, Observability

GenAI 专注领域：LangChain, RAG, Evaluation, Prompt + Flow Engineering

zhanghaili

webup

zhanghaili0610

沧海九粟 · Bilibili

引言：重塑 AI 应用中的数据交互模式

在当今的 AI 应用开发中，如何让用户以更自然、更直观的方式与复杂的数据存储系统交互，是一个关键挑战

自然语言交互的力量

- LangChain 使开发者能够构建允许用户通过自然语言与 MinIO 等专业数据存储系统交互的应用
- 这极大地简化了用户操作，使得复杂的数据管理变得直观易懂

开发效率的跃升

- 通过 LangChain，开发者可以快速构建智能数据管理接口，它提供了连接 AI 模型和数据存储的简洁框架
- 这大幅缩短了开发周期，使得从概念到上线的过程更加顺畅

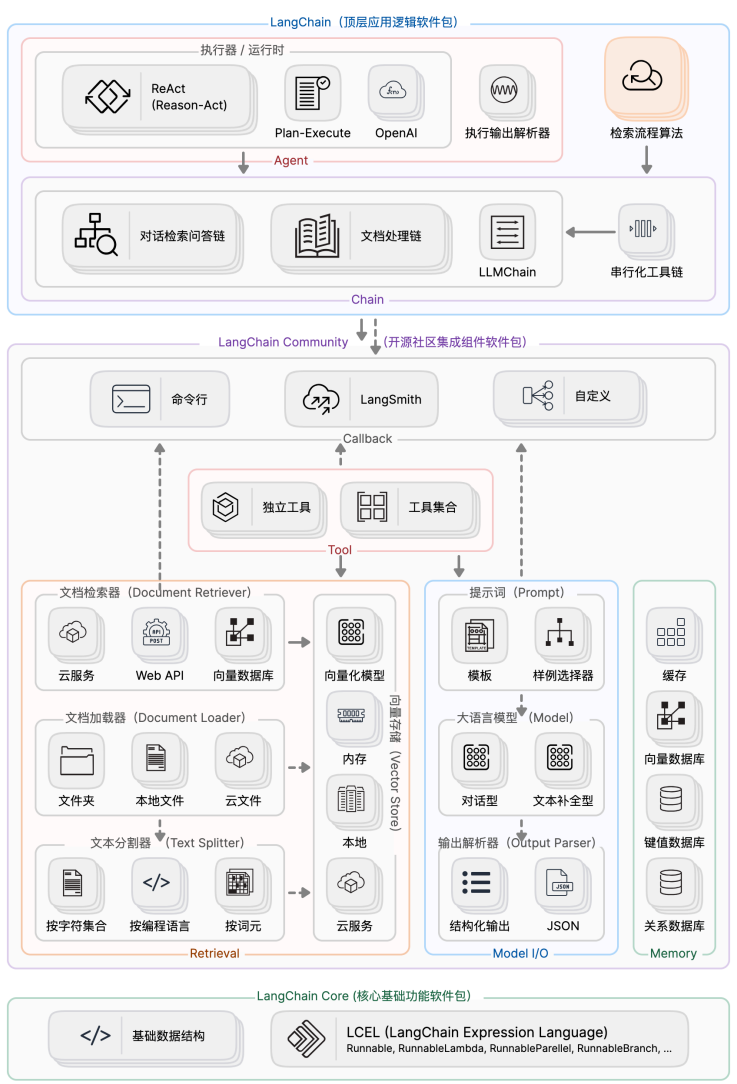
用户体验的质的提升

- 借助 LangChain 的能力，用户无需学习复杂的查询语言或了解底层存储结构，就能轻松管理和检索数据
- 这为非技术用户打开了数据分析的大门

GenAI 应用开发的助手

LangChain 提供了一套全面的工具和框架

- **RAG（检索增强生成）**
  - 集成外部知识源，大幅提升信息检索能力
  - 感知上下文，提高输出准确性和相关性
- **Agent（智能代理）**
  - 构建能自主决策和执行任务的 LLM 实体
  - 灵活调用多种工具和 API，处理复杂请求
- **Evaluation（评测评估）**
  - 提供评估框架，帮助开发者持续优化流程
  - 自定义评估指标，确保输出符合领域需求



LangChain

构建 LLM 应用的核心框架，提供丰富的组件和工具，支持从简单的对话系统到复杂的 LLM 应用开发

LangServe

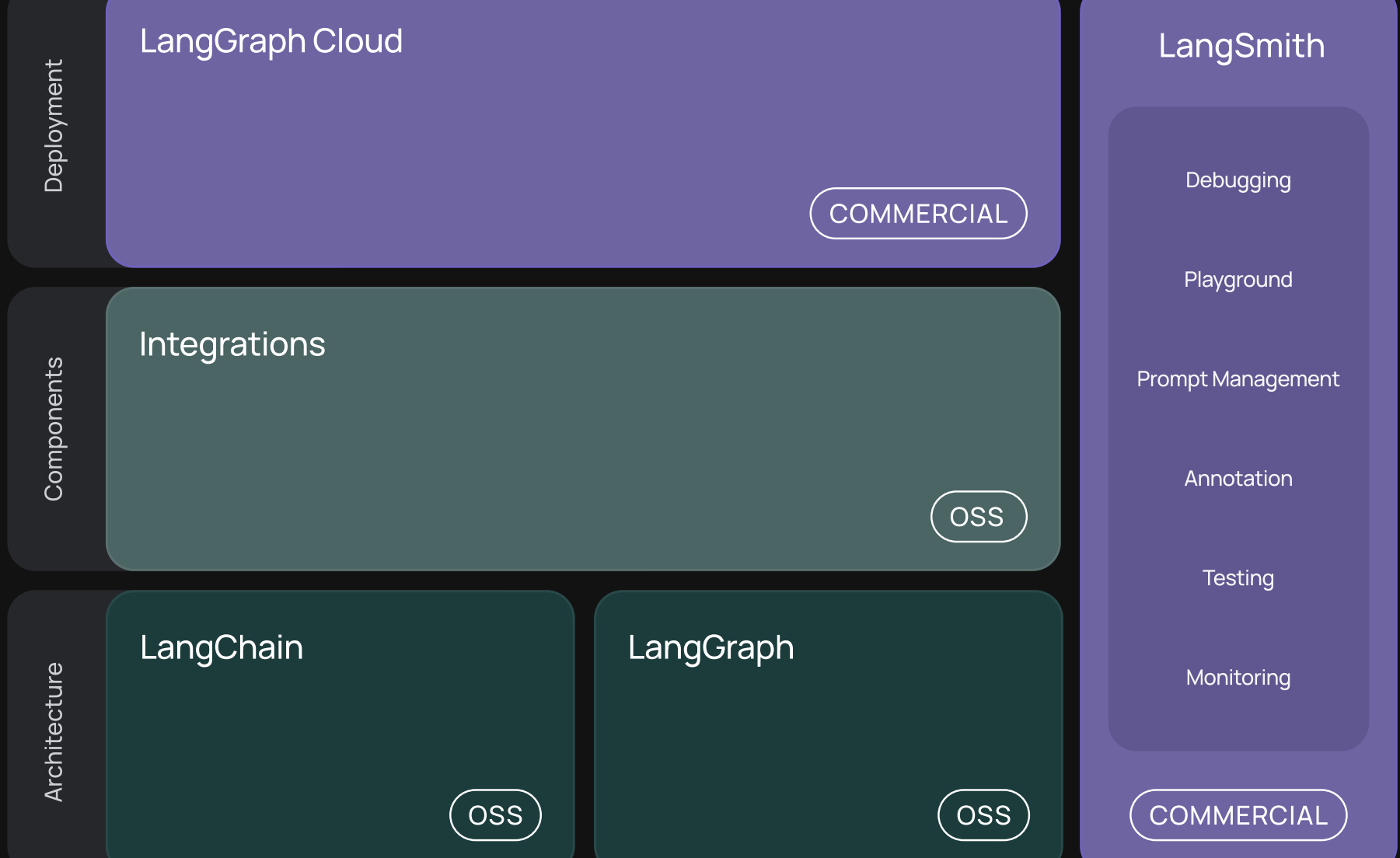
专为 LangChain 应用设计的部署解决方案，简化了 LLM 应用服务的部署和扩展过程，提高了开发效率和系统可靠性

LangSmith

强大的调试和监控平台，为开发者提供深入的洞察和优化工具，助力 LLM 应用的性能提升和质量保证

LangGraph

用于构建复杂多代理系统的创新工具，支持开发者设计和实现高度智能化的协作 LLM 应用系统


Deployment
LangGraph Cloud
COMMERCIAL
Components
Integrations
OSS
Architecture
LangChain
OSS
LangGraph
OSS
LangSmith
Debugging
Playground
Prompt Management
Annotation
Testing
Monitoring
COMMERCIAL

打造 AI 驱动的数据助手

以下是实验步骤的简要说明

1. **设置 MinIO 客户端**：配置连接参数，建立与 MinIO 存储的安全通信通道，为后续操作奠定基础
2. **创建自定义工具**：设计专门的工具函数，实现对 MinIO 存储的精确操作，如文件上传、下载和元数据管理
3. **实现 MinIO Agent**：利用 LangChain 框架构建智能代理，将自然语言处理能力与 MinIO 操作工具无缝集成
4. **使用 LangServe 创建服务**：使用 LangServe 将 Agent 转换为 API 服务，浏览自动生成的 API 端点

以下内容因为时间关系不做演示

- 设计提示工程：迭代调优提示模板，确保 Agent 能准确理解用户意图，并执行相应的 MinIO 操作
- 测试和优化：通过多轮测试和反馈，不断优化 Agent 性能和用户体验，提高系统的可靠性和效率

```
# 步骤 1: MinIO 客户端设置（如前一张幻灯片所示）

# 步骤 2: 创建自定义工具（下一张幻灯片）

# 步骤 3: 实现 LangChain 代理
agent = (
    {
        "input": lambda x: x["input"],
        "agent_scratchpad": lambda x: format_to_openai_tool_messages(x["intermediate_steps"]),
```

1️⃣ 设置 MinIO 客户端

MinIO 提供了多语言的开发框架，如 Python、JavaScript、Go 等

```python
from minio import Minio

# 首先初始化 MinIO 客户端
# play.min.io 是一个公共测试服务器，在生产环境中请替换为您自己的服务器
minio_client = Minio('play.min.io:443', access_key='minioadmin', secret_key='minioadmin', secure=True)


# 接下来检查指定 bucket 是否存在，如果不存在则创建
bucket_name = "test"

try:
    # 检查 bucket 是否存在
    if not minio_client.bucket_exists(bucket_name):
        # 如果 bucket 不存在，则创建
        minio_client.make_bucket(bucket_name)
        print(f"Bucket '{bucket_name}' created successfully.")
    else:
        print(f"Bucket '{bucket_name}' already exists.")
except S3Error as err:
    print(f"Error encountered: {err}")
```

2 创建自定义 MinIO 工具

这些自定义工具允许我们的 LangChain 代理与 MinIO 存储进行精确而高效的交互

```
@tool
def upload_file_to_minio(bucket_name: str, object_name: str, data_bytes: bytes):
    """
    将文件上传到 MinIO

    参数:
        bucket_name (str): 目标存储桶的名称，用于组织和管理对象存储中的文件
        object_name (str): 在存储桶中创建的对象名称，代表文件在 MinIO 中的唯一标识
        data_bytes (bytes): 要上传的文件的原始字节数据，确保数据完整性和安全传输
    """
    data_stream = io.BytesIO(data_bytes)
    minio_client.put_object(bucket_name, object_name, data_stream, length=len(data_bytes))
    return f"文件 {object_name} 成功上传到桶 {bucket_name}。上传过程确保了数据的完整性和安全性。"

@tool
def list_objects_in_minio_bucket(file_info):
    """
    列出 MinIO 桶中的对象，提供存储内容的全面概览

    期望 file_info 字典包含'bucket_name'键，指定要查询的存储桶
    返回包含'ObjectKey'和'Size'键的字典列表，详细展示每个对象的名称和大小信息
    """
    bucket_name = file_info['bucket_name']
```

3 实现 MinIO Agent

功能目标：基于用户输入执行精确的 MinIO 操作，实现 AI 与云存储的无缝集成

```
from langchain_core.prompts import ChatPromptTemplate
from langchain.agents.format_scratchpad.openai_tools import format_to_openai_tool_messages
from langchain.agents.output_parsers.openai_tools import OpenAIToolsAgentOutputParser
from langchain_core.messages import MessagesPlaceholder
from langchain_core.runnables import RunnableLambda

# 定义提示模板
prompt_template = ChatPromptTemplate.from_messages([
    ("system", "您是一个配备先进文件管理能力的 AI 助手，能够精确理解用户需求并高效执行复杂的数据操作任务。"),
    ("user", "{input}"),
    MessagesPlaceholder(variable_name="agent_scratchpad"),
])

# 绑定工具
upload_file_runnable = RunnableLambda(upload_file_to_minio)
list_objects_runnable = RunnableLambda(list_objects_in_minio_bucket)

tools = [upload_file_to_minio, list_objects_in_minio_bucket]
llm_with_tools = llm.bind_tools(tools)

# 创建代理
agent = (
```

4 用 LangServe 创建应用服务

将 MinIO Agent 作为 REST API 端点暴露在微服务的 /minio 路径

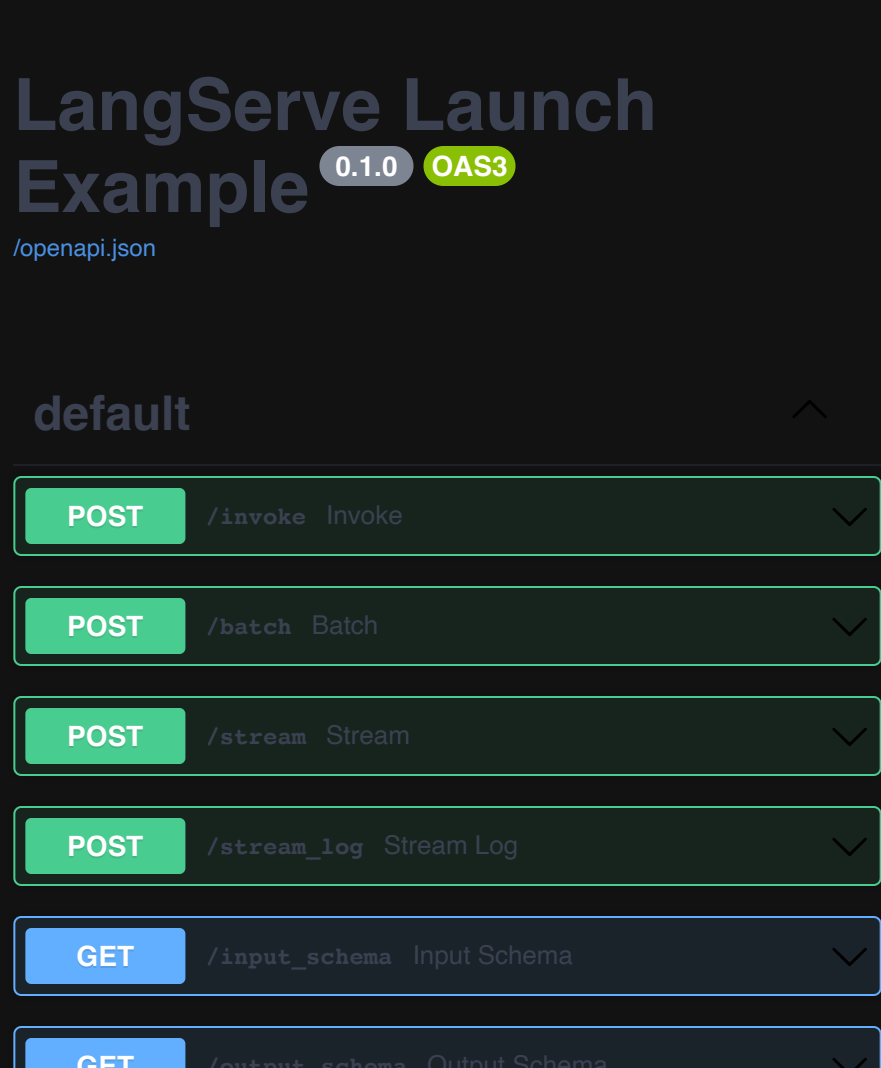
```python
from fastapi import FastAPI
from langserve import add_routes

# 初始化 FastAPI 应用
app = FastAPI(
    title="MinIO 智能数据管理 API",
    version="1.0",
)

# 添加 LangServe 路由
add_routes(
   app,
   agent_executor.with_config(
       {"run_name": "minio_data_agent"}
   ),
   path="/minio"
)

# 这段代码设置了一个集成 LangServe 的 FastAPI 服务器
# 使得 LLM 驱动的 MinIO 操作可以通过 HTTP 请求轻松调用
```

- **自动 API 生成**：LangServe 能够自动为 LangChain 应用创建 RESTful API 端点，大大简化了从开发到部署的过程，加速了 AI 服务的上线和迭代
- **内置模式验证**：提供强大的输入输出验证机制，确保数据的一致性和安全性，有效防止因数据格式错误导致的系统故障
- **无缝集成**：与现有 LangChain 代码完美兼容，使得开发者能够轻松将复杂的 LLM 应用转化为可部署的微服务，提高开发效率
- **监控和日志**：无缝集成 LangSmith 的监控和日志功能，便于开发者跟踪服务性能，快速定位和解决问题

调用 MinIO Agent

通过 LangServe SDK 调用 REST API 端点

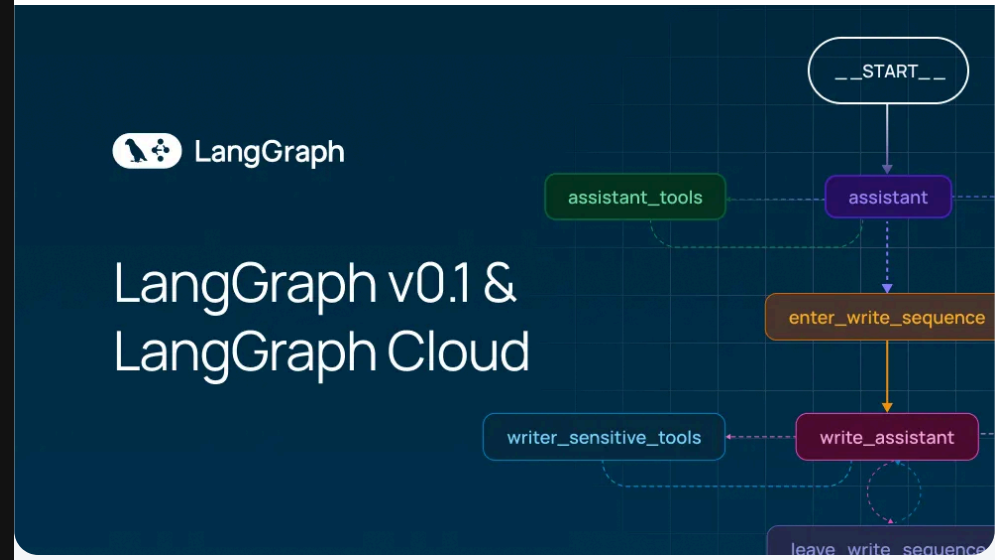
```python
from langserve import RemoteRunnable

# 连接到已部署的 API
remote_runnable = RemoteRunnable(
  "http://localhost:8000/minio/"
)

# 向代理发送请求
response = remote_runnable.invoke({
  "input": "请列出桶中的所有文件，并计算它们的总大小"
})

print(response)
# 输出 Agent 与 MinIO 交互后的详细响应
# 包括文件列表和总大小统计
```



未来展望：LangGraph 助力复杂云原生应用

- **复杂工作流编排**：LangGraph 允许开发者设计和实现复杂的多步骤 LLM 工作流，能够处理需要多个阶段决策和执行的复杂任务
- **状态管理优化**：通过图结构管理应用状态，使得长期运行的 LLM 任务能够更加稳定和可靠，提高了系统的鲁棒性
- **并行处理支持**：能够设计并执行并行任务，大大提高了复杂应用的处理效率，特别适合需要多任务协同的云原生环境
- **可视化和调试**：提供工作流程的可视化工具，更直观地设计和调试复杂的 LLM 应用逻辑

LangChain



Announcing LangGraph v0.1 & LangGraph Cloud: Running agents at scale, reliably

Subscribe

Our new infrastructure for running age

LangGraph Cloud, is available in beta. We also have a

您当前的浏览器不支持 HTML5 播放器

请更换浏览器再试试哦~

感谢聆听 🙏

有问题吗？让我们一起讨论！

LangChain 文档 | MinIO 文档 | KubeSphere 文档