

warning "Atualização de backend (outubro de 2026)" A inferência do GitHub Models foi descontinuada em 30 de julho de 2026. As referências abaixo são mantidas apenas como contexto histórico enquanto este lab é migrado. Use o Lab 013 ? Backends de LLM gratuitos para configurar o Foundry Local ou o Ollama antes de continuar.

Lab 029: LangChain & LangGraph Básico

Nível: L200 Trilha: ? Pro Code Tempo: ~60 min ? Custo: Gratuito ? GitHub Models nível gratuito

O que Você Vai Aprender

Construir um agente conversacional com LangChain e um loop de chamada de ferramentas
Modelar lógica de agente multi-etapas como um grafo de estado LangGraph Entender a diferença entre cadeias do LangChain e grafos do LangGraph Adicionar roteamento condicional: quando chamar uma ferramenta vs. retornar uma resposta Persistir estado da conversa com checkpoints do LangGraph

Introdução

LangChain é um dos frameworks open-source mais populares para construir aplicações alimentadas por LLM. LangGraph o estende com máquinas de estado explícitas ? grafos onde nós são funções e arestas são transições.

Quando usar cada um:

| LangChain | LangGraph | Melhor para | Pipelines lineares, cadeias RAG, agentes simples |
Agentes multi-etapas complexos, lógica de ramificação, ciclos | Estado | Implícito (passado pela cadeia) | Explícito (dict de estado tipado) | Loops | Não nativo | Suporte de primeira classe | Visibilidade | Logs da cadeia | Traces de execução do grafo |

Neste lab construímos o mesmo assistente de compras OutdoorGear de duas formas: primeiro com LangChain (mais simples), depois com LangGraph (controle mais explícito).

Pré-requisitos

[diagram or code sample]

Nenhuma assinatura Azure necessária ? usamos o endpoint compatível com OpenAI do GitHub Models:

[diagram or code sample]

Parte 1: Agente LangChain

Passo 1: Ferramentas

[diagram or code sample]

Passo 2: Agente LangChain com Chamada de Ferramentas

[diagram or code sample]

Execute: [diagram or code sample]

You should see the agent call search_products, inspect a result, then provide a recommendation.

Parte 2: Agente LangGraph

LangGraph modela o agente como uma máquina de estados. Isso torna a lógica explícita e testável.

Passo 3: Definir o estado do grafo

[diagram or code sample]

Passo 4: Definir os nós do grafo

[diagram or code sample]

Passo 5: Construir e executar o grafo

[diagram or code sample]

Parte 3: Adicionar Memória Persistente (Checkpointers)

LangGraph pode persistir o estado entre execuções usando um checkpointer ? é assim que você

constrói agentes multi-turno que lembram conversas:

[diagram or code sample]

? Verificação de Conhecimento

question "1. Qual é a principal vantagem do LangGraph sobre um agente LangChain simples?"

LangGraph usa uma máquina de estados explícita (grafo com nós e arestas) para modelar a lógica do agente. Isso torna ramificação, loops e roteamento condicional cidadãos de primeira classe ? visíveis, testáveis e depuráveis. Um agente LangChain esconde o fluxo de controle dentro do framework.

question "2. What does the add_messages reducer do in LangGraph state?" add_messages is a reducer function that tells LangGraph how to update the messages field: it appends new messages instead of replacing the whole list. Without it, each node return would overwrite the message history rather than adding to it.

question "3. Como o checkpointing do LangGraph habilita conversas multi-turno?" A checkpointer persists the graph state (all messages) to storage (memory, Redis, PostgreSQL) keyed by a thread_id. When you invoke the agent with the same thread_id, LangGraph loads the previous state and continues from where it left off ? the agent "remembers" prior turns without you managing history manually.

Resumo

Conceito | LangChain | LangGraph | Estrutura | Cadeia linear | Grafo dirigido (nós + arestas) | Loops | Não nativo | graph.add_edge("tools", "llm") | Ramificação | Limitada | add_conditional_edges() | State | Implicit | Explicit TypedDict | Memória | Manual | MemorySaver / PostgresSaver | Depuração | Logs da cadeia | Trace completo de execução do grafo |

Próximos Passos

Aprofundamento em chamada de funções: ? Lab 018 ? Function Calling & Tool Use Multi-agente

com Semantic Kernel: ? Lab 034 ? Multi-Agent Systems Agentes AutoGen em produção: ? Lab 040

? Production Multi-Agent

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

Supporting file: CHEATSHEET.md

Arquivos de apoio: Lab 029: LangChain & LangGraph Básico

Comece pelo caminho local ou gratuito documentado. Registre premissas e evidências de validação. Mantenha serviços hospedados opcionais, salvo quando o lab os exigir.

Supporting file: README.md

Lab 029: LangChain & LangGraph Básico

Use estes arquivos com o guia do lab e o kit gerado.