

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 029 : Bases de LangChain et LangGraph

Niveau : L200 Parcours : ? Pro Code Durée : ~60 min ? Coût : Gratuit ? Niveau gratuit  
GitHub Models

Ce que vous apprendrez

Construire un agent conversationnel avec LangChain et une boucle d'appel d'outils Modéliser la logique d'agent multi-étapes comme un graphe d'état LangGraph Comprendre la différence entre les chaînes LangChain et les graphes LangGraph Ajouter un routage conditionnel : quand appeler un outil vs retourner une réponse Persister l'état de conversation avec les checkpoints LangGraph

Introduction

LangChain est l'un des frameworks open-source les plus populaires pour construire des applications alimentées par des LLM. LangGraph l'étend avec des machines à états explicites ? des graphes où les nœuds sont des fonctions et les arêtes sont des transitions.

Quand utiliser chacun :

| LangChain | LangGraph | Idéal pour | Pipelines linéaires, chaînes RAG, agents simples | Agents multi-étapes complexes, logique de branchement, cycles | État | Implicite (passé à travers la chaîne) | Explicite (dictionnaire d'état typé) | Boucles | Pas natif | Support natif | Visibilité | Journaux de chaîne | Traces d'exécution du graphe |

Dans ce lab, nous construisons le même assistant shopping OutdoorGear de deux manières : d'abord avec LangChain (plus simple), puis avec LangGraph (contrôle plus explicite).

Prérequis

[diagram or code sample]

Pas besoin d'abonnement Azure ? nous utilisons le point de terminaison compatible OpenAI de GitHub Models :

[diagram or code sample]

Partie 1 : Agent LangChain

Étape 1 : Outils

[diagram or code sample]

Étape 2 : Agent LangChain avec appel d'outils

[diagram or code sample]

Exécutez-le : [diagram or code sample]

Vous devriez voir l'agent appeler search\_products, inspecter un résultat, puis fournir une recommandation.

Partie 2 : Agent LangGraph

LangGraph modélise l'agent comme une machine à états. Cela rend la logique explicite et testable.

Étape 3 : Définir l'état du graphe

[diagram or code sample]

Étape 4 : Définir les nœuds du graphe

[diagram or code sample]

Étape 5 : Construire et exécuter le graphe

[diagram or code sample]

Partie 3 : Ajouter la mémoire persistante (Checkpoint)

LangGraph peut persister l'état entre les exécutions grâce à un checkpoint ? c'est ainsi

que vous construisez des agents multi-tours qui se souviennent des conversations :

[diagram or code sample]

? Vérification des connaissances

question "1. Quel est le principal avantage de LangGraph par rapport à un simple agent LangChain ?" LangGraph utilise une machine à états explicite (graphe avec nœuds et arêtes) pour modéliser la logique de l'agent. Cela rend le branchement, les boucles et le routage conditionnel natifs ? visibles, testables et déboguables. Un agent LangChain masque le flux de contrôle à l'intérieur du framework.

question "2. Que fait le réducteur add\_messages dans l'état LangGraph ?" add\_messages est une fonction réducteur qui indique à LangGraph comment mettre à jour le champ messages : elle ajoute les nouveaux messages au lieu de remplacer la liste entière. Sans elle, chaque retour de nœud écraserait l'historique des messages au lieu de l'enrichir.

question "3. Comment le checkpointing LangGraph permet-il les conversations multi-tours ?"

Un checkpointer persiste l'état du graphe (tous les messages) dans un stockage (mémoire, Redis, PostgreSQL) indexé par un thread\_id. Lorsque vous invoquez l'agent avec le même thread\_id, LangGraph charge l'état précédent et continue là où il s'était arrêté ? l'agent « se souvient » des tours précédents sans que vous ayez à gérer l'historique manuellement.

Résumé

Concept | LangChain | LangGraph | Structure | Chaîne linéaire | Graphe orienté (nœuds + arêtes) | Boucles | Pas natif | graph.add\_edge("tools", "llm") | Branchement | Limité | add\_conditional\_edges() | État | Implicite | TypedDict explicite | Mémoire | Manuelle | MemorySaver / PostgresSaver | Débogage | Journaux de chaîne | Trace d'exécution complète du graphe |

Prochaines étapes

Plongée approfondie dans l'appel de fonctions : ? Lab 018 ? Appel de fonctions et utilisation d'outils Multi-agents avec Semantic Kernel : ? Lab 034 ? Systèmes multi-agents Agents AutoGen en production : ? Lab 040 ? Multi-agent en production

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

Supporting file: CHEATSHEET.md

Fichiers d'accompagnement: Lab 029 : Bases de LangChain et LangGraph

Commencez par le parcours local ou gratuit documenté. Consignez les hypothèses et les preuves de validation. Gardez les services hébergés facultatifs sauf exigence explicite du lab.

Supporting file: README.md

Lab 029 : Bases de LangChain et LangGraph

Utilisez ces fichiers avec le guide du lab et le kit généré.