생성형 AI 기반 RAG 개발자 과정

Agent + RAG 서비스 결합

Session 30 / 33 — Day 4

문서검색 챗봇 → 도구를 활용하는 비서로 확장

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

이번 시간을 마치면

🔧 결합 방법

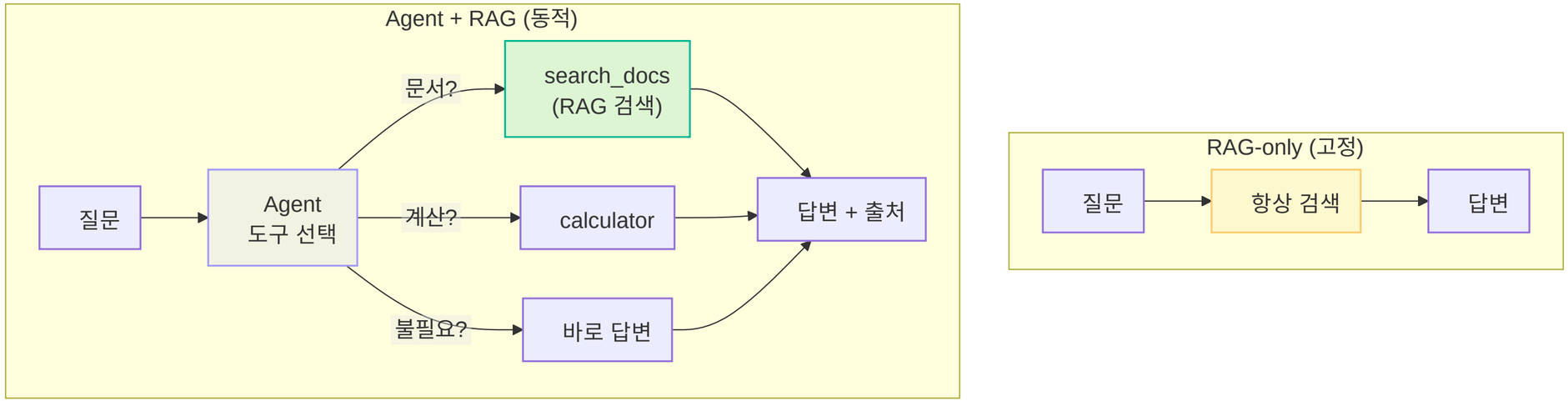
- **retriever 를 tool 로** 노출
- RAG 검색 = 여러 도구 중 하나
- LLM 이 **도구 선택**

🏗️ 서비스로 (최종 프로젝트)

- 작은 RAG + 검색·계산 도구 + **메모리**
- 멀티툴 라우팅 → 통합 **웹 데모**
- 실행 과정을 **trace** 로 시각화

🎯 목표: 문서검색 전용 봇 → 필요한 도구를 선택하는 비서로 전환.

고정 경로 → 동적 선택



💡 핵심 차이: RAG 는 **항상 검색**, Agent 는 **필요 시에만** RAG 도구를 호출.

RAG 검색을 @tool 로 노출 (스니펫)

```
from langchain_core.tools import tool

@tool
def search_docs(query: str) → str:
    """기술 문서에서 질문과 관련된 내용을 검색한다
    (SSD, RAID 등 저장장치 주제)."""
    hits = vectorstore.similarity_search(query, k=3)
    return "\n".join(f"- {d.page_content}" for d in hits) \
        or "관련 문서를 찾을 수 없습니다."
```

- **docstring 이 곧 도구 설명** → LLM 이 도구 사용 시점을 판단
- 소규모 RAG(`Chroma.from_documents`)를 검색 도구로 그대로 재사용

> 🔑 RAG 를 별도 개념이 아닌 **도구 하나**로 취급한다. 동일한 방식으로 `calculator` 도구도 추가한다.

도구 + 메모리 결합으로 Agent 구성 (스니펫)

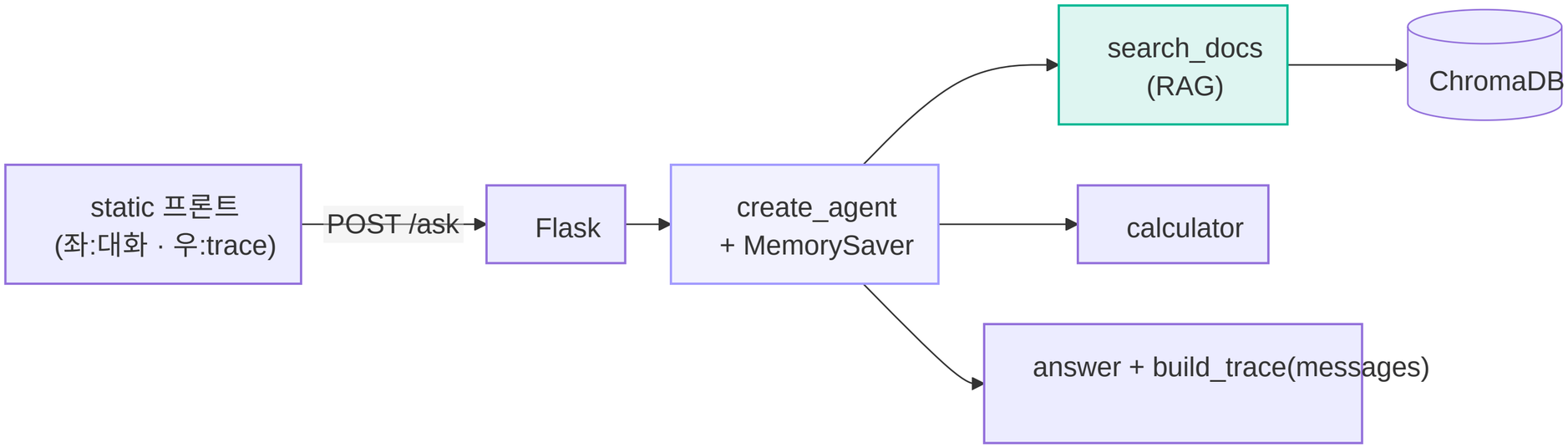
```
agent = create_agent(
    llm, [search_docs, calculator],
    system_prompt="저장장치 질문은 search_docs 로 근거를 찾고, "
                  "계산은 calculator 를 써라. 문서에 없으면 모른다고 답하라.",
    checkpointer=MemorySaver())  # thread_id 로 대화 기억

# LLM 이 질문 보고 도구 선택
config = {"configurable": {"thread_id": "console"}}
agent.invoke({"messages": [("user",
    "NVMe와 SATA 속도를 더하면?")]}, config=config)
# → search_docs(속도 찾기) + calculator(덧셈) 순서대로 호출
```

- 단일 질문에 **여러 도구**를 순차 호출 가능 (ReAct 루프)
- 문서에서 숫자 검색 → 계산 — **검색·계산 자동 분담**, `thread_id` 로 맥락 유지

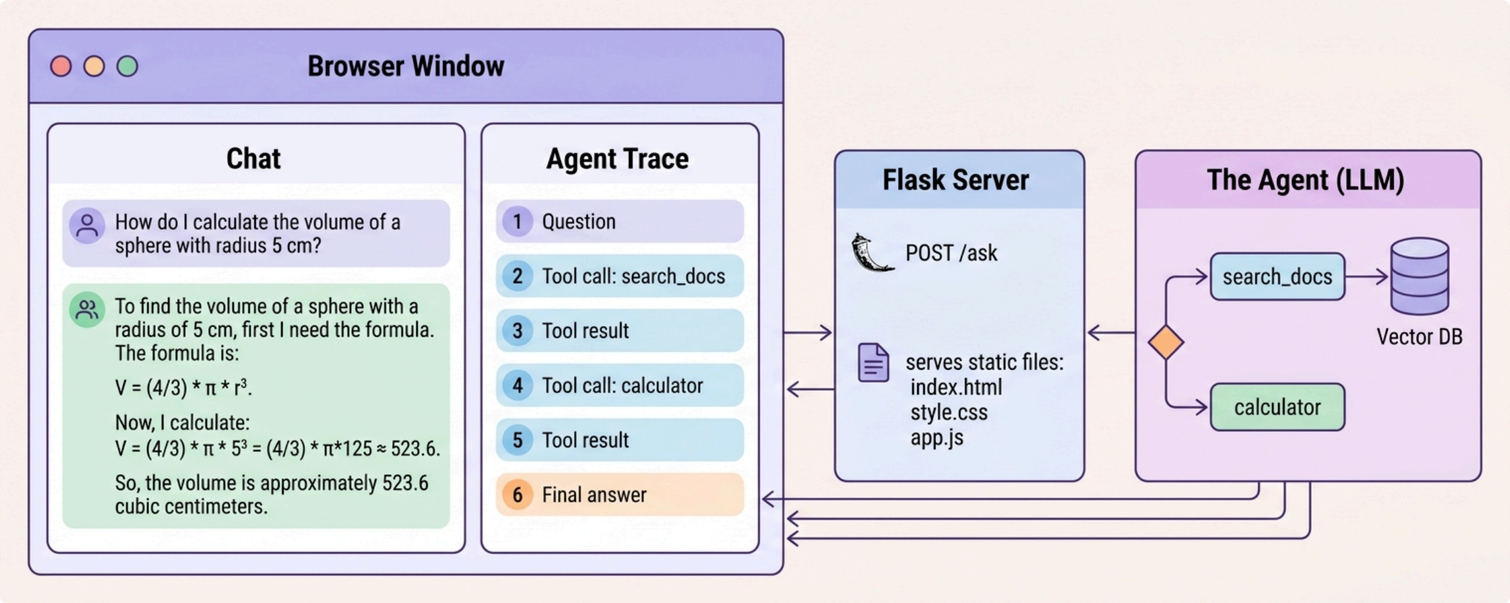
> 📁 참고: `8.agents/3.applied_agents/`, `7.RAG/6.agentic/`

기존 서비스에 Agent 계층 추가



💡 `/ask` 가 `agent.invoke(...)` 한 줄을 호출한다. 응답에 답변과 **trace**(질문→도구 호출→도구 결과→최종 답변)를 함께 담아 우측 패널에 시각화한다.

최종 프로젝트: trace 시각화 웹앱



통합 웹 데모 · 브라우저(좌측 대화 · 우측 에이전트 trace 패널) ↔ Flask /ask ↔ 에이전트가 search_docs·calculator 도구를 골라 실행, 단계를 trace로 반환

💡 `build_trace(result["messages"])` 가 "질문 → 판단·도구 호출 → 도구 결과 → 최종 답변"을 단계 카드로 분해해, **블랙박스였던 도구 선택 과정**을 가시화한다.

자율성과 통제의 균형

⚠️ 트레이드오프

- 도구 증가에 따른 **지연·비용 상승**
- 부적절한 도구 선택 가능성
- 무한 루프 위험 (max iterations)

✅ 통제 레버

- 도구 **description 정교화**
- **max_iterations** 제한
- 도구 호출 **로그·추적**
- 단순 질문은 RAG 직답, 복잡한 경우에만 Agent

> 🔑 Agent 가 항상 정답은 아니다 — **단순함 우선**, 자율성은 필요한 경우에 한정.

핵심 5가지

1. ✅ RAG 검색을 `@tool` **로 노출**(`search_docs`) → 도구 중 하나로
2. ✅ docstring = 도구 설명 → LLM 이 **선택 판단**
3. ✅ `search_docs` + `calculator` + **메모리**(`MemorySaver` / `thread_id`)로 조립
4. ✅ `agent.invoke` 한 줄을 **Flask** `/ask` 로 감싸 통합 웹 데모
5. ✅ `build_trace` 로 도구 선택 과정을 **trace 시각화**(좌:대화·우:패널)