# 차세대 AI 에이전트 오케스트레이션: Claude Code CLI 고도화 및 LangGraph 기반 회의 인텔리전스 플랫폼 아키텍처 가이드

## 서론: 이원화된 에이전트 생태계의 통합적 접근

현대 소프트웨어 엔지니어링의 패러다임은 정적인 도구 사용에서 능동적인 에이전트 협업으로 급격히 이동하고 있다. 이러한 변화의 최전선에는 두 가지 핵심 축이 존재한다. 하나는 개발자의 로컬 환경에서 코딩, 테스팅, 배포를 자율적으로 수행하는 터미널 기반 에이전트인 **Claude Code**이며, 다른 하나는 기업의 복잡한 비즈니스 프로세스를 상태 기반(Stateful)으로 처리하는 서버 사이드 멀티 에이전트 프레임워크인 **LangGraph**이다. 본 보고서는 이 두 기술의 심층적인 기술적 구성 방법론을 다루며, 특히 로컬 개발 생산성을 극대화하기 위한 Claude Code의 서브 에이전트(Sub-agent) 및 스킬(Skill) 설정 전략과, 이를 통해 개발된 기능을 실제 서비스로 구현하기 위한 LangGraph 기반 회의 서기(Meeting Secretary) 플랫폼의 엔드투엔드(End-to-End) 아키텍처를 제시한다.

우리는 먼저 Claude Code의 CLI 환경 내에서 nO 루프(Loop)라 불리는 단일 스레드 마스터 루프가 어떻게 컨텍스트를 관리하고 도구를 호출하는지 분석하고, 이를 바탕으로 YAML 기반의 서브 에이전트 구성과 '점진적 공개(Progressive Disclosure)' 원칙에 입각한 스킬 설계 패턴을 상술한다. 이어서, 회의 녹음 파일의 대용량 업로드 처리를 위한 Tus 프로토콜과 FastAPI의 통합, 그리고 인간의 검토(Human-in-the-Loop)를 필수적으로 요구하는 회의록 생성 및 액션 아이템 실행 워크플로우를 LangGraph의 순환 그래프 구조로 구현하는 방안을 논한다. 특히, 두 영역을 관통하는 \*\*Model Context Protocol (MCP)\*\*의 활용 전략은 로컬 개발 도구와 프로덕션 서비스 간의 기술적 간극을 좁히는 핵심 열쇠로 다루어질 것이다.

## 제1부: Claude Code CLI 기반 자율 개발 환경 구축 전략

### 1.1 Claude Code의 운영 아키텍처와 컨텍스트 관리

Claude Code는 단순한 챗봇 인터페이스를 넘어, 개발자의 터미널 환경에 상주하며 파일 시스템, 셸(Shell), 그리고 다양한 개발 도구에 직접 접근할 수 있는 권한을 가진 자율 에이전트이다. 이 시스템의 핵심은 "컨텍스트 수집 → 행동 수행 → 결과 검증 → 반복"으로 이어지는 피드백 루프에 있다.1 이 루프는 단일 스레드로 동작하며, 복잡한 병렬 처리 대신 명확한 디버깅과 추적 가능성을 우선시하는 설계를 채택하고 있다.2

그러나 단일 컨텍스트 윈도우(Context Window)는 대규모 프로젝트에서 필연적으로 '오염(Contamination)' 문제를 야기한다. 방대한 코드베이스를 모두 읽어들이거나, 서로 다른 성격의 작업(예: 보안 감사와 기능 구현)을 하나의 세션에서 수행할 경우, 모델의 주의력(Attention)이 분산되고 토큰 비용이 급증한다. 이를 해결하기 위해 Anthropic은 \*\*서브 에이전트(Sub-agent)\*\*와 \*\*스킬(Skill)\*\*이라는 두 가지 모듈화 메커니즘을 도입하였다.

#### 1.1.1 구성 계층과 범위(Scope)의 기술적 함의

Claude Code의 설정 시스템은 사용자(User), 프로젝트(Project), 로컬(Local) 등 다층적인 스코프(Scope)를 통해 제어된다. 이는 팀 단위의 표준화된 개발 환경과 개인의 맞춤형 설정을 조화시키기 위한 아키텍처적 장치이다.

**표 1: Claude Code 설정 스코프 및 우선순위 분석**

| **우선순위** | **스코프 유형** | **파일 경로 예시** | **기술적 특징 및 용도** |
| --- | --- | --- | --- |
| **1 (최상위)** | **Managed** | /etc/claude-code/ (Linux), System Dir | IT 관리자에 의해 배포되는 강제 정책. 보안 규정(예: 특정 MCP 서버 차단)을 엔지니어 레벨에서 우회하지 못하도록 설정.3 |
| **2** | **CLI Flags** | 런타임 인수 (예: --mcp-config) | 일회성 세션이나 CI/CD 파이프라인 스크립트 실행 시 특정 설정을 강제 주입하기 위해 사용.4 |
| **3** | **Local Project** | .claude/settings.local.json | 프로젝트 내에서 개인 개발자에게만 적용되는 설정. Git에 커밋되지 않으므로 개인적인 경로 설정이나 임시 테스트 도구 포함에 적합.5 |
| **4** | **Project** | .claude/settings.json | 팀 전체가 공유하는 표준 설정. 린트(Lint) 규칙, 허용된 도구 목록 등이 포함되며 버전 관리 시스템(VCS)에 커밋됨.3 |
| **5 (최하위)** | **User** | ~/.claude/settings.json | 모든 프로젝트에 적용되는 개발자의 전역 설정. 개인적인 서브 에이전트나 테마, 선호하는 모델 설정.3 |

이러한 계층 구조는 CLAUDE.md 메모리 파일과 서브 에이전트 정의에도 동일하게 적용된다. 예를 들어, 팀 리더는 프로젝트 루트의 .claude/agents/에 code-reviewer 에이전트를 정의하여 팀원들이 동일한 기준(예: 사내 코딩 컨벤션 준수 여부)으로 코드 리뷰를 수행하도록 강제할 수 있다.6 반면, 개별 개발자는 자신의 홈 디렉토리 ~/.claude/agents/에 개인적인 daily-planner 에이전트를 두어 업무 효율을 높일 수 있다.

### 1.2 서브 에이전트(Sub-agent)의 심층 구성 및 설계 패턴

서브 에이전트는 독립된 컨텍스트 윈도우를 가진 전문화된 작업자이다. 메인 대화 세션에서 서브 에이전트가 호출되면, 현재의 컨텍스트 중 필요한 부분만이 서브 에이전트에게 전달되며, 서브 에이전트의 작업이 완료되면 그 결과(요약된 산출물)만이 메인 세션으로 반환된다. 이는 토큰 효율성을 극대화하고, 작업의 격리성을 보장한다.6

#### 1.2.1 YAML 프론트매터(Frontmatter) 기반 설정 문법

서브 에이전트는 Markdown 파일로 정의되며, 상단에 YAML 프론트매터를 포함한다. 이 프론트매터는 에이전트의 메타데이터와 권한을 제어하는 핵심 영역이다.

YAML

---  
name: security-auditor # CLI나 메인 에이전트가 호출할 때 사용하는 식별자 (소문자, 하이픈 권장)  
description: >  
 전문 보안 감사자로서 코드를 분석하여 OWASP Top 10 취약점을 식별합니다.  
 코드 변경 전이나 보안 검토 요청 시 사용하십시오.  
tools: # 최소 권한 원칙(Least Privilege)에 따른 도구 화이트리스트  
 - Read  
 - Grep  
 - Glob  
 - mcp\_\_jira\_\_create\_issue # MCP를 통해 Jira 이슈 생성 권한만 부여 (삭제 권한 제외)  
model: sonnet # 작업 성격에 따른 모델 선택 (opus: 복잡한 추론, haiku: 단순 반복, sonnet: 균형)  
color: "#ff0000" # UI 식별을 위한 색상 코드  
---  
# 시스템 프롬프트 (System Prompt)  
  
당신은 10년 경력의 시니어 보안 엔지니어입니다. 사용자의 코드를 분석할 때 다음 프로토콜을 엄격히 준수하십시오.  
  
## 분석 프로토콜  
1. \*\*입력 검증(Input Validation)\*\*: 모든 외부 입력 지점에 대한 검증 로직 부재를 확인합니다.  
2. \*\*인증 및 인가(AuthN/AuthZ)\*\*: 하드코딩된 자격 증명이나 취약한 접근 제어를 탐지합니다.  
3. \*\*데이터 보호\*\*: 민감 정보(PII)의 평문 저장 여부를 감사합니다.  
  
## 보고 형식  
발견된 취약점은 다음 Markdown 테이블 형식으로 보고하십시오:  
  
| 심각도 | 파일명 | 라인 | 취약점 유행 | 권고 사항 |

출처: 6 재구성

#### 1.2.2 도구(Tool) 할당과 권한 제어 전략

tools 필드는 서브 에이전트 설계에서 가장 중요한 보안 장치이다. 만약 이 필드를 생략하면 서브 에이전트는 메인 세션이 가진 모든 권한(파일 수정, 셸 명령어 실행 등)을 상속받는다.6 그러나 security-auditor와 같은 감사 에이전트에게는 코드 수정 권한(Edit, Write)이나 임의의 셸 실행 권한(Bash)을 부여하지 않는 것이 바람직하다. 오직 읽기(Read, Grep)와 보고(mcp\_\_jira\_\_create\_issue) 권한만을 부여함으로써, 에이전트가 의도치 않게 코드를 망가뜨리는 것을 원천적으로 차단할 수 있다.9

**주요 에이전트 유형별 도구 구성 예시:**

1. **리서처(Researcher):**
   * tools:
   * 목적: 외부 문서를 검색하고 내부 코드와 비교 분석하지만, 코드를 수정하지는 않음.
2. **구현자(Implementer):**
   * tools:
   * 목적: 실제 코드를 작성하고, 테스트를 실행(Bash)하여 기능을 검증함.
3. **기획자(Product Owner):**
   * tools:
   * 목적: 기획 문서를 읽고 기존 이슈를 확인하여 요구사항을 구체화함.10

#### 1.2.3 미니언(Minion) 패턴과 체이닝(Chaining)

효과적인 개발 워크플로우를 위해 여러 서브 에이전트를 순차적으로 연결하는 '미니언 패턴'이 권장된다. 예를 들어, 기능 구현 요청이 들어오면 먼저 pm-agent가 요구사항을 명세화하고, 그 결과를 architect-agent가 설계하며, 최종적으로 coder-agent가 구현하는 방식이다.9

Claude Code는 서브 에이전트가 다른 서브 에이전트를 직접 생성(Spawn)하는 것을 허용하지 않지만, 메인 컨텍스트에서 훅(Hook)을 사용하여 이를 오케스트레이션할 수 있다. settings.json의 hooks 설정을 통해 특정 에이전트의 작업이 종료(SubagentStop)되었을 때, 다음 단계의 에이전트를 호출하도록 유도하는 메시지를 출력하거나 스크립트를 실행할 수 있다.10

### 1.3 스킬(Skill)의 점진적 공개(Progressive Disclosure) 아키텍처

스킬은 서브 에이전트와 달리 독립된 페르소나보다는 특정 작업을 수행하기 위한 지식과 절차의 집합체이다. 스킬 시스템의 핵심 철학은 \*\*점진적 공개(Progressive Disclosure)\*\*이다.11

#### 1.3.1 스킬 디렉토리 구조 및 SKILL.md

스킬은 단순한 텍스트 파일이 아니라, 디렉토리 단위로 관리되는 지식 베이스이다.

~/.claude/skills/

└── flutter-architecture/

├── SKILL.md # 메타데이터 및 진입점

├── rules/

│ ├── clean\_architecture.md

│ └── bloc\_pattern.md

└── templates/

├── use\_case.dart

└── repository.dart

SKILL.md 파일은 메타데이터와 개략적인 지침만을 포함한다. Claude는 세션 시작 시 이 파일의 name과 description만을 읽어 컨텍스트에 로드한다. 이는 수십 개의 스킬이 있어도 초기 토큰 사용량을 최소화할 수 있게 해준다.

## name: flutter-architecture description: Flutter 앱 개발 시 Clean Architecture와 BLoC 패턴을 적용하기 위한 가이드라인과 템플릿을 제공합니다.

# Flutter 아키텍처 스킬

사용자가 새로운 기능을 구현하거나 리팩토링을 요청할 때 이 스킬을 사용하십시오.

## 절차

1. 요구사항이 데이터 계층을 포함하는지 분석합니다.
2. rules/clean\_architecture.md를 참조하여 레이어를 구분하십시오.
3. 상태 관리가 필요한 경우 rules/bloc\_pattern.md의 규칙을 따르고, templates/ 폴더의 코드를 활용하여 구현하십시오.  
   출처: 12

#### 1.3.2 동적 로딩 메커니즘

사용자가 "이 기능을 BLoC 패턴으로 구현해줘"라고 요청하면, Claude는 description과의 의미적 유사성을 판단하여 flutter-architecture 스킬을 활성화한다. 이때 비로소 SKILL.md의 전체 내용이 로드되며, Claude는 지침에 따라 rules/bloc\_pattern.md 파일을 추가로 읽어들일지 결정한다.11 이처럼 필요한 시점에 필요한 파일만 읽어들이는 방식은 CLAUDE.md에 모든 내용을 때려넣는 방식(Embedded)에 비해 월등히 높은 효율성과 정확성을 보장한다.14

### 1.4 MCP (Model Context Protocol) 통합 및 설정

Claude Code는 그 자체로 MCP 클라이언트로 동작하여 외부 도구와 데이터를 CLI 환경으로 가져올 수 있다. 이는 개발자가 IDE를 벗어나지 않고 Jira 티켓을 생성하거나, 데이터베이스 스키마를 조회하는 것을 가능하게 한다.

#### 1.4.1 CLI를 통한 MCP 서버 추가 및 보안

가장 간단한 설정 방법은 claude mcp add 명령어를 사용하는 것이다.

Bash

claude mcp add jira --transport http https://mcp.atlassian.com/v1/mcp \  
 --header "Authorization: Bearer <YOUR\_PAT\_TOKEN>" \  
 --scope project

출처: 5

여기서 --scope project 플래그는 해당 설정을 현재 프로젝트의 .mcp.json 파일에 저장한다는 의미이다. 이 파일은 팀원들과 공유될 수 있으므로(Git 커밋 가능), API 키와 같은 민감 정보는 절대 직접 기록되어서는 안 된다. 대신 환경 변수나 헤더 주입 방식을 사용하여, 각 개발자가 자신의 로컬 환경 변수(JIRA\_API\_TOKEN 등)를 통해 인증하도록 구성해야 한다.15

#### 1.4.2 로컬 및 원격 MCP 서버 구성

로컬에서 개발 중인 MCP 서버(예: 사내 DB 접속용 Python 스크립트)를 연결할 때는 uv나 docker를 활용한 실행 구성을 claude\_desktop\_config.json (또는 프로젝트별 .mcp.json)에 정의한다.

JSON

{  
 "mcpServers": {  
 "internal-db": {  
 "command": "uv",  
 "args": [  
 "run",  
 "--with", "mcp[cli]",  
 "path/to/db\_server.py"  
 ],  
 "env": {  
 "DB\_CONNECTION\_STRING": "${DB\_URL}"  
 }  
 }  
 }  
}

출처: 3

Google Calendar와 같이 OAuth 인증이 필요한 서버의 경우, Claude Code는 초기 연결 시 브라우저를 통한 인증 흐름을 지원하며, 발급받은 토큰은 로컬 파일(예: token.json)에 저장된다. 이 파일은 반드시 .gitignore에 포함되어야 한다.18

## 제2부: LangGraph 기반 멀티 에이전트 회의 서기 플랫폼 아키텍처

로컬 개발 환경에서의 에이전트 활용을 넘어, 이제 우리는 실제 비즈니스 가치를 창출하는 **회의 서기(Meeting Secretary) 플랫폼**의 아키텍처를 설계한다. 이 플랫폼은 사용자가 업로드한 회의 녹음 파일을 텍스트로 변환(Transcribe)하고, 내용을 분석(Analyze)하여 회의록을 작성하며, 승인된 액션 아이템을 외부 시스템(Jira, Calendar)에 등록(Execute)하는 엔드투엔드 워크플로우를 수행한다.

이 시스템은 장시간 실행되는 비동기 프로세스와 인간의 개입(Human-in-the-Loop)을 필요로 하므로, 상태 관리(State Management)와 영속성(Persistence)이 보장되는 **LangGraph** 프레임워크가 최적의 솔루션이다.

### 2.1 시스템 아키텍처 오버뷰 (Microservices & Gateway)

전체 시스템은 **FastAPI**를 게이트웨이 및 백엔드 서버로 사용하고, **LangGraph**를 비즈니스 로직 오케스트레이터로, **PostgreSQL**을 상태 저장소로 활용하는 구조를 가진다.

**주요 컴포넌트:**

1. **Ingestion Layer (FastAPI + Tus):** 대용량 오디오 파일의 안정적인 업로드를 담당.
2. **Processing Layer (LangGraph):** 전사, 분석, 검토, 실행의 4단계 에이전트 워크플로우 제어.
3. **Persistence Layer (PostgreSQL):** LangGraph의 체크포인트(Checkpoint)를 저장하여 워크플로우의 일시 정지 및 재개를 지원.
4. **Integration Layer (MCP Clients):** 외부 도구(Jira, G-Cal)와의 통신을 표준화된 MCP 프로토콜로 처리.

### 2.2 대용량 오디오 인제스션: Tus 프로토콜과 FastAPI 통합

회의 녹음 파일은 수백 메가바이트(MB)에서 기가바이트(GB) 단위에 이르기 때문에, 일반적인 HTTP POST 요청으로는 타임아웃이나 네트워크 불안정으로 인한 실패 가능성이 높다. 따라서 재개 가능한 업로드(Resumable Upload)를 지원하는 **Tus 프로토콜**의 도입이 필수적이다.20

#### 2.2.1 FastAPI에서의 Tus 서버 구현

Python 생태계에서는 fastapi-tusd 또는 tuspyserver 라이브러리를 통해 이를 구현할 수 있다.

Python

from fastapi import FastAPI, Depends  
from fastapi\_tusd import TusRouter  
import os  
  
app = FastAPI()  
  
# 파일 저장 경로 설정  
UPLOAD\_DIR = "./meeting\_uploads"  
os.makedirs(UPLOAD\_DIR, exist\_ok=True)  
  
# Tus 라우터 등록  
# 클라이언트는 /files 엔드포인트를 통해 HEAD, PATCH, POST 요청으로 파일을 청크 단위로 전송  
tus\_router = TusRouter(store\_dir=UPLOAD\_DIR, location="/files")  
app.include\_router(tus\_router, prefix="/files")  
  
# 업로드 완료 후 후처리를 위한 훅(Hook)  
@app.post("/meetings/process/{file\_id}")  
async def start\_meeting\_processing(file\_id: str, background\_tasks: BackgroundTasks):  
 file\_path = os.path.join(UPLOAD\_DIR, file\_id)  
 if not os.path.exists(file\_path):  
 raise HTTPException(status\_code=404, detail="File not found")  
   
 # LangGraph 워크플로우 비동기 시작  
 # thread\_id는 회의 ID와 매핑되어 상태 관리에 사용됨  
 thread\_id = str(uuid.uuid4())  
 config = {"configurable": {"thread\_id": thread\_id}}  
   
 # 초기 상태 주입 및 그래프 실행 시작  
 background\_tasks.add\_task(  
 graph.ainvoke,   
 {"audio\_file\_path": file\_path, "meeting\_id": file\_id},   
 config  
 )  
   
 return {"status": "processing\_started", "thread\_id": thread\_id}

출처: 20 종합

이 구조에서 클라이언트(Flutter 앱 등)는 tus\_client를 사용하여 파일을 업로드하고, 업로드가 완료되면 /meetings/process 엔드포인트를 호출하여 LangGraph 에이전트를 깨운다.24

### 2.3 LangGraph 스키마 설계 및 상태 관리

LangGraph의 핵심은 노드 간에 공유되는 \*\*State(상태)\*\*이다. 이 상태 객체는 회의록 데이터, 액션 아이템 리스트, 그리고 인간의 피드백을 모두 담아야 한다. Python의 TypedDict나 Pydantic 모델을 사용하여 엄격한 타입을 정의한다.

Python

from typing import TypedDict, Annotated, List, Optional, Union  
import operator  
from pydantic import BaseModel, Field  
  
# 액션 아이템 구조 정의  
class ActionItem(BaseModel):  
 id: str  
 description: str  
 assignee: str  
 priority: str  
 tool\_call\_payload: Optional[dict] = None # MCP 도구 호출을 위한 페이로드 (예: Jira 이슈 생성 JSON)  
 status: str = "pending" # pending, approved, rejected  
  
# 그래프 전체 상태 정의  
class MeetingState(TypedDict):  
 # 메시지 기록은 operator.add를 사용하여 덮어쓰지 않고 추가됨 (대화형 인터페이스 지원)  
 messages: Annotated[List[AnyMessage], operator.add]  
   
 # 컨텍스트 데이터  
 meeting\_id: str  
 audio\_file\_path: str  
 transcript: Optional[str]  
   
 # 분석 결과  
 minutes\_markdown: Optional[str]  
 action\_items: List[ActionItem]  
   
 # 인간 검토 상태  
 review\_status: str # 'awaiting\_review', 'approved', 'revision\_requested'  
 human\_feedback: Optional[str]

출처: 25

### 2.4 멀티 에이전트 노드 구현 및 워크플로우

워크플로우는 **Transcriber -> Analyst -> Human Review -> Executor**의 순차적 흐름을 기본으로 하되, 검토 단계에서 거절(Reject)될 경우 다시 Analyst로 돌아가는 순환 구조를 가진다.

#### 2.4.1 Transcriber 노드 (STT 통합)

STT(Speech-to-Text) 엔진(예: Deepgram, OpenAI Whisper)을 호출하여 오디오를 텍스트로 변환한다. 화자 분리(Speaker Diarization) 기능이 포함된 모델을 사용하여 "누가 무엇을 말했는지" 정확히 파악하는 것이 중요하다.27

#### 2.4.2 Analyst 노드 (구조적 데이터 추출)

이 노드는 LLM을 사용하여 텍스트에서 회의록을 요약하고 액션 아이템을 추출한다. 여기서 중요한 것은 단순 텍스트가 아니라, 시스템이 실행 가능한 \*\*구조화된 데이터(JSON)\*\*를 생성하도록 프롬프트를 설계하는 것이다.30

시스템 프롬프트 예시:

"당신은 전문 회의 서기입니다. 전사된 텍스트를 분석하여 다음을 수행하십시오.

1. 핵심 안건과 결정 사항을 포함한 마크다운 요약본 작성.
2. 각 액션 아이템을 추출하고, 이를 실행하기 위한 적절한 도구(Jira, Calendar 등)를 매핑하십시오.
3. 출력은 반드시 MeetingState 스키마의 action\_items 형식을 따라야 합니다."

#### 2.4.3 Human-in-the-Loop: Reviewer 노드와 Interrupt 패턴

AI가 생성한 액션 아이템(예: "CEO에게 메일 보내기")을 사용자 승인 없이 실행하는 것은 위험하다. 따라서 LangGraph의 interrupt 메커니즘을 사용하여 실행을 일시 중지하고 사용자 입력을 기다려야 한다.

최신 LangGraph 버전에서는 interrupt() 함수를 사용하여 노드 실행 중간에 제어권을 반환할 수 있다.32

Python

from langgraph.types import interrupt, Command  
  
def human\_review\_node(state: MeetingState):  
 # 현재 생성된 회의록과 액션 아이템을 사용자에게 검토 요청  
 # 이 시점에서 그래프 실행은 멈추고(Suspend), 상태는 DB에 저장됨  
 user\_decision = interrupt({  
 "type": "review\_request",  
 "minutes": state["minutes\_markdown"],  
 "proposed\_actions": [item.dict() for item in state["action\_items"]]  
 })  
   
 # 사용자가 API를 통해 Command(resume=...)를 보내면 여기서부터 실행 재개  
 # user\_decision 변수에 사용자가 보낸 데이터가 담김  
   
 if user\_decision["action"] == "approve":  
 # 사용자가 수정한 액션 아이템이 있다면 업데이트  
 updated\_actions = user\_decision.get("updated\_actions", state["action\_items"])  
 return {  
 "review\_status": "approved",   
 "action\_items": updated\_actions  
 }  
 else:  
 # 거절 시 피드백을 반영하여 다시 분석 요청  
 return {  
 "review\_status": "revision\_requested",  
 "human\_feedback": user\_decision.get("feedback")  
 }

#### 2.4.4 Executor 노드 (MCP 클라이언트 구현)

승인된 액션 아이템을 실행하는 단계이다. LangGraph 노드 내부에서 **MCP Client**를 인스턴스화하여 외부 MCP 서버(Jira 등)에 요청을 보낸다. 이는 1부에서 다룬 CLI의 MCP 연결과 개념적으로 동일하나, 서버 사이드에서 프로그래매틱하게 처리된다는 점이 다르다.16

Python

# 가상의 MCP 클라이언트 사용 예시  
async def executor\_node(state: MeetingState):  
 results =  
 for action in state["action\_items"]:  
 if action.status == "pending":  
 # Jira MCP 서버 도구 호출  
 if action.tool\_call\_payload["tool"] == "jira\_create\_issue":  
 # mcp\_client는 사전에 연결 설정됨  
 result = await mcp\_client.call\_tool(  
 "jira\_create\_issue",   
 action.tool\_call\_payload["args"]  
 )  
 results.append(result)  
 return {"execution\_results": results}

### 2.5 상태 영속성과 스케일링 전략

HITL 워크플로우에서는 시스템이 사용자 입력을 몇 시간, 심지어 며칠 동안 기다려야 할 수 있다. 따라서 메모리 기반 저장소(InMemorySaver)는 사용할 수 없으며, 반드시 **PostgreSQL**과 같은 영구 저장소를 사용하는 체크포인터(AsyncPostgresSaver)를 설정해야 한다.32

* **Thread ID 관리:** 각 회의(Meeting)는 고유한 thread\_id를 가지며, 모든 API 요청(상태 조회, 재개 등)은 이 ID를 기준으로 라우팅된다.
* **비동기 처리:** FastAPI와 LangGraph의 모든 I/O 작업(DB 접근, LLM 호출, MCP 통신)은 async/await을 통해 비차단(Non-blocking) 방식으로 처리되어야 높은 동시성을 감당할 수 있다.

## 결론 및 제언

본 보고서에서 제시한 아키텍처는 Claude Code를 통한 개발자 경험(DX)의 혁신과 LangGraph를 통한 사용자 경험(UX)의 혁신을 동시에 추구한다.

1. **개발 환경(DevEnv):** 개발자는 ~/.claude/agents에 정의된 **서브 에이전트**를 통해 반복적인 코딩 업무를 위임하고, **MCP**를 통해 로컬 터미널을 엔터프라이즈 도구(Jira, GitHub)와 연결하여 '컨텍스트 스위칭' 없는 몰입형 개발을 수행한다.
2. **서비스 환경(ProdEnv):** **LangGraph**로 구축된 회의 서기 플랫폼은 상태 기반의 복잡한 워크플로우를 안정적으로 처리하며, **Tus 프로토콜**과 **HITL** 패턴을 통해 기술적 안정성과 결과물의 신뢰성을 모두 확보한다.

특히 주목할 점은 **MCP의 보편성**이다. 로컬 CLI 환경에서 사용하던 MCP 서버(예: Jira Integration)를 그대로 프로덕션 LangGraph 노드에서 재사용할 수 있다는 점은, 향후 에이전트 생태계가 도구(Tool) 중심의 마이크로서비스 아키텍처로 수렴할 것임을 시사한다. 따라서 조직은 독자적인 API 연동 코드를 작성하는 대신, 표준화된 MCP 서버를 구축하고 이를 내/외부 에이전트가 공용으로 사용하도록 하는 전략을 수립해야 할 것이다.

## 데이터 요약 및 비교

**표 2: 에이전트 기술 스택 비교 (Claude Code vs. LangGraph)**

| **특징** | **Claude Code (CLI)** | **LangGraph (Platform)** |
| --- | --- | --- |
| **주 사용처** | 로컬 개발, 일회성 작업, 코드 리팩토링 | 엔터프라이즈 서비스, 장기 실행 프로세스 |
| **상태 관리** | 세션 기반 (일시적), CLAUDE.md로 컨텍스트 보존 | 체크포인트 기반 (영구적), DB에 상태 저장 |
| **에이전트 구조** | 메인 루프 + 서브 에이전트 (Sub-agent) | 노드(Node)와 엣지(Edge)로 구성된 그래프 |
| **도구 연결** | MCP Client (로컬 프로세스/HTTP) | MCP Client (HTTP/SSE), 커스텀 Python 함수 |
| **인간 개입 (HITL)** | 대화형 프롬프트 (Y/N 승인) | interrupt 함수 및 상태 수정 API |
| **설정 방식** | YAML/Markdown 파일 (~/.claude/) | Python 코드 (Graph Definition) |

(본 보고서는 제공된 리서치 자료 6 ~ 5의 내용을 바탕으로 재구성 및 심층 분석되었습니다.)

#### 참고 자료

1. Building agents with the Claude Agent SDK - Anthropic, 1월 10, 2026에 액세스, <https://www.anthropic.com/engineering/building-agents-with-the-claude-agent-sdk>
2. Claude Code: Behind-the-scenes of the master agent loop - PromptLayer Blog, 1월 10, 2026에 액세스, <https://blog.promptlayer.com/claude-code-behind-the-scenes-of-the-master-agent-loop/>
3. Claude Code settings - Claude Code Docs, 1월 10, 2026에 액세스, <https://code.claude.com/docs/en/settings>
4. CLI reference - Claude Code Docs, 1월 10, 2026에 액세스, <https://code.claude.com/docs/en/cli-reference>
5. Claude Code CLI Cheatsheet: config, commands, prompts, + best practices - Shipyard.build, 1월 10, 2026에 액세스, <https://shipyard.build/blog/claude-code-cheat-sheet/>
6. Create custom subagents - Claude Code Docs, 1월 10, 2026에 액세스, <https://code.claude.com/docs/en/sub-agents>
7. VoltAgent/awesome-claude-code-subagents: A collection ... - GitHub, 1월 10, 2026에 액세스, <https://github.com/VoltAgent/awesome-claude-code-subagents>
8. How to create and use Subagents in Claude Code— a practical guide - CometAPI, 1월 10, 2026에 액세스, <https://www.cometapi.com/how-to-create-and-use-subagents-in-claude-code/>
9. Building with Claude Code Subagents (My Beloved Minions) | by Yee Fei - Medium, 1월 10, 2026에 액세스, <https://medium.com/@ooi_yee_fei/building-with-claude-code-subagents-my-beloved-minions-b5a9a4318ba5>
10. Best practices for Claude Code subagents - PubNub, 1월 10, 2026에 액세스, <https://www.pubnub.com/blog/best-practices-for-claude-code-sub-agents/>
11. Equipping agents for the real world with Agent Skills - Anthropic, 1월 10, 2026에 액세스, <https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills>
12. Agent Skills - Claude Code Docs, 1월 10, 2026에 액세스, <https://code.claude.com/docs/en/skills>
13. How to Build Claude Skills: Lesson Plan Generator Tutorial - Codecademy, 1월 10, 2026에 액세스, <https://www.codecademy.com/article/how-to-build-claude-skills>
14. CLAUDE.md and Skills Experiment: What's the Best Way to Organize Instructions for Claude? : r/ClaudeAI - Reddit, 1월 10, 2026에 액세스, <https://www.reddit.com/r/ClaudeAI/comments/1pe37e3/claudemd_and_skills_experiment_whats_the_best_way/>
15. Connect Claude Code to tools via MCP, 1월 10, 2026에 액세스, <https://code.claude.com/docs/en/mcp>
16. Using Claude Code with Gram-hosted MCP servers - Speakeasy, 1월 10, 2026에 액세스, <https://www.speakeasy.com/docs/gram/clients/using-claude-code-with-gram-mcp-servers>
17. google-calendars-mcp-server-by-cdata - GitHub, 1월 10, 2026에 액세스, <https://github.com/CDataSoftware/google-calendars-mcp-server-by-cdata>
18. epaproditus/google-workspace-mcp-server - GitHub, 1월 10, 2026에 액세스, <https://github.com/epaproditus/google-workspace-mcp-server>
19. MCP integration for Google Calendar to manage events. - GitHub, 1월 10, 2026에 액세스, <https://github.com/nspady/google-calendar-mcp>
20. liviaerxin/fastapi-tusd: Tus python server implementation on FastAPI - GitHub, 1월 10, 2026에 액세스, <https://github.com/liviaerxin/fastapi-tusd>
21. tuspyserver - PyPI, 1월 10, 2026에 액세스, <https://pypi.org/project/tuspyserver/2.2.2/>
22. Background Tasks - FastAPI, 1월 10, 2026에 액세스, <https://fastapi.tiangolo.com/tutorial/background-tasks/>
23. fastapi-tusd - PyPI, 1월 10, 2026에 액세스, <https://pypi.org/project/fastapi-tusd/>
24. flutter\_chunked\_upload 1.0.1 example | Flutter package - Pub.dev, 1월 10, 2026에 액세스, <https://pub.dev/packages/flutter_chunked_upload/versions/1.0.1/example>
25. LangGraph Beginner to Advance: Part 3: Multi-Entries Inputs for State in LangGraph, 1월 10, 2026에 액세스, <https://pub.towardsai.net/langgraph-beginner-to-advance-part-3-multi-entries-inputs-for-state-in-langgraph-fcd5f0ca0dca>
26. Use the graph API - Docs by LangChain, 1월 10, 2026에 액세스, <https://docs.langchain.com/oss/python/langgraph/use-graph-api>
27. Top Speech Recognition Engines You Can Use in 2025, 1월 10, 2026에 액세스, <https://blog.openreplay.com/top-speech-recognition-engines-2025/>
28. Best Speech-to-Text APIs in 2025 - Deepgram, 1월 10, 2026에 액세스, <https://deepgram.com/learn/best-speech-to-text-apis>
29. Top 8 speaker diarization libraries and APIs in 2025 - AssemblyAI, 1월 10, 2026에 액세스, <https://www.assemblyai.com/blog/top-speaker-diarization-libraries-and-apis>
30. 50 Prompts to Improve the Summary of Your One-on-One Meetings - Fireflies.ai, 1월 10, 2026에 액세스, <https://fireflies.ai/blog/prompts-to-improve-the-summary-of-one-on-one-meetings/>
31. Automating Action Item Extraction from Meeting Transcripts - Instructor, 1월 10, 2026에 액세스, <https://python.useinstructor.com/examples/action_items/>
32. Human-in-the-loop - Docs by LangChain, 1월 10, 2026에 액세스, <https://docs.langchain.com/oss/python/langchain/human-in-the-loop>
33. Interrupts - Docs by LangChain, 1월 10, 2026에 액세스, <https://docs.langchain.com/oss/javascript/langgraph/interrupts>
34. Connecting Jira and GitHub with MCP (Model Context Protocol) | by Shubhambayas, 1월 10, 2026에 액세스, <https://medium.com/@shubhambayas7/connecting-jira-and-github-with-mcp-model-context-protocol-4b164df8675c>
35. How to update graph state while preserving interrupts? - LangGraph - LangChain Forum, 1월 10, 2026에 액세스, <https://forum.langchain.com/t/how-to-update-graph-state-while-preserving-interrupts/1655>