

DMAP 빌더: 코드 없이 멀티에이전트 플러그인

만들기

선언형 멀티에이전트 플러그인의 세계로 초대합니다

멀티에이전트 개발, 지금 뭐가 불편한가요?

오케스트라를 상상해보세요. 지휘자가 악보 없이 수십 명의 연주자에게 "알아서 잘 맞춰봐"라고 한다면 어떨까요? 아마 끔찍한 소음이 나겠죠. AI 에이전트도 마찬가지입니다.

에이전트 1 개일 때는 괜찮습니다. 하지만 여러 에이전트가 협업해야 하는 순간, 복잡도가 폭발적으로 늘어나요. 기존 방식은 Python SDK로 직접 코딩하거나, LangChain, CrewAI, AutoGen 같은 프레임워크를 배워야 합니다. 프레임워크마다 문법이 다르고, 특정 런타임에 종속되어 버리죠.

진입장벽은 높고, 유지보수는 어렵고, 다른 환경으로 옮기려면 처음부터 다시 만들어야 합니다. 저는 그래서 근본적인 질문부터 해봤습니다.

"에이전트를 정의하는 데 꼭 코드가 필요할까?"

DMAP 빌더란?

레고를 떠올려보세요. 설명서만 보고 블록을 조립하면 멋진 작품이 완성되잖아요?

DMAP 빌더도 같은 원리입니다. **코드 한 줄 없이, Markdown 과 YAML 파일만 작성하면 멀티에이전트 플러그인이 똑딱 만들어집니다.**

DMAP 는 **Declarative Multi-Agent Plugin** 의 약자입니다. "선언형"이라는 말이 어렵게 느껴질 수 있는데, 쉽게 말하면 "어떻게(HOW)" 대신 "무엇을(WHAT)"만 적는 방식이에요.

택시 기사에게 경로를 일일이 지시하는 대신, "강남역이요"라고 목적지만 말하는 것처럼요.

DMAP 의 핵심 가치

- 선언형 명세 — 코드 대신 문서로 에이전트를 정의합니다
- 런타임 종립 — Claude Code, Codex CLI 등 어디서든 동작해요
- 관심사 분리 — 역할별로 깔끔하게 나뉘어 유지보수가 쉽습니다
- 비개발자 접근성 — 마크다운만 쓸 줄 알면 누구나 만들 수 있어요
- 도메인 범용 — 어떤 업무 영역이든 적용 가능합니다

구분	기존 프레임워크	DMAP
명세 방식	Python/TypeScript 코드	Markdown + YAML
런타임	프레임워크 전용 엔진	Claude Code, Codex CLI 등
진입장벽	코딩 필수	마크다운 작성 가능하면 OK
도구/모델 교체	코드 수정 필요	YAML 한 줄 변경

현재 버전 안내 (2026.2 월): v1.0.0

현재 DMAP 빌더는 **Claude Code 전용**으로 제공됨.

DMAP 표준 자체는 런타임 종립적으로 설계되었으며, 향후 Codex CLI, Gemini CLI 등
멀티 런타임 지원으로 확장 예정임.

핵심 개념: 회사에 비유하면

DMAP의 구조를 회사 조직에 비유하면 이해하기 쉬워요. 회사에는 부서장, 전문가, 통역사가
있잖아요? DMAP도 비슷한 구조입니다.

Skills = 부서장

스킬은 일을 배분하는 부서장 같은 역할이에요. 사용자의 요청을 받아서 어떤 에이전트에게
어떤 일을 시킬지 결정합니다. SKILL.md 파일 하나로 워크플로우 전체를 선언하죠.

Agents = 전문가

에이전트는 실제 일을 하는 전문가입니다. 각 에이전트는 AGENT.md(역할 정의),
agentcard.yaml(메타데이터), tools.yaml(사용 가능한 도구) 3개 파일로 구성돼요.

Gateway = 통역사

게이트웨이는 추상적인 선언을 구체적인 실행 환경으로 번역하는 통역사입니다. 에이전트가
"파일 검색 도구"라고 선언하면, 게이트웨이가 실제 런타임에서 어떤 도구를 쓸지 매핑해줘요.

리소스 마켓플레이스 = 사내 공유 드라이브

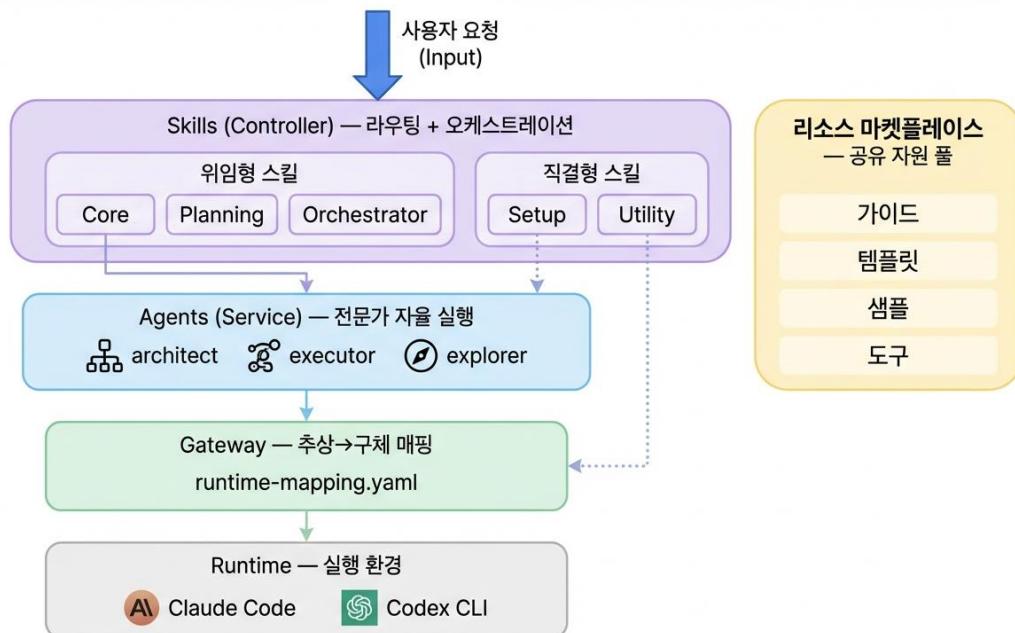
가이드, 템플릿, 샘플, 도구를 모아둔 공유 저장소입니다. 여러 플러그인이 함께 사용할 수 있는 공용 자원이에요.

이 컴포넌트들은 **Clean Architecture 원칙**을 따릅니다.

Skills → Agents → Gateway 순서로 단방향 의존하기 때문에, 각 부분을 독립적으로 수정하거나 교체할 수 있어요.

TIP: 선언형이란? "어떻게(HOW)" 대신 "무엇을(WHAT)"만 적으면 됩니다!

SQL처럼 원하는 결과만 말하면, 실행은 엔진이 알아서 해줘요.



DMAP 아키텍처 — Clean Architecture 기반 컴포넌트 구성

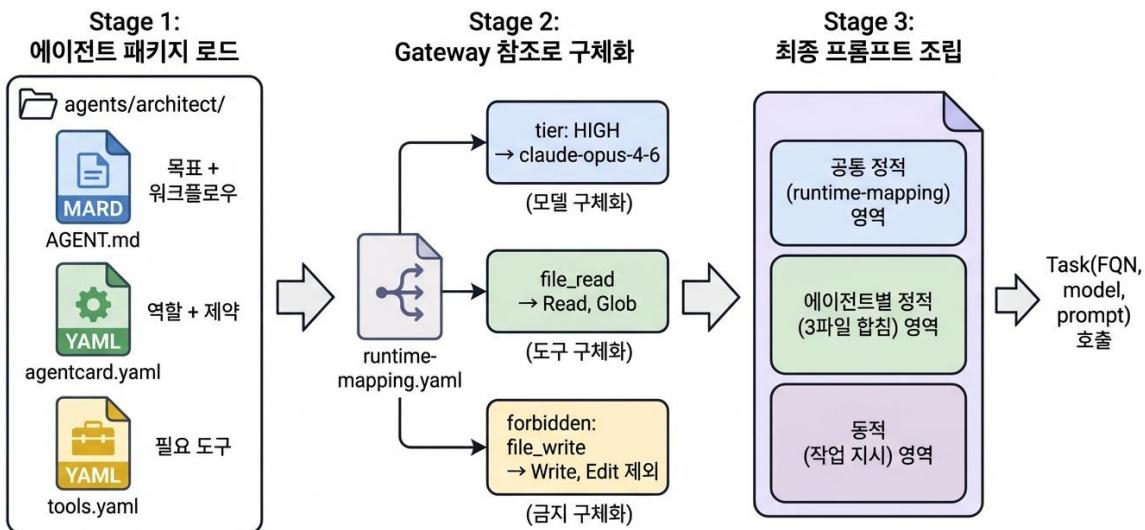
동작 원리: 어떻게 돌아갈까요?

프롬프트 조립 — 여행 가이드북 만들기

해외여행을 갈 때 가이드북을 만든다고 생각해보세요. 기본 정보(공통 규칙)에 목적지별

정보(에이전트 역할)를 합치고, 마지막에 오늘의 일정(작업 지시)을 추가하면 완성되죠?

DMAP 도 같은 방식으로 프롬프트를 조립합니다. 먼저 Gateway 의 runtime-mapping 으로 공통 설정을 로드하고, 에이전트 패키지 3 개 파일(AGENT.md, agentcard.yaml, tools.yaml)을 합친 뒤, 마지막으로 스킬이 전달하는 동적 작업 지시를 붙여서 최종 프롬프트를 완성합니다.



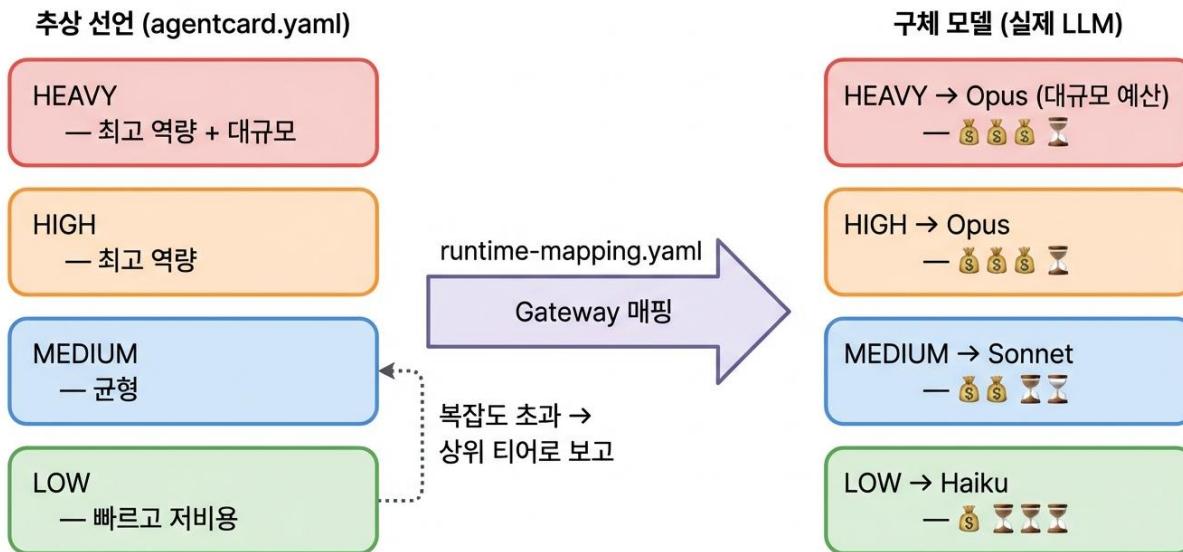
프롬프트 조립 과정 — 3 단계 레이어 구성

4-Tier 맵핑 — 택시 vs 버스 vs KTX

모든 일에 택시를 탈 필요는 없잖아요? 가까운 곳은 버스로, 먼 곳은 KTX 로 가면 비용도 절약되고 효율적이죠. DMAP 의 4-Tier 맵핑도 같은 원리입니다.

작업의 복잡도에 따라 HEAVY, HIGH, MEDIUM, LOW 4 단계로 나누고, 각 단계에 적합한 AI 모델을 배정해요. 간단한 파일 검색은 가벼운 모델(Haiku)이, 복잡한 아키텍처 설계는 강력한 모델(Opus)이 담당합니다. 작업 중 난이도가 올라가면 자동으로 상위 모델로 업그레이드되기도 해요.

DMAP 4-Tier 에이전트 모델 티어 매팅



4-Tier 모델 매핑 — 복잡도별 최적 모델 배정

스킬 활성화 — 114 안내교환대

예전 114 안내교환대를 기억하시나요? "피자집 전화번호 알려주세요"라고 하면 알아서

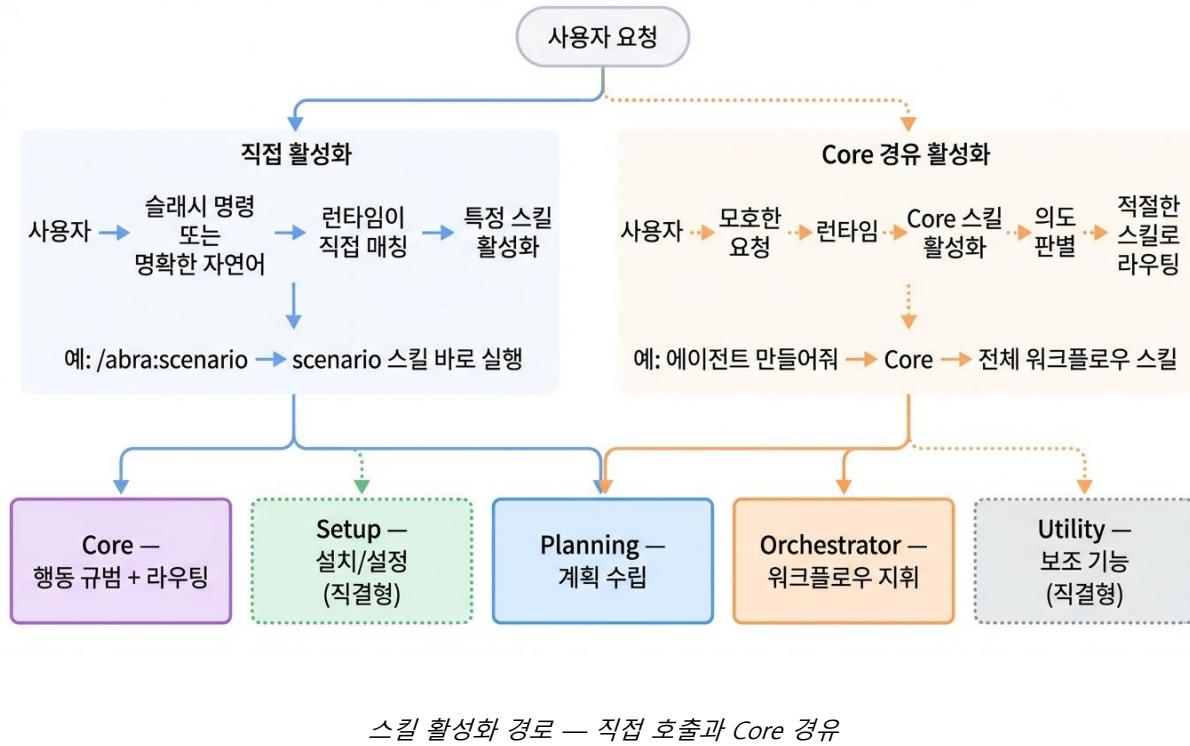
연결해주었죠. DMAP 의 스킬 활성화도 비슷합니다.

두 가지 경로가 있어요.

첫째, 슬래시 명령어로 직접 호출하는 방법 — 정확히 어떤 스킴이 필요하지 알 때 사용합니다.

둘째, Core 스킬을 경운하는 방법 — 모호한 요청이 들어오면 Core 가 의도를 분석해서 적절한

스킬로 라우팅해줘요. 런타임은 skills/ 디렉토리를 자동 스캔해서 사용 가능한 스킬을 발견합니다.



플러그인 개발 방법: 요리 레시피 따라하기

요리를 할 때 레시피를 따라하면 누구나 맛있는 음식을 만들 수 있잖아요? DMAP 플러그인 개발도 마찬가지입니다. 4 단계 레시피만 따라가면 플러그인이 완성돼요.

Phase	단계	비유
1	요구사항 수집	어떤 요리를 만들지 정하기

2	설계 및 계획	레시피 작성하기
3	플러그인 개발	실제 요리하기
4	검증 및 완료	시식하고 플레이팅

각 Phase 가 끝날 때마다 사용자의 확인을 받습니다. "이 요리 맞죠?"라고 중간중간 물어보는 셈이에요. 덕분에 엉뚱한 결과물이 나오는 걸 미리 방지할 수 있습니다.

실전 예제: abra 플러그인

백문이 불여일견! 실제로 DMAP 으로 만든 플러그인을 살펴볼게요.

"abra"는 Dify 워크플로우 DSL 자동화 플러그인입니다. Dify 라는 AI 워크플로우 플랫폼에서 사용하는 DSL(Domain Specific Language)을 자동으로 생성하고, 프로토타이핑까지 해주는 똑똑한 플러그인이에요.

Abra 는 아브라카다브라(Abracadabra: 원하는 것이 모두 이루어지는 마법의 주문)에서 착안한 이름

에이전트 5 명의 전문가 팀

- scenario-analyst — 요구사항을 분석하고 시나리오를 만드는 기획자
- dsl-architect — Dify DSL 코드를 설계하는 아키텍트
- plan-writer — 개발계획서를 작성하는 PM
- prototype-runner — Dify 에서 프로토타이핑을 실행하는 테스터

- agent-developer — 최종 Agent 코드를 개발하는 개발자

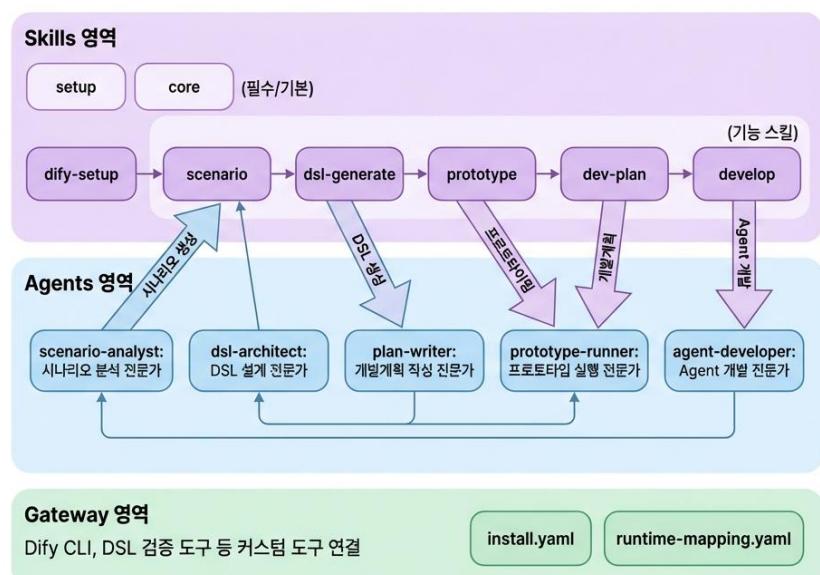
스킬 8 개 — 부서장들의 업무 지시서

setup, core, dify-setup, scenario, dsl-generate, prototype, dev-plan, develop — 각 스킬이 워크플로우의 한 단계를 담당합니다.

사용자 입장에서는 시나리오 생성 → DSL 생성 → 프로토타이핑 → 개발계획 → Agent 개발 순서로 자연스럽게 진행돼요. 복잡한 내부 구조를 몰라도 됩니다!

abra 플러그인 — Dify DSL 자동화

```
abra/
├── .claude-plugin/
│   └── plugin.json
│       └── marketplace.json
├── skills/ (8개 스킬)
├── agents/ (5개 에이전트)
│   ├── scenario-analyst/
│   ├── dsl-architect/
│   ├── plan-writer/
│   ├── prototype-runner/
│   └── agent-developer/
└── gateway/
    ├── install.yaml
    └── runtime-mapping.yaml
└── commands/ (슬래시 명령)
    └── README.md
```



abra 플러그인 구조 — 5 개 에이전트와 8 개 스킬의 협업

앞으로의 로드맵

DMAP는 계속 진화하고 있습니다. 앞으로 어떤 모습이 될지 살짝 엿볼게요.

현재

- Claude Code 런타임 기반으로 안정적으로 동작 중

단기 계획

- 다중 런타임 지원 — Codex CLI, Gemini CLI 등 다양한 환경에서 실행

중기 계획

- 플러그인 마켓플레이스 — 만든 플러그인을 공유하고 다운로드하는 생태계 구축
- 커뮤니티 생태계 — 개발자들이 함께 성장하는 오픈소스 커뮤니티

장기 비전

- 비개발자용 시각적 플러그인 빌더 — 드래그 앤 드롭으로 플러그인을 만드는 노코드 UI

지금 바로 시작하기

어렵게 느껴지셨나요? 걱정 마세요. 4 단계면 시작할 수 있습니다!

- Step 1: DMAP 표준 문서를 읽고 전체 구조를 파악
- Step 2: 플로그인 요구사항 명세서 작성
- Step 3: /dmap:develop-plugin 명령으로 플러그인 개발 시작
- Step 4: 플러그인 설치 및 테스트(플러그인의 README.md 참조)

코드 한 줄 없이 멀티에이전트 플러그인을 만들 수 있다는 걸 직접 경험해보세요.

지금 바로 시작해보세요!

TIP: DMAP 표준 문서와 샘플은 GitHub에서 확인하세요:

- 표준문서: <https://github.com/cna-bootcamp/gen-dmap/blob/main/standards/plugin-standard.md>
- 요구사항 샘플: <https://github.com/cna-bootcamp/abra/blob/main/putput/requirement.md>