PLAYBOOK

MSAP.ai

기업이 AI 에이전트에 요구하는 여덟 가지 — Hermes Agent는 어디까지 기본으로 주나

챗봇과 에이전트를 가르는 선은 화면

챗봇과 에이전트를 가르는 선은 화면 모양이 아니라 실행 방식에 있습니다. 챗봇은 질문 하나를 받으면 답 하나를 내놓고 멈춥니다. 다음 행동을 정하는 사람은 언제나 사용자입니다. 반면 에이전트는 목표를 받으면 그 목표를 달성할 때까지 스스로 다음 행동을 고릅니다. 어떤 도구를 쓸지, 어떤 파일을 열지, 언제 멈추고 언제 계속할지를 사람이 매 단계 지시하지 않아도 에이전트가 판단합니다.

2026-08-22 · 작성자: SMART CF Team · msap.ai

목차

함께 읽을 자료

기업이 AI 에이전트에 요구하는 여덟 가지 — Hermes Agent 는 어디까지 기본으로 주나

1장: 기업이 AI 에이전트에 요구하는 여덟 가지

1.1 챗봇과 에이전트의 경계

1.1.1 응답형 도구와 자율 실행의 차이

챗봇과 에이전트를 가르는 선은 화면 모양이 아니라 실행 방식에 있습니다. 챗봇은 질문 하나를 받으면 답 하나를 내놓고 멈춥니다. 다음 행동을 정하는 사람은 언제나 사용자입니다. 반면 에이전트는 목표를 받으면 그 목표를 달성할 때까지 스스로 다음 행동을 고릅니다. 어떤 도구를 쓸지, 어떤 파일을 열지, 언제 멈추고 언제 계속할지를 사람이 매 단계 지시하지 않아도 에이전트가 판단합니다. 공식 문서에서도 실행형 에이전트를 목표 지향적으로 도구를 선택해 이어 실행하는 존재로 설명합니다[S1].

이 차이는 사소해 보이지만 결과는 사소하지 않습니다. 챗봇에게 "이 파일 요약해줘"라고 하면 사람이 결과를 읽고 다음 지시를 내리는 자리가 매번 끼어듭니다. 반면 "이번 주 보고서 초안을 만들어줘"라고 요청받은 에이전트는 파일을 찾고, 내용을 읽고, 문서를 쓰고, 필요하면 외부 도구를 호출하는 과정을 사람 개입 없이 이어갑니다. 이 이어붙임 자체가 자율 실행이고, 자율 실행이 파일 쓰기·외부 호출·메시지 발송처럼 되돌리기 어려운 동작을 만들어냅니다. 사람이 답을 확인하고 다음 질문을 던지는 순환이 사라지는 자리마다, 잘못된 판단이 다음 단계로 그대로 넘어갈 여지가 생깁니다.

그렇다고 모든 에이전트가 똑같이 위험한 것은 아닙니다. 읽기 전용으로 자료를 찾아 요약만 하는 에이전트와, 파일을 고쳐 쓰고 외부로 메시지를 내보내는 에이전트는 위험 수준이 다릅니다. "에이전트라고 이름 붙었으니 위험하다"는 식의 일반화는 질문 자체가 잘못됐습니다. 진짜 물어야 할 질문은 "이 에이전트가 어떤 되돌리기 어려운 동작을 사람 확인 없이 스스로 수행하는가"입니다. 조회만 하는 에이전트라면 자율성이 주는 위험은 크지 않지만, 쓰기와 발송까지 맡기는 순간부터는 이야기가 달라집니다.

그래서 자율 실행을 허용하기로 정하는 순간부터 "무엇을 확인하고 무엇을 막을지"는 도입하고 나서 운영 중에 손보는 문제가 아니라 도입 전에 먼저 설계해 둬야 할 항목으로 바뀝니다. 챗봇을 쓸 때는 사람이 매번 브레이크를 밟으므로 이 설계가 크게 필요 없었습니다. 자율 실행 앞에서는 브레이크를 시스템 쪽에 미리 달아 둬야 합니다. 이 장 나머지 절이 다루는 여덟 가지 요구는 결국 이 브레이크를 어디에, 어떻게 다는가에 대한 답입니다.

1.1.2 반복 업무를 굴리는 네 부품

같은 업무를 에이전트에게 계속 맡기려면 네 가지 부품이 갖춰져야 합니다. 절차 기억은 같은 일을 다시 처음부터 지시하지 않아도 되도록 수행 절차를 저장해 두는 능력이고, 채널은 사람이 화면 앞에 없을 때도 요청이 들어올 수 있는 입구입니다. 실제로 필요할 때 불러 쓰는 절차 문서를 단계적으로 공개하는 방식으로 저장해 두는 사례가 공식 문서에도 나옵니다[S4]. 무인 실행은 사람이 각 단계마다 개입하지 않아도 목표를 끝까지 이어 처리하는 능력이며, 검증은 실행이 끝난 뒤 결과가 실제로 맞는지 확인하는 절차입니다. 이 네 가지는 순서대로 쌓이는 층이 아니라 한 업무를 굴리는 데 동시에 필요한 부품에 가깝습니다.

왜 네 개 모두 있어야 하는지는 하나씩 빼 보면 드러납니다. 절차 기억이 없으면 매번 처음부터 맥락을 다시 설명해야 하니 반복 업무를 맡기는 의미가 줄어듭니다. 채널이 없으면 사람이 화면 앞에 앉아 있는 시간에만 요청을 받을 수 있어 "반복"이라는 말이 무색해집니다. 무인 실행이 없으면 결국 사람이 각 단계를 확인해야 하니 자율 실행의 이점이 사라지고, 검증이 없으면 무인 실행이 만든 결과를 아무도 확인하지 않는 채로 다음 업무에 흘러들어갑니다. 공식 문서는 이런 반복 작업 처리와 절차 저장, 메신저 등 다양한 입구를 통한 요청 수신을 기능 개요에서 함께 설명합니다[S6].

물론 모든 반복 업무에 네 부품이 똑같은 비중으로 필요하지는 않습니다. 하루 한 번 사람이 확인 버튼만 눌러 주면 되는 업무라면 무인 실행의 비중은 낮고 검증의 비중이 큽니다. 반대로 야간에 들어오는 요청을 받아 처리해야 하는 업무라면 채널과 무인 실행이 핵심이고 절차 기억은 상대적으로 가볍게 갖춰도 됩니다. 그러니 이 네 부품을 "전부 최고 수준으로 갖춰야 한다"는 기준으로 읽을 필요는 없습니다. 자기 업무에 맞게 각 부품이 얼마나 필요한지 저울질하는 틀로 쓰면 됩니다.

이 네 부품은 이 문서 3장과 4장이 각각 절차 기억, 채널과 무인 실행을 다룰 때 그대로 다시 등장하는 골격입니다. 지금 자기 조직의 반복 업무 하나를 떠올려 네 칸을 채워 보면, 뒤에 이어지는 장을 읽을 때 "이 부분이 내 업무의 어느 칸에 해당하는가"를 바로 대입할 수 있습니다.

부품	한 줄 정의
절차 기억	같은 일을 다시 지시하지 않아도 되도록 수행 절차를 저장해 두는 능력
채널	사람이 화면 앞에 없을 때도 요청이 들어올 수 있는 입구
무인 실행	사람이 각 단계마다 개입하지 않아도 목표까지 실행을 이어가는 능력
검증	실행이 끝난 뒤 결과가 실제로 맞는지 확인하는 절차

1.2 위험이 기능 요구로 바뀌는 지점

1.2.1 OWASP가 분류한 에이전트형 애플리케이션의 위험

앞 절에서 "자율 실행이 위험을 만든다"고 짚었는데, 이 위험을 막연한 우려가 아니라 구체적인 범주로 정리한 표준 기구 자료가 있습니다. OWASP Gen AI Security Project가 2025년 12월에 공개한 「Top 10 for Agentic Applications 2026」은 에이전트형 애플리케이션에서 반복적으로 나타나는 위험을 열 개 범주로 정리합니다[S10]. 이 열 개 범주는 특정 제품을 평가하려고 만든 목록이 아니라, 목표를 받아 스스로 행동을 이어가는 소프트웨어 전반에서 관찰된 위험을 정리한 결과입니다.

열 개 범주는 다음과 같습니다[S10].

- 목표 탈취: 에이전트에게 주어진 목표 자체가 조작되어 원래 의도와 다른 방향으로 실행되는 위험
- 도구 오남용: 에이전트가 쓸 수 있는 도구가 의도한 범위를 벗어나 사용되는 위험
- 신원과 권한 남용: 에이전트가 자신의 권한 또는 다른 사용자의 권한을 부당하게 행사하는 위험
- 공급망: 에이전트가 불러오는 외부 코드·모델·데이터 경로에 신뢰할 수 없는 요소가 섞여 드는 위험
- 임의 코드 실행: 에이전트가 검증되지 않은 코드를 그대로 실행하는 위험

- 메모리 오염: 에이전트가 저장해 둔 절차나 기억에 잘못된 내용이 섞여 이후 실행에 영향을 주는 위험
- 에이전트 간 통신: 여러 에이전트나 채널이 주고받는 메시지가 검증 없이 신뢰되는 위험
- 연쇄 장애: 한 단계의 오류가 뒤따르는 단계로 그대로 전파되는 위험
- 신뢰 착취: 사람이 에이전트의 판단을 그대로 믿고 확인 절차를 건너뛰게 되는 위험
- 불량 에이전트: 원래 설계된 범위를 벗어나 행동하는 에이전트가 발생하는 위험

이 목록을 처음 보면 범주 이름부터 외워야 할 것 같지만, 실제로 필요한 건 암기가 아닙니다. 열 개 범주를 하나하나 따지는 대신 "누가 실행을 승인하는가"와 "결과를 누가 확인하는가"라는 두 질문으로 요약해서 읽으면 충분합니다. 이 두 질문에 답할 장치가 있는지가, 뒤에서 이 범주들을 기능 요구로 좁힐 때 기준이 됩니다.

1.2.2 열 개 위험에서 여덟 개 기능으로

OWASP의 열 개 범주를 그대로 제품 평가표로 쓰기는 어렵습니다. 범주가 너무 세분화되어 있어서 "이 기능이 있으면 이 위험이 없어진다"는 식의 일대일 대응이 되지 않기 때문입니다. 그래서 이 문서는 열 개 범주를 실행 단계로 다시 묶는 방식을 택합니다. 누가 실행을 시작하는가, 무엇을 만지는가, 어디로 나가는가, 누가 확인하는가라는 네 단계로 범주를 묶으면 절차 기억·채널 통합·무인 실행·실행 격리·데이터 처리 위치 공개·권한 분리·위험 명령 승인·실행 결과 검증이라는 여덟 항목으로 좁혀집니다.

이 여덟 항목은 특정 제품의 기능 목록을 그대로 옮겨 온 것이 아닙니다. 위 열 개 범주에서 역으로 추출한 요구이고, 그 추출 과정은 아래 표로 확인할 수 있습니다. "시작 지점"에 걸리는 목표 탈취·신뢰 착취·불량 에이전트는 누가, 어떤 조건에서 실행을 시작할 수 있는가의 문제이므로 무인 실행과 위험 명령 승인으로 이어집니다. 실행을 시작할 권한과 확인 절차를 나누어 두는 방식은 이미 여러 공개 자료에서 별개의 확인 단계로 다뤄집니다[S3]. "실행 범위"에 걸리는 도구 오남용·신원과 권한 남용·임의 코드 실행은 에이전트가 무엇을 만질 수 있는가의 문제이므로 실행 격리와 권한 분리로 이어집니다. "외부 노출"에 걸리는 공급망·에이전트 간 통신은 데이터와 메시지가 어디로 나가는가의 문제이므로 데이터 처리 위치 공개와 채널 통합으로 이어집니다. 실제로 에이전트가 여러 외부 연동 지점을 통해 검색·추출 기능을 확장하는 사례는 공개된 연동 문서에서도 확인됩니다[S7]. "사후 확인"에 걸리는 메모리 오염·연쇄 장애는 저장된 절차와 실행 결과를 누가 확인하는가의 문제이므로 절차 기억과 실행 결과 검증으로 이어집니다.

이 대응이 완벽하게 깔끔하지는 않습니다. 예를 들어 도구 오남용은 실행 격리뿐 아니라 위험 명령 승인과도 맞닿아 있고, 연쇄 장애는 검증뿐 아니라 무인 실행의 정지 조건과도 관련이 있습니다. 위험 범주와 기능 요구가 정확히 하나씩 짝지어지는 관계가 아니라, 여러 범주가 겹쳐서 하나의 요구를 만드는 관계라는 점은 미리 밝혀 둡니다. 그럼에도 네 단계로 묶어 보면 여덟 항목 각각이 어느 위험을 막기 위한 자리인지는 분명해집니다.

이 여덟 항목이 이 장에서 정한 뒤로는 문서 전체에서 재정의 없이 그대로 쓰이는 잣대입니다. 2장은 이 여덟 항목 중 제품이 기본으로 제공하는 범위를 대응표로 보여주고, 3장부터 7장까지는 항목을 하나씩 다루며, 8장은 남는 몫을 조직이 채우는 판단으로 정리합니다. "이 항목이 왜 여덟 개냐"는 질문을 받으면, 열 개 위험 범주를 실행 단계 네 갈래로 묶었더니 여덟 개로 좁혀졌다는 한 문장으로 답할 수 있으면 충분합니다.

실행 단계	해당 OWASP 범주	좁혀지는 기능 요구
누가 시작하는가	목표 탈취 · 신뢰 착취 · 불량 에이전트	③ 무인 실행 · ⑦ 위험 명령 승인
무엇을 만지는가	도구 오남용 · 신원과 권한 남용 · 임의 코드 실행	④ 실행 격리 · ⑥ 권한 분리

실행 단계	해당 OWASP 범주	좁혀지는 기능 요구
어디로 나가는가	공급망 · 에이전트 간 통신	⑤ 데이터 처리 위치 공개 · ② 채널 통합
누가 확인하는가	메모리 오염 · 연쇄 장애	① 절차 기억 · ⑧ 실행 결과 검증

1.3 장 구성과 확인 표시 규칙

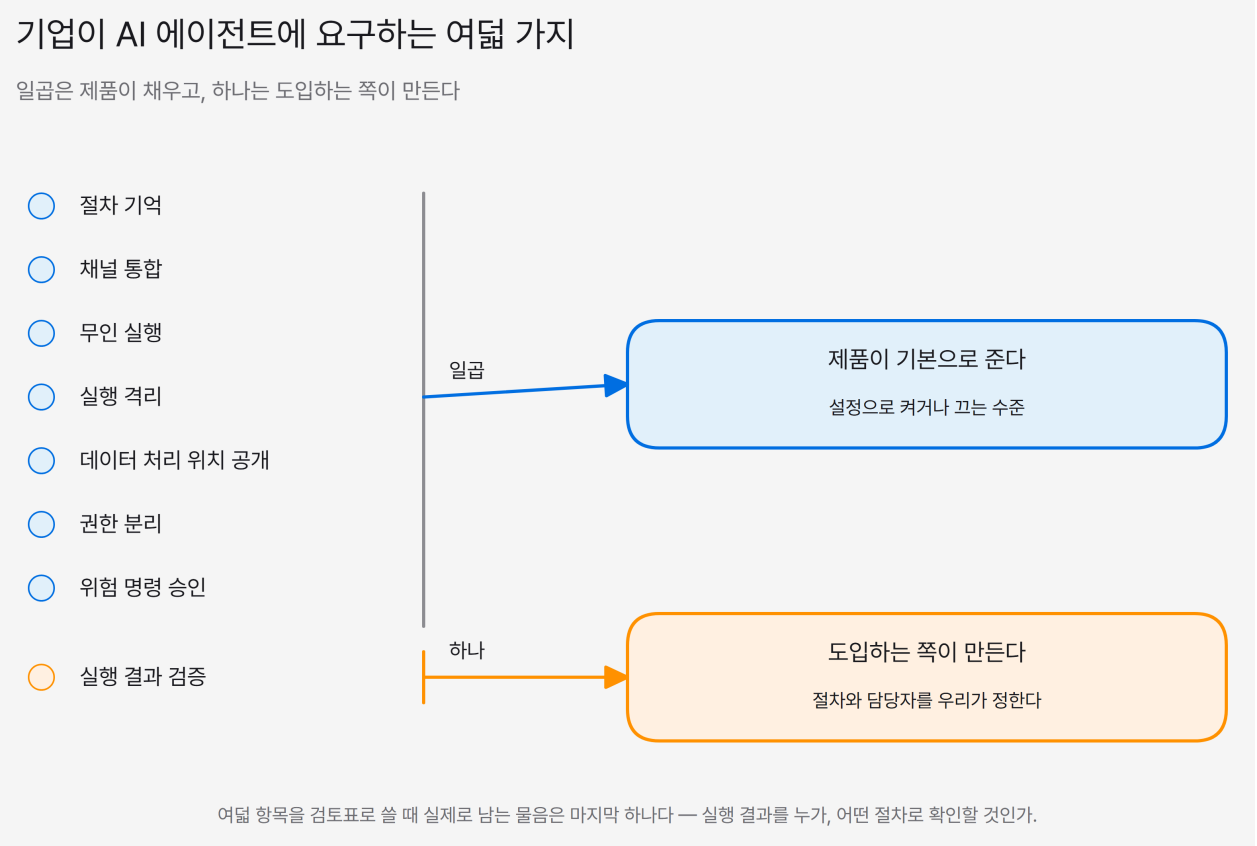
1.3.1 장별 대응표

여덟 항목이 정해졌으니, 이 문서 뒤 여섯 개 장이 이 항목을 어떻게 나눠 다루는지 미리 지도로 그려 두면 필요한 장만 골라 읽으려는 독자가 자기 위치를 잃지 않습니다. 2장은 여덟 항목을 한 번에 훑는 대응표로 시작해 전체 그림을 먼저 보여줍니다. 3장부터 7장까지는 항목을 하나씩 또는 짝지어 깊이 다루고, 8장은 공식 자료만으로는 확인되지 않는 부분을 조직이 스스로 판단해야 할 몫으로 정리합니다.

각 장이 어떤 항목을 다루는지, 그리고 그 장을 읽으며 스스로에게 던질 핵심 질문이 무엇인지를 표로 정리하면 다음과 같습니다. 이 표는 목차 역할을 겸하므로, 지금 당장 특정 항목만 궁금한 독자는 해당 장으로 바로 건너뛰어도 무방합니다.

장 번호	다루는 항목	핵심 질문
2장	여덟 항목 전체 대응표	이 제품은 여덟 항목 중 무엇을 기본으로 제공하는가
3장	① 절차 기억	같은 일을 다시 지시하지 않아도 되는가
4장	② 채널 통합 · ③ 무인 실행	사람이 없을 때도 요청을 받아 끝까지 처리하는가
5장	④ 실행 격리	잘못된 실행이 다른 영역으로 번지지 않는가
6장	⑤ 데이터 처리 위치 공개	어떤 데이터가 어디로 나가는지 확인할 수 있는가
7장	⑥ 권한 분리 · ⑦ 위험 명령 승인 · ⑧ 실행 결과 검증	누가 실행을 승인하고 누가 결과를 확인하는가
8장	남은 몫	공식 자료가 채우지 못하는 부분을 조직이 어떻게 판단할 것인가

이 지도에서 눈여겨볼 점은 8장이 "부족한 점을 지적하는 장"이 아니라는 것입니다. 여덟 항목 모두를 어느 한 제품이 완벽하게 채우기는 어렵습니다. 8장은 채워지지 않는 몫을 숨기지 않고 짚어서, 도입을 검토하는 조직이 그 몫을 어떻게 메울지 스스로 판단하도록 돕는 자리입니다.



1.3.2 확인 표시 규칙

이 문서는 공식 문서·공개 이슈·표준 기구 자료로 확인된 사실과, 공식 자료가 침묵하는 지점을 구분해서 씁니다. 확인된 사실은 위 절에서처럼 근거를 밝히며 서술하고, 공식 자료가 답하지 않는 지점은 추측으로 메우지 않고 "공식 문서에 명시가 없다"처럼 사실 그대로 적습니다. 예를 들어 업데이트 시 로컬에서 손댄 설정이 그대로 보존되는지에 대해 공식 FAQ는 스킬을 여러 작업 공간에 자동으로 동기화한다고만 밝히고, 이 동기화가 로컬 수정분을 덮어쓰지 않는다는 점까지 명시적으로 보장하지는 않습니다[S5]. 이런 경우 이 문서는 "보존된다"고 단정하지 않고 부분 확인이라는 사실을 그대로 남깁니다.

이렇게 구분하는 이유는 독자가 자신이 검토 중인 제품을 이 문서와 대조할 때 헷갈리지 않게 하기 위해서입니다. 확인된 사실과 확인되지 않은 지점이 같은 문장 안에서 뒤섞이면, 독자는 어느 부분을 그대로 믿고 어느 부분을 직접 검증해야 하는지 구분할 수 없습니다. 계측이나 사용 기록을 외부로 내보내는 절차가 있는지도 비슷한 사례입니다. 공개된 이슈 기록은 토큰 사용량이나 세션 길이 같은 일부 계측이 이미 존재한다고 밝히면서도, 통합된 계측 체계는 아직 없다고 말합니다[S9]. 한편 개발 정책 문서는 새로운 계측 항목을 추가할 때 사용자가 먼저 동의하는 절차를 거치도록 정해 두고 있다고 밝힙니다[S8]. 이 문서는 이 지점에서도 "계측이 전혀 없다"거나 "계측이 외부로 나간다"고 단정하지 않고, 공개된 자료가 밝힌 범위까지만 서술합니다.

이 규칙에는 한계도 있습니다. 공식 자료가 침묵한다고 해서 그 기능이 없다고 단정할 수는 없습니다. 문서화되지 않았을 뿐 실제로는 존재하는 절차일 수도 있습니다. 그래서 이 문서는 "확인 실패"를 "기능 없음"으로 바꿔 읽지 말라고 미리 당부해 둡니다. 확인 실패는 이 문서가 그 지점을 검증하지 못했다는 뜻이지, 제품에 그 기능이 없다는 결론이 아닙니다.

이 확인 표시 규칙을 1장에서 먼저 약속해 두면, 2장부터는 매 문단마다 "이것이 확인된 사실인지 아닌지"를 다시 설명할 필요가 없습니다. 본문에 표기된 출처 번호가 있으면 확인된 사실이고, "공식 문서에 명시가 없다"는

문장이 나오면 그 지점은 침묵으로 남아 있다는 뜻으로 읽으면 됩니다. 이 약속이 이 문서 전체가 과장 없이 쓰였다는 신뢰를 뒷받침하는 기반이 됩니다.

2장: Hermes Agent 는 무엇인가

2.1 제품 정의와 실행 방식

2.1.1 오픈소스 에이전트 런타임으로서의 위치

Hermes Agent 는 Nous Research 가 공개한 오픈소스 에이전트 런타임(agent runtime, 에이전트를 구동하는 실행 환경)입니다[S1]. 공식 문서 사이트와 GitHub 저장소에 코드 전체와 사용 방법이 그대로 공개되어 있고, 누구나 저장소를 내려받아 실행 흐름을 직접 읽을 수 있습니다[S2]. 이 사실이 왜 첫 번째로 다룰 만한지 설명하면, 도입을 검토하는 조직이 가장 먼저 묻는 질문이 "이 제품을 우리가 직접 들여다볼 수 있는가"이기 때문입니다. 문서화된 기능 설명과 실제 코드가 어긋나는지, 특정 동작이 왜 그렇게 구현되었는지를 벤더의 답변에만 의존하지 않고 조직 내부에서 확인할 길이 열려 있다는 뜻입니다.

다만 오픈소스라는 사실 자체가 안전성이나 완성도를 보장하지는 않습니다. 코드가 공개되어 있어도 그것을 읽고 판단할 역량이 조직 안에 없다면 공개는 실질적인 이점으로 이어지지 않습니다. 또한 공개 저장소라는 것은 변경 이력과 이슈 트래커도 함께 공개된다는 뜻이어서, 아직 해결되지 않은 논의나 미완성 기능도 같이 드러납니다. 이 점은 뒤에서 다룰 항목별 확인 과정에서도 계속 등장합니다 — 공식 문서에 쓰여 있다는 것과 실제로 검증되었다는 것은 다른 층위의 확인이기 때문입니다.

그래서 이 절에서 확정하고 넘어갈 전제는 두 가지입니다. 첫째, Hermes Agent 는 폐쇄형 상용 제품이 아니라 코드와 문서가 모두 열람 가능한 오픈소스 런타임입니다[S1][S2]. 둘째, 이후 장에서 다루는 모든 기능 확인은 이 공개된 문서와 저장소를 근거로 삼습니다. 도입을 검토하는 IT 담당자라면 이 두 전제를 먼저 자기 조직의 내부 검토 프로세스에 대입해 보는 것이 좋습니다. 오픈소스 검토 역량이 없는 조직이라면 이 장 이후에 나오는 "공식 문서에 명시됨"이라는 표현을 벤더의 일방적 주장이 아니라 직접 재확인 가능한 근거로 받아들일 수 있는지 스스로 판단해야 합니다.

2.1.2 실행 규모의 유연성

공식 저장소는 이 런타임을 어디서 돌릴 수 있는지에 관해 뚜렷한 입장을 밝힙니다. "5달러짜리 VPS(가상 사설 서버, Virtual Private Server)에서도, GPU 클러스터에서도, 서버리스 인프라에서도 실행할 수 있다"고 설명합니다[S2]. 설치에 필요한 전제조건은 세 가지로 정리되어 있습니다.

- Python 3.11
- Node.js
- ripgrep(코드 저장소를 빠르게 검색하는 명령행 도구)

이 세 가지만 갖추면 실행 환경을 시작할 수 있다는 뜻이고, 별도의 전용 하드웨어나 특수 클러스터 구성을 전제로 하지 않는다는 뜻이기도 합니다. 소규모 시범 운영과 대규모 운영이 같은 코드베이스에서 출발할 수 있다는 점은 도입 초기의 인프라 결정 부담을 줄여 줍니다. 작은 VPS 한 대로 먼저 시작해 보고, 필요할 때 더 큰 인프라로 옮겨가는 방식이 문서상으로는 막혀 있지 않습니다.

다만 이 설명에는 한 가지 정직하게 짚어야 할 한계가 있습니다. 공식 문서에는 "5달러짜리 VPS부터 GPU 클러스터까지"라는 정성적인 범위만 나와 있을 뿐, 최소 CPU 코어 수나 메모리 용량, 디스크 용량 같은 정량적인 사양은 어디에도 나와 있지 않습니다[S2]. 구체적인 서버 스펙을 근거로 예산을 세우려는 담당자라면 이 문서만으

로는 답을 얻을 수 없고, 실제로 시범 설치를 해 보며 자기 조직의 작업량에 맞는 사양을 직접 가늠하는 과정이 필요합니다. 정성적 범위는 실행 자체가 특정 인프라에 종속되지 않는다는 설계 방향을 보여 주지만, 그 방향이 곧 정량적 보증은 아니라는 점을 구분해서 받아들여야 합니다.

2.2 핵심 구성요소

2.2.1 코어와 스킬

Hermes Agent 의 구조는 크게 코어(core)와 스킬(skill) 두 층으로 나뉩니다. 코어는 대화를 주고받고 실제 작업을 실행하는 본체이고, 스킬은 코어가 필요할 때 불러 쓰는 절차 기억입니다. 공식 문서는 스킬을 "에이전트가 필요할 때 불러오는 온디맨드 지식 문서"라고 설명하며, 한 번에 모든 정보를 열지 않고 필요한 순간에 필요한 만큼만 펼쳐 보는 점진적 공개 방식을 따르고, agentskills.io(에이전트 스킬을 위한 공개 규격 사이트)라는 공개 표준과 호환된다고 밝힙니다[S4]. 이 구조가 왜 중요한지 짚어 보면, 기능이 코어 안에 고정되어 있는지 아니면 스킬로 나중에 계속 늘어나는지에 따라 제품의 확장성 자체가 달라지기 때문입니다.

실제로 이 스킬 체계는 설치 시점에 고정된 목록이 아닙니다. 공식 문서는 "에이전트가 스스로 스킬을 자유롭게 작성한다 — 한 번의 작업 턴이 끝난 뒤 실행되는 백그라운드 자기개선 리뷰에서 만들어지는 스킬도 포함해서"라고 설명합니다[S4]. 즉 에이전트가 반복 작업을 처리하는 과정에서 스스로 절차를 문서화하고, 그것이 다음 실행에서 다시 쓰이는 절차 기억으로 쌓입니다. 이 메커니즘의 세부 흐름은 3장에서 다시 자세히 다룹니다. 여기서는 "설치하면 끝인가, 계속 채워 넣는 구조인가"라는 질문에 대한 답이 후자라는 점만 확정해 두면 됩니다.

이 구조에는 운영상 짚어야 할 지점도 있습니다. FAQ 문서는 "hermes update 명령은 최신 코드를 받아 오고 의존성을 한 번만(프로파일마다 반복하지 않고) 다시 설치한다. 그런 다음 갱신된 스킬을 모든 프로파일에 자동으로 동기화한다"고 설명합니다[S5]. 여러 프로파일(하나의 설치에서 분리 운용하는 여러 실행 단위)을 동시에 쓰는 조직이라면, 스킬이 중앙에서 갱신되고 각 프로파일로 퍼진다는 이 동작 방식을 미리 파악해 두는 편이 좋습니다. 결국 코어는 고정된 실행 본체이고 스킬은 계속 쌓이고 갱신되는 절차 기억이라는 이 두 층 구조가, 이 제품이 설치 이후에도 기능이 늘어날 수 있다고 말하는 근거입니다.

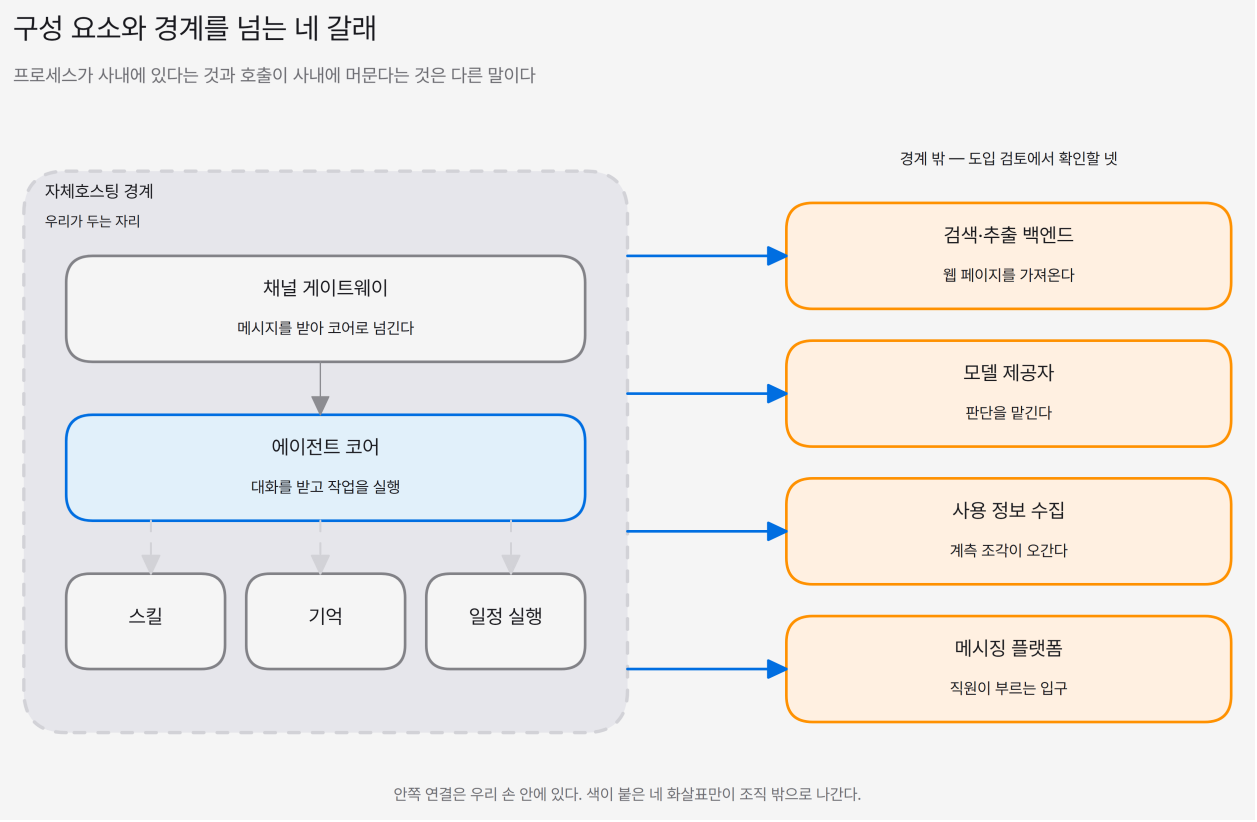
2.2.2 채널 · 도구 · 모델 연동 세 축

Hermes Agent 는 서로 다른 역할을 하는 세 축을 각각 교체 가능한 구성으로 둡니다. 요청이 들어오는 입구인 메시징 채널, 실제 작업을 실행하는 손에 해당하는 도구와 검색·추출 백엔드, 판단을 내리는 두뇌에 해당하는 모델 제공자입니다. 이 세 축이 분리되어 있다는 사실은 이 제품이 특정 메신저나 특정 모델에 고정되지 않는다는 뜻이고, 6장에서 다룰 데이터 처리 위치 논의의 전제가 되는 구조이기도 합니다.

세 축의 실제 예시는 다음과 같습니다.

- 채널: 메시징 플랫폼을 통해 요청을 주고받는 입구(문서는 "messaging platforms" 라고만 통칭할 뿐 개별 채널명을 세부적으로 나열하지는 않습니다)[S6]
- 도구·검색 백엔드: 기본값은 Firecrawl(웹 페이지 추출 서비스)이고, 별도 설정(web.backend)이 없으면 사용 가능한 API 키를 기준으로 자동 감지되며, SearXNG·Brave·DuckDuckGo·Tavily·Exa·Parallel·xAI 등 대안 백엔드로 바꿀 수 있습니다[S7]
- 모델 제공자: 네이티브 Anthropic 연동, OpenRouter, Nous Portal 세 경로가 공식 문서에 언급되어 있습니다[S6]

이 목록에서 확인할 수 있듯, 도구·검색 축은 특정 서비스 하나에 묶여 있지 않고 일곱 개가 넘는 대안이 준비되어 있다는 점에서 교체 가능성이 비교적 뚜렷하게 문서화되어 있습니다[S7]. 반면 채널 축은 "메시징 플랫폼"이라는 통칭 수준에서만 설명되어 있어, 각 채널의 메시지가 구체적으로 어떤 경로를 거치는지는 이 문서만으로 세부까지 확인되지 않습니다[S6]. 이런 차이는 축마다 문서화 수준이 균일하지 않다는 뜻이므로, 자기 조직이 이미 쓰는 메신저나 모델이 이 구조에 맞물릴 수 있는지 확인하려는 담당자는 축별로 공개된 정보의 깊이가 다르다는 점을 염두에 두고 접근해야 합니다. 그럼에도 불구하고 세 축이 각각 별도로 교체 가능하게 설계되어 있다는 구조 자체는 공식 문서에 일관되게 나타나며, 이는 하나의 요소를 바꾸기 위해 나머지 요소까지 함께 바꿀 필요가 없다는 뜻이기도 합니다.



2.3 여덟 항목 대응표

2.3.1 기본 제공 일곱 항목

1장에서 정한 여덟 항목 — ① 절차 기억 ② 채널 통합 ③ 무인 실행 ④ 실행 격리 ⑤ 데이터 처리 위치 공개 ⑥ 권한 분리 ⑦ 위험 명령 승인 ⑧ 실행 결과 검증 — 을 Hermes Agent 의 공식 문서에 대 보면, 일곱 항목은 기능으로 명시되어 있습니다. 절차 기억은 앞서 살펴본 스킬 체계가 그대로 담당하고, 채널 통합은 방금 살펴본 메시징 채널 축이 담당합니다. 나머지 다섯 항목의 근거는 보안 문서와 기능 개요 문서에 흩어져 있는데, 그 근거를 한 번에 정리해 두는 것이 이 표의 역할입니다.

보안 문서는 보안 모델이 여덟 개 층으로 이루어져 있다고 설명합니다. "보안 모델은 여덟 개 층으로 구성된다: 1. 사용자 인가, 2. 위험 명령 승인, 3. 파일 쓰기 안전장치, 4. 컨테이너 격리, 5. MCP(모델 컨텍스트 프로토콜) 자격증명 필터링, 6. 컨텍스트 파일 스캔, 7. 세션 간 격리, 8. 입력 정제"라고 밝히고 있습니다[S3]. 이 여덟 개 층 가운데 컨테이너 격리는 실행 격리 항목의 근거이고, 위험 명령 승인은 그대로 같은 이름의 항목입니다. 세션 간 격리와 사용자 인가는 권한 분리 항목의 근거로 이어집니다. 데이터 처리 위치 공개는 앞서 살펴본 도구·모델 축

이 교체 가능하게 문서화되어 있다는 점, 그리고 백엔드 선택이 명시적 설정이나 API 키 유무로 결정된다는 점에서 근거를 찾을 수 있습니다[S7]. 무인 실행은 기능 개요 문서에서 확인됩니다[S6]. 이 대응 관계를 표로 정리하면 다음과 같습니다.

번호	항목	제공 여부	근거 출처	자세히 다루는 장
①	절차 기억	제공	[S4][S5]	3장
②	채널 통합	제공	[S6]	3장
③	무인 실행	제공	[S6]	4장
④	실행 격리	제공	[S3]	5장
⑤	데이터 처리 위치 공개	제공	[S6][S7]	6장
⑥	권한 분리	제공	[S3]	5장
⑦	위험 명령 승인	제공	[S3]	7장
⑧	실행 결과 검증	제공되지 않음	—	8장

이 표에서 "제공"으로 표시한 일곱 항목은 각각 3장부터 7장까지 하나씩 근거를 더 깊이 파고듭니다. 다만 이 표가 확정하는 것은 공식 문서에 기능으로 명시되어 있다는 사실이지, 그 기능이 실제 운영 환경에서 얼마나 견고하게 작동하는지까지는 아닙니다. 문서에 이름이 올라 있다는 것과 실무에서 검증되었다는 것은 다른 수준의 확인이므로, 그 깊이 있는 확인은 뒤따르는 각 장의 몫으로 남겨 둡니다. 다른 제품을 함께 검토하고 있는 담당자라면, 그 제품의 기능 목록을 옆에 놓고 이 표의 항목명을 그대로 대입해 보는 방식으로 비교 기준을 통일할 수 있습니다.

2.3.2 표에서 비어 있는 한 칸

여덟 번째 항목인 실행 결과 검증은 이 표에서 채워지지 않습니다. 보안 문서가 설명하는 여덟 개 층 어디에도, 그리고 기능 개요 문서 어디에도 에이전트가 실행한 작업의 결과가 실제로 맞는지 확인하는 절차에 해당하는 기능 서술이 없습니다[S3][S6]. 사용자 인가부터 입력 정제까지 여덟 개 층은 모두 무엇을 실행하도록 허락할지, 무엇을 실행 전에 승인받게 할지, 실행 범위를 어떻게 격리할지를 다루는 층이고, 실행이 끝난 뒤 그 결과가 의도한 대로 맞게 나왔는지를 자동으로 대조하는 층은 이 여덟 개 안에 들어 있지 않습니다.

이 빈 칸을 확인하는 과정에서 짚어야 할 점은, 이것이 제품의 결함을 지적하는 것이 아니라 문서화된 범위의 경계를 있는 그대로 확인하는 작업이라는 사실입니다. 보안 문서에 여덟 개 층이 명시되어 있다는 것은 이미 상당히 촘촘한 안전장치를 갖추고 있다는 뜻이지만[S3], 그 촘촘함이 "실행이 끝난 뒤 결과가 맞는지 확인한다"는 별개의 질문에 대한 답까지 포함하지는 않습니다. 승인 절차와 격리 장치가 잘못된 실행이 시작되는 것을 막아 주는 층이라면, 결과 검증은 실행이 이미 끝난 다음에 그 산출물이 실제로 맞는지 확인하는 층이고, 이 둘은 서로 다른 지점을 지킵니다.

"일곱 개나 되니 충분하다"고 이 대목을 그냥 넘기지 않는 것이 중요합니다. 이 빈 칸을 2장에서 미리 드러내 두어야 7장에서 정리할 안전장치 지도와 8장에서 다룰 도입 판단이 근거를 가질 수 있습니다. 이 항목을 뒤로 미루면 마치 여덟 항목이 모두 채워진 것처럼 읽히는 은폐가 되므로, 지금 이 자리에서 명확히 표시해 둡니다. 이 제

품을 검토하는 조직이라면, 자기 조직 안에서 실행 결과 검증을 누가 맡을 것인지를 미리 물어보는 편이 좋습니다. 이 질문에 대한 구체적인 논의는 8장에서 다시 이어집니다.

3장: 스킬 — 한 번 시킨 일을 다시 시키지 않는다

1장에서 정리한 필수 기능 여덟 가지 가운데 첫 번째는 절차 기억이었습니다. 같은 업무를 두 번째, 세 번째 시킬 때마다 처음부터 맥락을 다시 설명해야 한다면, 에이전트를 도입한 효과는 반복될 때마다 줄어듭니다. 이 장은 그 절차 기억이 Hermes Agent 안에서 스킬이라는 구체적인 메커니즘으로 어떻게 구현되는지, 그리고 그 메커니즘의 보장 범위가 공식 문서상 어디까지인지를 정리합니다. 스킬이 무엇인지 정의하는 데서 시작해, 스킬이 누구에 의해 언제 쌓이는지를 따라가고, 마지막으로 버전이 바뀔 때 무엇이 보장되고 무엇이 확인되지 않는지를 가릅니다.

3.1 스킬이라는 절차 기억

3.1.1 온디맨드로 불러오는 지식 문서

스킬은 에이전트가 필요할 때 불러 쓰는 온디맨드(on-demand, 필요 시점) 지식 문서입니다. 공식 문서는 스킬을 "점진적 공개(progressive disclosure) 방식을 따르는, 필요할 때 불러 쓰는 지식 문서"로 설명합니다 [S4]. 절차 전체를 한 번에 다 읽어들이는 대신, 지금 하려는 작업과 관련된 부분만 그때그때 펼쳐 보는 방식이라는 뜻입니다.

이 방식이 해결하는 문제는 "매번 같은 설명을 반복해야 하는가"라는 실무 불만입니다. 특정 업무 절차를 한 번 문서로 남겨두면, 다음번에는 그 절차를 처음부터 구술할 필요 없이 에이전트가 필요한 순간에 문서를 열람합니다. 절차 하나하나가 대화창에 매번 다시 타이핑되지 않아도 되는 구조라는 점이 스킬의 핵심입니다 [S4].

다만 이 온디맨드 구조가 무엇을 언제 불러올지 판단하는 알고리즘의 정밀도까지 보증하는 것은 아닙니다. 점진적 공개라는 원칙은 확인되지만, 스킬이 많아졌을 때 관련 없는 스킬을 얼마나 정확히 걸러내는지에 대한 정량적 수치는 공식 문서에서 확인되지 않았습니다. 그럼에도 구조 자체가 "쌓일수록 매 실행마다 전체를 다시 읽는" 방식이 아니라는 점은 분명합니다.

이 설계가 의미하는 바는 단순합니다. 조직이 스킬을 열 개 쌓든 백 개 쌓든, 실행 한 번에 드는 비용이 스킬 개수에 비례해 무한정 늘어나지 않는 구조를 전제로 한다는 것입니다. IT 담당자 입장에서는 스킬 라이브러리를 키우는 것이 실행 부담을 키우는 방향으로만 작동하지 않는다는 뜻으로 읽을 수 있습니다.

전통적인 방식과 비교하면 이 차이가 더 뚜렷해집니다. 매 실행마다 전체 지침을 다시 읽어들이는 구조라면, 담당자가 절차 하나를 새로 추가할 때마다 다음 실행의 부담이 함께 늘어나는 셈이 됩니다. 반면 필요한 순간에만 필요한 만큼 펼쳐 보는 구조에서는, 새 절차를 추가하는 일과 다음 실행의 비용이 곧바로 비례하지 않습니다. 이 차이는 스킬 라이브러리를 적극적으로 키우려는 조직일수록 체감하게 되는 부분입니다.

3.1.2 공개 표준과의 호환

스킬은 agentskills.io 라는 공개 표준과 호환됩니다 [S4]. 이 제품만을 위해 고안된 전용 포맷이 아니라, 다른 도구에서도 통용될 수 있는 규격을 따른다는 뜻입니다. 프런트매터와 본문 구조가 특정 벤더의 폐쇄적 스키마가 아니라 외부에 공개된 명세를 따릅니다.

이 사실이 중요한 이유는 절차 기억이 특정 제품 안에 갇히지 않는다는 데 있습니다. 조직이 시간을 들여 만든 스킬 자료가 그 제품의 내부 포맷으로만 읽히는 자산이라면, 도구를 바꾸는 순간 그 자산의 가치가 함께 사라질 위

험이 있습니다. 공개 표준을 따른다는 것은 최소한 그 형식 자체는 다른 도구도 읽어낼 수 있는 구조로 남아있다는 뜻입니다.

물론 표준 호환이 곧바로 원클릭 이전을 보장하지는 않습니다. 형식이 같은 표준을 따른다는 사실과, 그 형식을 실제로 읽어 들이는 도구가 시장에 얼마나 있는지, 이전 절차가 얼마나 매끄러운지는 별개의 문제입니다. 공식 문서는 호환성 자체만 확인해 줄 뿐, 이식 경험의 수월함까지 보장하지는 않습니다.

그럼에도 이 지점은 "이 제품을 나중에 바꾸면 지금까지 만든 스킬이 사라지는가"라는 질문에 대한 직접적인 답이 됩니다. 답은 "적어도 형식 차원에서는 사라지지 않는다"입니다. 절차 기억을 특정 제품 전용 형식에 가두지 않았다는 사실은, 장기적으로 조직이 특정 벤더에 종속되는 위험을 낮추는 방향으로 작동합니다. 잠금(lock-in) 문제를 걱정하는 도입 담당자라면 이 연결을 눈여겨볼 지점입니다.

여러 조직이 도구를 오래 쓰면서 겪는 흔한 어려움은, 그 도구 안에서 축적한 자료가 그 도구를 벗어나는 순간 다시 만들어야 하는 자료로 바뀌는 경우입니다. 절차 문서가 표준 규격을 따른다는 사실은 이 위험을 완전히 없애지는 못해도, 적어도 형식 자체가 벤더 종속의 원인이 되지는 않는다는 최소한의 안전판을 제공합니다. 도입 전 검토 단계에서 스킬 자산의 향후 활용 가능성을 따지는 담당자라면 이 지점을 체크리스트에 넣을 만합니다.

3.2 만들어지고 쌓이는 과정

3.2.1 실행 중 자유롭게 쓰는 스킬

에이전트는 작업을 처리하는 도중에도 스킬을 자유롭게 씁니다. 정해진 트리거나 사람의 "이 절차를 스킬로 저장하라"는 명시적 지시 없이도, 에이전트가 판단해서 절차를 문서로 남깁니다 [S4]. 다시 말해 스킬을 쓰는 주체는 사람만이 아닙니다.

이 흐름이 갖는 의미는 "누가 스킬을 관리하는가"라는 질문에 있습니다. 사람이 일일이 나서서 절차를 문서화하지 않아도, 에이전트가 작업을 수행하면서 스스로 절차 기억을 쌓아 나갑니다. 반복되는 업무 패턴을 사람이 뒤늦게 알아채고 정리해 주는 방식이 아니라, 작업이 진행되는 그 순간 절차가 함께 기록되는 방식입니다.

이 자율성에는 한계도 함께 따라옵니다. 스킬 생성이 실시간으로, 사람의 승인 없이 이루어질 수 있다는 것은 곧 어떤 절차가 스킬로 남을지에 대한 품질 관리가 전적으로 결정론적이지는 않다는 뜻이기도 합니다. 어떤 절차를 남길지에 대한 판단은 에이전트 몫이며, 이 판단이 매번 조직이 원하는 수준으로 정제되어 있으리라는 보장은 별도로 확인되지 않습니다.

도입하는 조직 입장에서 이 지점이 주는 함의는, 스킬 라이브러리가 관리자의 개입 없이도 자연스럽게 불어난다는 편의와, 그렇게 쌓인 스킬을 주기적으로 들여다볼 필요성이 함께 딸려온다는 것입니다. 스킬이 쌓이는 속도와 그 내용을 검토하는 절차는 별개의 문제로 남습니다.

이 편의와 필요성의 균형은 결국 7장에서 다룰 실행 결과 검증과 맞닿습니다. 절차가 자동으로 문서화된다는 것과, 그 절차가 실제로 옳은 방향으로 정리되었는지는 서로 다른 층위의 질문이기 때문입니다. 스킬이 쌓이는 과정 자체는 여기서 확인되지만, 쌓인 스킬의 품질을 어떻게 지속적으로 담보할지는 이 장의 범위를 넘어서는 별도의 검증 장치가 필요한 영역입니다.

3.2.2 세션이 끝난 뒤의 자기개선 리뷰

턴이 끝난 뒤 백그라운드에서 도는 자기개선 리뷰 과정에서도 스킬이 만들어집니다 [S4]. 이 리뷰는 사용자가 화면을 보고 있는 시점이 아니라, 대화 턴이 종료된 이후에 별도로 실행되는 과정입니다. 즉 사람이 지켜보지 않

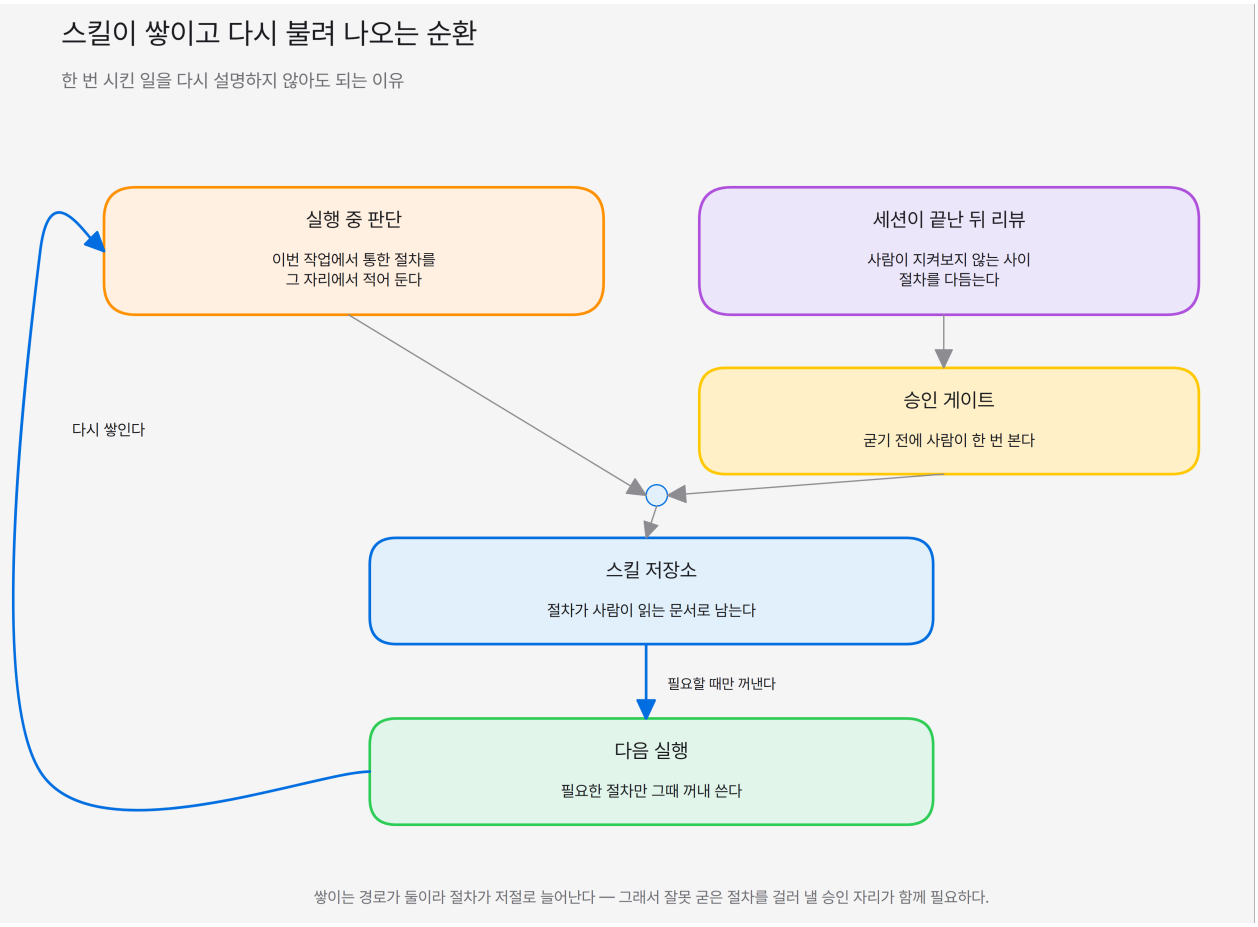
는 시점에도 절차 기억이 갱신될 수 있습니다.

이 사실이 중요한 이유는 절차 기억이 실시간 작업 중에만 쌓이는 것이 아니라 사후 회고를 거쳐서도 쌓인다는 데 있습니다. 방금 처리한 작업을 되짚어보고 개선할 점이 있으면 그 결과를 다시 스킬 형태로 남기는 구조입니다. 이 지점은 4장에서 다룰 무인 실행 문제와 맞물립니다 — 사람이 개입하지 않는 시간에도 시스템이 스스로 무언가를 실행한다는 점에서, 절차 기억의 축적과 무인 실행의 위험은 같은 뿌리를 공유합니다.

다만 이 백그라운드 리뷰가 무엇을 스킬로 남길지 판단하는 기준이 사람이 지켜보는 실시간 작업 때와 얼마나 다른지, 혹은 같은지는 공식 문서에서 별도로 구분되어 있지 않습니다. 사후 리뷰라는 시점의 차이만 확인될 뿐, 판단 기준 자체의 차이는 확인되지 않습니다.

이 구조가 실무적으로 의미하는 바는, "이 제품이 밤새 뭘 하는가"라는 질문에 대한 구체적인 답 가운데 하나가 바로 이것이라는 점입니다. 세션이 끝난 뒤에도 시스템은 방금 처리한 작업을 돌아보고, 다음번에 같은 일을 더 잘 처리할 수 있도록 절차를 문서로 정리해 둡니다. 사람이 자리를 비운 시간이 곧 아무 일도 일어나지 않는 시간은 아니라는 뜻입니다.

이 지점에서 짚어야 할 대가도 있습니다. 사람이 지켜보지 않는 시점에 절차가 문서로 굳어진다는 것은, 그 절차가 잘못 연결된 채로 굳어질 가능성도 함께 안고 간다는 뜻입니다. 절차 자체가 틀렸는데도 에이전트는 멈추지 않고 그럴듯하게 다듬어진 결과를 계속 내놓을 수 있습니다. 이런 상황을 막는 장치가 바로 위험 명령 승인 게이트이며, 배경 자기개선 리뷰가 만든 스킬에도 승인 게이트를 걸 수 있다는 사실은 이 위험을 인지한 설계임을 보여줍니다 [S4]. 절차 기억이 왜 실행 결과 검증이라는 여덟 번째 항목과 짝을 이뤄야 하는지가 여기서 드러나며, 이 관계는 7장에서 다시 다룹니다.



3.3 버전과 갱신 경계

3.3.1 프런트매터 버전과 origin hash

스킬 문서의 프런트매터에는 버전 필드가 있습니다 [S4]. 여기에 더해 번들로 제공되는 스킬은 origin hash 라는 값으로 로컬에서 수정되었는지 여부를 대조하는 메커니즘을 갖습니다 [S4]. 업데이트 절차인 `hermes update` 는 코드를 한 번만 내려받아 갱신한 뒤, 갱신된 스킬을 전체 프로파일에 자동으로 동기화합니다 [S5].

이 메커니즘이 해결하는 문제는 여러 프로파일을 운영하는 조직에서 두드러집니다. 프로파일마다 따로 업데이트 명령을 실행하고 스킬을 일일이 맞춰야 한다면, 프로파일 수가 늘어날수록 운영 부담도 함께 늘어납니다. 코드 갱신은 한 번, 스킬 동기화는 자동이라는 구조는 이 부담을 줄이는 방향으로 설계되어 있습니다 [S5].

다만 이 메커니즘이 보장하는 범위는 "동기화가 자동으로 일어난다"는 사실까지입니다. 동기화 이후 각 프로파일에서 실제로 같은 버전의 스킬이 똑같이 동작하는지, 혹은 프로파일별 설정 차이가 동기화 결과에 영향을 주는지에 대한 세부 사항까지 공식 문서가 낱낱이 규정하지는 않습니다.

이 지점은 여러 팀, 여러 프로파일을 운영할 계획이 있는 조직에게 분명한 안내가 됩니다. 업데이트 한 번으로 전체가 맞춰지는가는 운영 부담을 가르는 실질적인 기준이며, 이 부분은 도입하는 조직이 별도로 설계하지 않아도 제품이 대신 처리해 주는 몫입니다.

프로파일을 부서별, 업무별로 나누어 운영하는 조직이라면 이 차이가 특히 크게 느껴집니다. 프로파일 다섯 개를 운영하는 조직이 업데이트가 나올 때마다 다섯 번의 수동 동기화 작업을 반복해야 한다면, 이 반복 작업 자체가 하나의 운영 비용으로 쌓입니다. 코드 갱신과 스킬 동기화가 한 번의 명령으로 전체 프로파일에 퍼진다는 구조는 이 반복 비용을 애초에 설계에서 제거한 것입니다 [S5].

3.3.2 로컬 수정 스킬의 생존

origin hash 로 로컬 수정 여부를 대조하는 메커니즘은 분명히 존재합니다 [S4]. 하지만 이 메커니즘이 업데이트가 실행되는 동안 로컬에서 손을 본 스킬을 그대로 보존해 주는지, 아니면 해시로 추적되지 않는 변경분을 초기화해 버리는지는 공식 문서에 명시되어 있지 않습니다 [S4] [S5].

FAQ 문서는 "`hermes update` 가 스킬을 전체 프로파일에 자동으로 동기화한다"는 사실만 밝히고 있을 뿐 [S5], 로컬 수정과 원본 갱신이 충돌할 때 어느 쪽이 우선하는지에 대한 규정은 확인되지 않습니다. origin hash 라는 대조 장치가 있다는 사실과, 그 장치가 로컬 수정분을 보호하는 방향으로 작동한다는 사실은 서로 다른 주장이며, 지금 확인할 수 있는 것은 앞의 것뿐입니다.

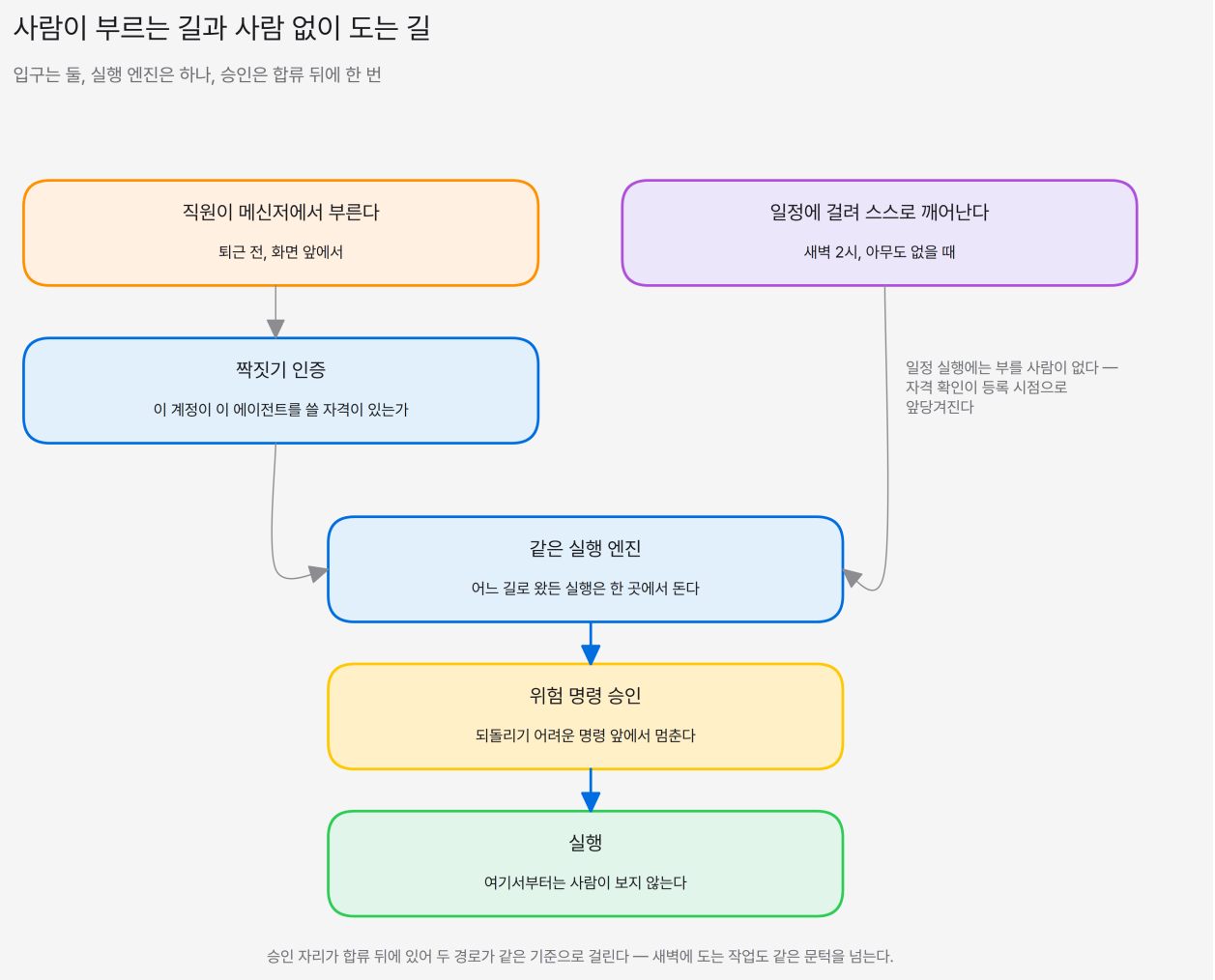
이 지점은 추측으로 채우지 않는 편이 정직합니다. 스킬을 직접 고쳐 쓰는 조직이라면, "우리가 손본 스킬이 다음 업데이트로 사라지지는 않는가"라는 질문에 공식 문서만으로는 답이 확정되지 않습니다. 로컬 수정이 완전히 안전하다고 단정할 근거도, 완전히 위험하다고 단정할 근거도 현재로서는 부족합니다.

그렇다고 이 지점을 그냥 넘어갈 수는 없습니다. 스킬을 표준 그대로 쓰는 조직이라면 이 문제가 크게 부각되지 않지만, 조직 사정에 맞춰 번들 스킬을 직접 손봐서 쓰는 운영 방식을 택했다면 이야기가 달라집니다. 업데이트를 실제 운영 환경에 적용하기 전에, 테스트 프로파일 하나를 따로 두고 로컬 수정분이 어떻게 처리되는지 먼저 확인해 보는 절차를 두는 편이 안전합니다. 이 확인 한 번이 이후의 업데이트 주기마다 반복해서 겪을 수 있는 불확실성을 줄여 줍니다.

여기서 이 장을 닫는 기준을 정리할 수 있습니다. 제품이 대신하는 것은 분명합니다 — 버전 필드를 통한 식별, 원본과의 대조, 전체 프로파일에 걸친 자동 동기화까지는 메커니즘으로 확인됩니다. 반면 도입하는 쪽 몫으로 남는 것도 있습니다 — 스킬을 직접 고쳐 쓰는 운영을 계획한다면, 업데이트가 그 수정분에 어떤 영향을 주는지를 실제 환경에서 먼저 확인해 보는 일입니다. 절차 기억이라는 첫 번째 항목은 이렇게 제품이 보장하는 경계와 조직이 확인해야 할 경계를 함께 남긴 채로, 다음 장의 채널 통합으로 이어집니다.

4장: 채널과 무인 실행

1장에서 확정한 여덟 항목 가운데 이 장은 두 번째 항목인 채널 통합과 세 번째 항목인 무인 실행을 다룹니다. 절차 기억이 "무엇을 반복할지"를 정하는 항목이라면, 채널 통합은 "그 반복을 어디서 불러 쓰는가"를 정하는 항목이고, 무인 실행은 "그 반복이 사람 없이도 끝까지 도는가"를 정하는 항목입니다. IT 담당자가 실제로 던지는 물음은 두 가지로 좁혀집니다. 하나는 직원들이 어디서 에이전트를 불러 쓰는가이고, 다른 하나는 퇴근한 뒤에 도는 작업은 어떻게 거는가입니다. 이 장은 이 두 물음을 4.1과 4.2에서 각각 다루고, 4.3에서 무인 실행의 결과를 나중에 확인할 수단이 지금 어디까지 갖춰져 있는지로 마무리합니다.



4.1 채널 연결

4.1.1 메시징 플랫폼 개관

채널을 여는 일은 결국 접근 경로를 여는 일입니다. Hermes Agent는 별도 전용 콘솔을 새로 배우게 하는 대신, 조직이 이미 쓰고 있는 메시징 플랫폼을 통해 요청을 받는 구조를 취합니다. 공식 기능 개요 문서는 어떤 메시징 플랫폼이 채널로 연결되는지를 밝히고 있습니다[S6]. 직원 입장에서는 새 도구를 켜는 대신 평소에 쓰던 대화창에 메시지를 남기는 것으로 요청이 시작된다는 뜻입니다.

이 목록이 IT 담당자에게 갖는 무게는 기능 나열 이상입니다. 도입을 검토하는 조직마다 이미 굳어진 메신저 습관이 있고, 그 습관을 바꾸라고 요구하는 순간 도입 자체가 늦어지기 때문입니다. 그래서 "우리 팀이 쓰는 메신저로 바로 시킬 수 있나"라는 질문은 기능 설명서를 읽기 전에 가장 먼저 던져야 하는 질문이 됩니다. 이 질문에 대한 답은 공식 문서의 채널 목록을 그대로 대조하는 것으로 확인할 수 있습니다[S6].

다만 이 개관에는 한계가 있습니다. 공식 문서는 어떤 플랫폼이 연결되는지는 밝히지만, 채널별 메시지가 내부에서 어떤 경로를 거치는지, 즉 별도 게이트웨이 서버를 지나는지 에이전트 프로세스에 직접 이어지는지 같은 세부 경로까지는 밝히지 않습니다[S6]. 이 공백을 감추지 않고 그대로 짚어 두는 이유는, 메시지 경로를 보안 검토 항목에 넣어야 하는 조직이라면 이 개관만으로는 검토가 끝나지 않는다는 점을 미리 알아야 하기 때문입니다. 결국 채널 목록은 도입 문턱을 가늠하는 첫 필터일 뿐, 내부 경로 검토를 대신해 주지는 않습니다.

이 한계가 함의하는 바는 채널 연결을 순수하게 편의 기능으로만 보면 안 된다는 것입니다. 채널을 하나 여는 순간, 그 채널은 조직 바깥에서 에이전트로 들어오는 경로가 되기도 합니다. "직원들이 어디서 불러 쓰나"라는 물음에 대한 답이 곧 "어디서부터 접근을 통제해야 하나"라는 물음의 답이기도 한 이유입니다. 다음 항에서 다루는 짝짓기 인증은 이 접근 경로를 지키는 최소한의 장치이자, 채널을 여는 결정과 분리해서 생각할 수 없는 전제입니다.

4.1.2 DM 짝짓기 인증

채널을 여는 일이 접근 경로를 여는 일이라면, 그 경로에 아무나 들어올 수 없도록 막는 장치가 함께 있어야 합니다. Hermes Agent는 메신저 계정과 에이전트 사용 권한을 연결할 때 짝짓기 인증 절차를 거치도록 설계되어 있습니다. 공식 보안 문서는 이 절차를 구체적인 수치로 명시합니다. 인증 코드는 8자리이고 32자 비모호 알파벳(혼동되기 쉬운 문자를 뺀 문자 집합)으로 구성되며, 발급된 코드는 1시간이 지나면 만료됩니다[S3].

이 수치들이 중요한 이유는 추상적인 "인증을 거친다"는 말과 다르게, IT 담당자가 자기 조직의 보안 기준과 직접 대조할 수 있는 구체적인 항목이라는 데 있습니다. 요청 제한도 함께 걸려 있어서 같은 사용자가 짝짓기 코드를 요청할 수 있는 빈도는 10분에 1회로 제한되고, 한 플랫폼에 동시에 대기할 수 있는 코드는 최대 3개까지입니다. 인증에 다섯 번 연속 실패하면 해당 시도는 1시간 동안 잠깁니다[S3]. 아래 표는 이 다섯 가지 수치를 그대로 정리한 것입니다.

항목	값
코드 길이	8자리, 32자 비모호 알파벳
유효 기간(TTL)	1시간
요청 제한	사용자당 10분에 1회
대기 코드 상한	플랫폼당 최대 3개
잠금 조건	실패 5회 시 1시간 잠금

이 수치들은 채널 인증이 형식적인 절차가 아니라 실제로 재현 공격이나 무차별 대입 시도를 어렵게 만들도록 설계되었다는 근거이기도 합니다. 코드 길이와 TTL, 요청 제한, 잠금 조건이 서로 맞물려 있어서 어느 하나만 따로 떼어 평가하기보다는 조합으로 봐야 합니다. 다만 이 표는 짝짓기 시점의 인증 강도를 보여줄 뿐, 짝짓기가 끝난 뒤 세션이 얼마나 오래 유지되는지, 재인증 주기가 있는지는 공식 보안 문서에서 별도로 다루는 항목입니다. IT

담당자가 이 절을 읽고 나서 할 일은 명확합니다. 이 다섯 수치를 자기 조직의 접근 통제 기준과 나란히 놓고, 기준에 못 미치는 항목이 있는지 확인하는 것입니다.

4.2 퇴근 후 무인 실행

4.2.1 무인 실행이 성립하는 조건

채널이 입구라면, 그 입구로 들어온 요청이 사람 없이 끝까지 처리되는 조건이 무인 실행입니다. 여기서 짚어야 할 첫 번째 사실은 무인 실행이 저절로 주어지는 기능이 아니라 환경을 갖춰야 성립하는 기능이라는 점입니다. 채널로 들어온 요청이 사람의 추가 승인 없이 끝까지 실행되려면, 에이전트 프로세스가 그 요청이 들어오는 동안 계속 떠 있어야 합니다[S2][S5]. 프로세스가 꺼져 있으면 메신저로 메시지를 보내도 응답할 주체가 없으므로, 무인 실행은 논리적으로 상시 구동 환경을 전제로 합니다.

이 전제는 "퇴근 후에도 일을 시킬 수 있다"는 말을 IT 담당자가 어떻게 받아들여야 하는지를 결정합니다. 이 말은 기능 하나를 켜면 끝나는 이야기가 아니라, 인프라 쪽에서 별도로 준비해야 할 조건이 딸려 있다는 뜻입니다. 공식 저장소 설명은 이 제품을 5달러짜리 VPS(가상 사설 서버)에서도, GPU 클러스터에서도, 서버리스 인프라에서도 돌릴 수 있다고 밝히고 있어[S2], 구동 환경의 폭 자체는 넓은 편입니다. 하지만 폭이 넓다는 것과 그 폭 안의 특정 환경에서 상시 구동이 안정적으로 이루어진다는 것은 다른 이야기입니다.

이 구조적 논증이 갖는 함의는 분명합니다. 무인 실행을 도입 체크리스트에 올릴 때는 "이 기능을 켤 것인가"가 아니라 "이 기능이 밤새 떠 있을 인프라를 우리가 갖추고 있는가"를 먼저 물어야 합니다. 다음 항에서 다루는 상시 구동 환경의 제약은 바로 이 물음에 대한 답을 좁혀 줍니다.

무인 실행에는 또 하나의 전제가 딸려 있습니다. 사람이 없는 시간에 도는 작업이라는 것은, 그 시간 동안 무엇을 에이전트가 스스로 결정하고 무엇을 사람의 승인 없이는 넘어가지 못하게 막을지를 미리 정해 두어야 한다는 뜻이기도 합니다. 낮 시간이라면 애매한 판단이 나왔을 때 담당자에게 바로 물어보면 되지만, 퇴근 후에는 그 되물을 상대가 없습니다. 그래서 무인 실행은 승인 설계와 따로 떼어 다룰 수 없는 한 몸입니다. 이 승인 설계를 구체적으로 어떻게 짜는지는 7장에서 위험 명령 승인을 다룰 때 자세히 이어집니다. 여기서는 무인 실행을 검토하는 시점에 이미 승인 범위를 함께 정해야 한다는 점만 짚어 둡니다.

4.2.2 상시 구동 환경의 제약

무인 실행이 상시 구동을 전제로 한다면, 그 전제가 흔들리는 환경도 미리 알아 두어야 합니다. 공식 FAQ(자주 묻는 질문) 문서는 WSL(Windows Subsystem for Linux, 윈도우용 리눅스 하위 시스템)의 systemd(서비스 관리 데몬) 지원이 불안정하다고 명시적으로 밝히고 있습니다[S5]. 많은 WSL2 설치본에는 systemd가 기본적으로 꺼져 있고, 켜져 있는 경우에도 WSL 재시작이나 Windows의 유휴 종료를 넘기지 못하고 죽는 경우가 있다고 공식 문서는 적고 있습니다[S5].

이 제약이 중요한 이유는 사내 인프라의 상당수가 여전히 Windows 위에서 돌아가기 때문입니다. Windows 환경에서 상시 구동을 계획하는 조직이라면, 이 제약은 무인 실행의 신뢰성을 직접 깎아 먹는 요인이 됩니다. 밤사이 프로세스가 재시작되거나 종료되면, 그 시점 이후로 들어온 채널 요청은 아무리 짝짓기 인증을 통과했어도 응답을 받을 주체가 없는 상태가 됩니다. 이 문제는 기능 설정을 바꾼다고 해결되는 종류가 아니라 구동 환경 자체를 점검해야 풀리는 종류입니다.

여기서 감추지 말아야 할 것은 이 제약이 예외적인 사례가 아니라 공식 문서가 스스로 밝힌 알려진 한계라는 점입니다[S5]. 동시에 이 한계를 넘어서서 단정하지 않아야 할 것도 있습니다. 공식 문서는 systemd 지원이 불안

정하다는 사실과 그 구체적인 실패 양상 두 가지를 밝히지만, 어떤 대체 구성이 이 문제를 완전히 없애는지는 다루지 않습니다. 그래서 이 장이 줄 수 있는 조언은 하나로 좁혀집니다. Windows 사내 인프라를 쓰는 IT 담당자라면 문서의 서술을 그대로 믿기보다, 자기 환경에서 며칠 밤 연속으로 프로세스가 살아 있는지 직접 확인해 보는 편이 안전합니다.

이 제약을 4.2.1에서 짚은 승인 설계와 함께 놓고 보면 위험은 한 겹 더 늘어납니다. 상시 구동이 중간에 끊기면 그 시점까지 처리되던 작업이 어느 단계에서 멈췄는지, 승인이 필요한 지점에서 대기하다 끊긴 것인지, 아니면 실행 도중에 끊긴 것인지가 곧바로 드러나지 않을 수 있습니다. 이 문제 역시 뒤에서 다루는 실행 결과 확인과 맞닿아 있습니다. 상시 구동이 흔들릴 수 있다는 사실을 알고 있어야, 다음 절에서 다루는 결과 확인의 필요성도 같은 무게로 받아들일 수 있습니다.

4.3 무인 실행 결과를 나중에 들여다보기

4.3.1 이미 있는 계측 조각들

채널이 열려 있고 프로세스가 밤새 떠 있어서 요청이 실제로 처리되었다고 해도, 그다음 물음이 남습니다. 그 실행이 잘 끝났는지 아침에 무엇으로 확인하느냐는 물음입니다. 이 물음에 대해 아무것도 없는 것은 아닙니다. 제품은 이미 사용량을 들여다볼 수 있는 계측 조각들을 갖고 있습니다. 토큰 회계, 추정 비용, 세션 길이, 그리고 인사이트 기능이 그 조각들입니다. 개발 커뮤니티가 남긴 열린 이슈도 이 데이터가 이미 존재한다고 스스로 확인하고 있습니다[S9].

- 토큰 회계
- 추정 비용
- 세션 길이
- 인사이트(/insights)

이 네 가지가 뜻하는 바는, 밤새 무인 실행이 몇 번이나 돌았는지, 그 실행에 비용이 얼마나 들었는지는 이미 들여다볼 수 있는 정보라는 것입니다. 그래서 밤새 뭘 했는지 전혀 모른 채 아침을 맞는 것은 아닙니다. 이 부분은 조직이 별도로 만들지 않아도 제품이 미리 갖춰 준 몫입니다. IT 담당자가 아침에 세션 길이와 추정 비용을 확인하는 것만으로도 무인 실행이 과도하게 돌지는 않았는지, 예상 밖의 비용이 발생하지는 않았는지 정도는 가늠할 수 있습니다.

4.3.2 조각이 하나로 묶이지 않았다는 사실

하지만 이 계측이 사용량 확인에는 쓸 수 있어도, 실행 결과가 실제로 맞게 끝났는지를 확인하는 용도로는 아직 부족합니다. 같은 열린 이슈는 이 계측이 조각으로 흩어져 있고, 통합된 텔레메트리(원격 계측) 계층이 없다고 분명히 밝히고 있습니다[S9]. 토큰 회계와 추정 비용, 세션 길이는 각각 따로 존재하는 지표일 뿐, 이 지표들을 하나로 묶어서 "이 밤 사이 실행된 작업이 의도한 대로 끝났는가"를 한눈에 확인하는 절차는 지금 존재하지 않습니다.

이 공백은 우연히 남은 빈틈이 아닙니다. 이 공백은 2장에서 정리한 여덟 항목의 대응표 가운데 여덟 번째 항목, 즉 실행 결과 검증이 비어 있던 자리와 정확히 같은 자리입니다. 사용량은 조각으로 보이지만 결과의 정확성은 조각으로도 보이지 않는다는 차이가 있습니다. 그리고 무인 실행이 늘어날수록 이 공백의 무게는 가벼워지지 않고 오히려 커집니다. 채널이 늘고 밤새 도는 작업이 늘수록, 그 결과를 사람이 일일이 열어서 확인해야 하는 부담도 함께 늘어나기 때문입니다.

그래서 이 장을 닫으며 남기는 물음은 하나입니다. 그래서 결과 확인은 누가 하는가. 이 장에서 정리한 두 항목을 놓고 보면 몫이 갈립니다. 채널을 열고 짝짓기 인증으로 그 채널을 지키는 일, 그리고 상시 구동 환경만 갖추면 요청을 사람 없이 받아 처리하는 일까지는 제품이 기본으로 제공하는 몫입니다. 반면 그 실행이 실제로 옳게 끝났는지를 조각난 계측을 넘어 하나로 확인하는 일은, 지금 이 시점에는 도입하는 조직이 스스로 절차를 만들어 채워야 하는 몫으로 남아 있습니다. 이 물음은 8장에서 조직이 채울 항목으로 다시 등장합니다.

5장: 도구 실행과 격리

앞선 4장은 사람 없이 작업이 진행되는 무인 실행을 다뤘습니다. 무인 실행이 성립하려면 에이전트가 도구를 실제로 실행할 수 있어야 하고, 동시에 그 실행이 어디까지 나갈 수 있는지 경계가 있어야 합니다. 1장에서 확정한 여덟 가지 필수 기능 가운데 이 장이 다루는 항목은 넷째인 실행 격리(도구가 시스템 자원에 접근하는 범위를 물리적·논리적으로 가두는 장치)입니다. 이 장은 도구 호출이 실제로 지나가는 경로를 먼저 보여주고, 그 경로가 어디서 멈추도록 설계되어 있는지를 이어서 설명합니다. 여덟 가지 안전장치 전체를 한눈에 정리하는 지도는 7장에서 다시 다루므로, 여기서는 같은 목록을 반복하지 않고 실행 경로와 차단 지점에만 집중합니다.

5.1 도구가 실행되는 경로

에이전트가 "도구를 부른다"는 말은 자칫 추상적으로 들립니다. 실제로 이 말이 가리키는 동작은 미리 정의된 연동 지점을 거쳐 특정 외부 서비스를 호출하는 것이고, 어떤 연동이 열려 있는지는 즉흥적으로 정해지지 않습니다. 이 절에서는 두 가지 구체적인 사례로 그 경로를 보여줍니다.

5.1.1 외부 도구를 연동하는 경로

에이전트가 조직의 기존 도구에 손을 댈 수 있는지 여부는 사전에 정의된 연동 지점을 통해서만 결정됩니다. 공식 Integrations(연동) 문서는 어떤 종류의 외부 서비스가 연결 대상으로 나열되어 있는지를 목록으로 제공합니다[S7]. 이 목록에 없는 서비스는 즉석에서 임의로 호출되지 않고, 반대로 목록에 있는 서비스는 별도의 커스텀 개발 없이 바로 연결 지점을 통해 붙습니다.

IT 담당자 입장에서 이 항목이 중요한 이유는 단순합니다. 도입을 검토할 때 가장 먼저 묻는 질문이 "우리가 이미 쓰고 있는 도구를 여기 붙일 수 있는가"이기 때문입니다. 연동 문서에 나열된 항목을 확인하는 절차가 그 질문에 대한 가장 직접적인 답이 되고, 목록에 없다면 별도 구성이나 개발이 필요하다는 뜻으로 읽어야 합니다.

다만 이 구조에는 분명한 한계가 있습니다. 연동 문서가 나열하는 것은 "연결이 가능한 지점"이지 "이미 검증되어 안전하다고 보장된 지점"은 아닙니다. 새로 붙이는 연동마다 그 서비스가 요구하는 자격증명과 권한 범위를 조직이 직접 검토해야 하며, 이 부분은 6장에서 다루는 데이터 처리 위치, 7장에서 다루는 권한 분리 문제와 맞물립니다. 이 장에서는 "연동 지점이 사전에 정의되어 있다"는 구조 자체에만 집중합니다.

이 구조가 실제로 어디까지 이어지는지도 짚어 둘 필요가 있습니다. 공식 보안 문서는 여덟 계층 가운데 하나로 MCP(Model Context Protocol, 도구·서비스를 표준화된 방식으로 연결하는 프로토콜) 자격증명 필터링을 명시하고 있습니다[S3]. 이 계층이 별도로 존재한다는 사실 자체가, 연동 지점을 늘리는 통로와 그 통로로 흘러 들어가는 자격증명을 걸러내는 장치가 애초에 분리되어 설계되어 있음을 보여줍니다. 즉 연동을 여는 결정과 그 연동이 자격증명을 어떻게 다루는지 확인하는 절차는 같은 항목이 아니라 별개의 항목으로 다뤄야 합니다.

5.1.2 웹 검색·추출 도구 호출 사례

앞서 말한 연동 지점 가운데 가장 자주 쓰이는 사례가 웹 검색·추출 도구입니다. 이 도구는 별도 설정을 하지 않아도 기본적으로 동작하도록 되어 있고, 기본 백엔드(요청을 실제로 처리하는 서비스)는 Firecrawl 입니다[S7]. 사용자가 검색이나 웹 페이지 추출을 요청하면 에이전트는 이 백엔드로 요청을 넘기고, 결과를 받아 작업에 반영합니다.

여기서 IT 담당자가 미리 알아야 할 동작 방식이 하나 있습니다. 웹 백엔드 설정(web.backend)을 명시적으로 지정하지 않으면, 시스템이 사용 가능한 API 키를 기준으로 백엔드를 자동으로 감지합니다[S7]. 즉 아무 설정도 건드리지 않아도 무언가 동작하기 시작하는 구조입니다.

이 자동 감지 방식은 언뜻 편리해 보이지만 실무에서는 다르게 읽어야 합니다. 조직이 다른 목적으로 특정 서비스의 API 키를 환경에 등록해 두면, 그 키가 존재한다는 사실만으로 웹 검색 백엔드가 조직이 의도하지 않은 서비스로 바뀔 수 있습니다. 편의 기능이 곧 구성 관리 항목이 된다는 뜻이고, 이 함의는 5.3.2에서 다시 구체적으로 다룹니다.

이 사례를 5.1.1에서 본 연동 지점 구조와 겹쳐 보면 도구 호출 경로 전체의 성격이 드러납니다. 연동 지점은 미리 정의되어 있지만, 그 지점 가운데 어떤 서비스가 실제로 켜지는지는 환경 설정(어떤 API 키가 등록되어 있는가)에 좌우됩니다. 즉 "무엇을 부를 수 있는가"를 정하는 목록과 "지금 무엇이 켜져 있는가"를 정하는 상태는 서로 다른 층위이고, IT 담당자는 이 두 층위를 각각 따로 확인해야 실제 호출 경로를 정확히 파악할 수 있습니다.

5.2 실행 범위가 막히는 지점

5.1에서 본 경로는 열려 있는 문입니다. 이 절은 그 문이 실제로 어디서, 어떤 방식으로 닫히는지를 보여줍니다. 여기서 다루는 두 가지 장치(컨테이너 격리, 파일 쓰기 안전장치)는 7장에서 여덟 계층 전체 지도의 항목으로 다시 등장하지만, 이 절의 목적은 목록에 이름을 올리는 것이 아니라 각 장치가 작동하는 방식을 보여주는 데 있습니다.

5.2.1 실행 환경을 컨테이너로 가두기

공식 보안 문서는 시스템이 갖춘 안전장치를 여덟 계층으로 정리하고 있으며, 그 가운데 네 번째가 컨테이너 격리(Container isolation)입니다[S3]. 이 계층은 도구가 실제로 실행되는 환경 자체를 컨테이너(격리된 실행 단위) 안에 가두어, 도구 호출이 호스트 시스템 전체로 곧바로 번지지 않도록 경계를 긋는 역할을 합니다.

이 경계가 의미 있는 지점은 "무엇을 부를 수 있는가"가 아니라 "어디까지 나갈 수 있는가"입니다. 5.1에서 본 것처럼 도구 호출 자체는 연동 지점을 통해 비교적 자유롭게 이뤄지지만, 그 호출이 실행되는 자리가 컨테이너 경계 안으로 한정되어 있다면 문제가 생겨도 피해 범위가 그 경계를 넘지 않습니다. IT 담당자가 실무에서 궁금해하는 것은 결국 이 질문, 즉 "에이전트가 서버 명령을 직접 실행하는가, 그 실행 범위는 어디까지 막혀 있는가"이고, 컨테이너 격리는 그 물음에 대한 구조적인 답입니다.

다만 공개된 자료는 이 계층이 존재한다는 사실과 그 이름까지만 확인해 줄 뿐, 어떤 디렉터리가 마운트되고 어떤 네트워크 경로가 열려 있는지 같은 세부 구성까지는 밝히지 않습니다. 컨테이너 격리라는 이름을 실제 배치에서 어떻게 채우는지는 도입하는 조직이 자신의 인프라 정책에 맞춰 정해야 할 몫으로 남습니다. 이 구분은 5.3에서 다시 정리합니다.

결국 이 항이 답하려는 질문은 하나로 좁혀집니다. "에이전트가 서버 명령을 직접 실행하는가"라는 물음에는 그렇다고 답해야 합니다. 도구 호출 경로는 5.1에서 본 대로 실제 명령 실행으로 이어지기 때문입니다. 그러나 "그 실행 범위를 어디까지 막을 수 있는가"라는 이어지는 물음에는, 실행 자체는 막지 않되 실행이 일어나는 자리를 컨테이너 경계 안으로 한정한다는 것이 공개 자료가 확인해 주는 답입니다. 명령을 아예 못 하게 막는 방식이 아니라, 명령이 벗어날 수 있는 반경을 좁히는 방식이라는 점이 실무에서 갖는 차이는 큽니다.

5.2.2 쓰기 가능 범위를 제한하는 장치

여덟 계층 가운데 세 번째로 명시된 항목은 파일 쓰기 안전장치(File write safety)입니다[S3]. 이 장치는 에이전트가 도구 실행 과정에서 파일을 만들거나 고칠 때, 손댈 수 있는 파일의 범위를 제한하는 역할을 합니다.

이 장치가 4장에서 다룬 무인 실행과 왜 함께 있어야 하는지는 한 문장으로 설명됩니다. 사람이 매 단계를 확인하지 않는 상태에서 도구가 파일을 쓴다면, 잘못된 경로에 잘못된 내용을 쓰는 사고가 사람이 지켜볼 때보다 훨씬 늦게 발견됩니다. 파일 쓰기 안전장치는 그 사고가 애초에 일어날 수 있는 범위 자체를 좁히는 장치이고, 무인 실행이라는 편의를 안전하게 쓰기 위한 짝입니다.

물론 이 장치도 만능은 아닙니다. 공개 문서는 이 계층의 존재와 역할만을 밝힐 뿐, 어떤 경로가 허용 목록에 들어가고 어떤 경로가 막히는지 구체적인 판정 기준까지는 공개하지 않습니다. 그래서 이 장치를 "설정 없이도 안전을 보장하는 장치"로 오해하면 안 되고, 실제 운영에서 어떤 디렉터리에 쓰기를 허용할지는 5.3.2에서 보듯 도입 조직이 함께 결정해야 하는 부분으로 남습니다.

5.2.1의 컨테이너 격리와 이 항의 파일 쓰기 안전장치는 서로 다른 축에서 실행 범위를 좁힙니다. 컨테이너 격리가 "어느 환경 안에서 실행되는가"를 가두는 장치라면, 파일 쓰기 안전장치는 "그 환경 안에서도 어떤 파일까지 건드릴 수 있는가"를 다시 한 번 좁히는 장치입니다. 두 장치가 같은 목적을 이중으로 반복하는 것이 아니라 서로 다른 층위에서 각자의 역할을 맡고 있다는 점이, 실행 범위를 하나의 장치에만 맡기지 않는 설계 의도를 보여줍니다.

5.3 제품이 대신하는 것과 도입하는 쪽 몫

지금까지 본 실행 경로와 차단 지점을 두 갈래로 나눠 정리하면 이후 판단이 쉬워집니다. 하나는 별도 설정 없이도 기본으로 작동하는 부분이고, 다른 하나는 도입하는 조직이 직접 정해야 하는 부분입니다. 이 구분은 6장과 7장에서도 같은 형식으로 반복되고, 8장의 확인 질문으로 이어집니다.

5.3.1 제품이 기본으로 막아주는 실행 범위

5.2에서 설명한 컨테이너 격리와 파일 쓰기 안전장치는 별도의 설정 작업 없이도 기본값으로 작동하는 안전장치입니다[S3]. 도구가 실행되는 환경이 컨테이너 경계 안으로 한정되어 있고, 파일을 쓸 수 있는 범위 역시 제한되어 있다는 점은 조직이 추가로 손대지 않아도 이미 갖춰진 기준선입니다.

IT 담당자에게 이 기준선이 갖는 의미는 "아무것도 하지 않아도 이 정도는 막혀 있다"는 최소 보장입니다. 도입 검토 초기 단계에서 "우리가 아무 설정도 하지 않으면 어떤 상태로 시작하는가"라는 질문에, 이 두 장치가 답이 됩니다. 다만 이 기준선은 출발점이지 종착점은 아니며, 조직의 실제 배치 환경에 맞춰 조정이 필요한 부분은 다음 항에서 정리합니다.

이 기준선을 다른 각도에서 보면, 도구 실행이라는 위험 요소를 처음부터 조직이 전부 설계해야 하는 것은 아니라는 뜻이기도 합니다. 5.1에서 본 도구 호출 경로가 아무리 다양해도, 그 호출이 실행되는 자리와 쓸 수 있는 파일 범위는 이미 좁혀진 상태로 출발합니다. 도입 조직이 처음부터 격리 경계와 쓰기 범위를 직접 설계해야 하는 상황과, 이미 그어진 경계를 검토하고 필요하면 좁히거나 넓히는 상황은 실무 부담이 다릅니다.

5.3.2 도입 조직이 직접 정해야 하는 부분

5.1.2에서 본 웹 검색·추출 백엔드는 기본값이 Firecrawl 이지만, 그 자리를 대체할 수 있는 선택지가 이미 일곱 가지 더 마련되어 있습니다. SearXNG, Brave, DuckDuckGo, Tavily, Exa, Parallel, xAI 가 그 대안이고, 도입 조직은 여덟 종 가운데 자신의 데이터 처리 방침에 맞는 백엔드를 직접 선택할 수 있습니다[S7].

구분	백엔드
기본값	Firecrawl
대안	SearXNG · Brave · DuckDuckGo · Tavily · Exa · Parallel · xAI

이 선택이 그냥 취향의 문제가 아닌 이유는 5.1.2에서 짚은 자동 감지 방식 때문입니다. 웹 백엔드 설정을 명시적으로 지정하지 않으면 사용 가능한 API 키를 기준으로 백엔드가 자동으로 바뀔 수 있으므로[S7], 조직이 원하는 백엔드와 실제로 켜져 있는 백엔드가 일치하는지를 도입 조직이 스스로 점검해야 합니다. 이 점검을 건너뛰면 검색 요청이 조직이 의도하지 않은 서비스로 새어 나가는 경로가 생길 수 있습니다.

편의 기능인 자동 감지가 동시에 확인 항목이 된다는 이 구조는 이 장에서 가장 실무적인 결론입니다. 어떤 백엔드를 쓸지, API 키를 어떻게 관리할지는 제품이 대신 정해 주지 않고 도입하는 쪽이 정책으로 명시해야 하는 몫이며, 이 선택은 8장에서 확인 질문 항목으로 다시 다룹니다.

정리하면 이 장에서 본 실행 경로는 두 겹으로 이뤄져 있습니다. 바깥쪽 겹은 어떤 도구를 부를 수 있는지를 정하는 연동 지점이고, 안쪽 겹은 그 호출이 실제로 어디까지 나갈 수 있는지를 정하는 격리 경계입니다. 바깥쪽 겹은 조직이 필요에 따라 넓히거나 새 서비스를 등록하며 계속 바뀌는 반면, 안쪽 겹은 별도 조치가 없는 한 기본값으로 유지됩니다. 이 두 겹을 각각 다른 질문으로 나눠 검토하는 습관이, 도구 실행이라는 위험 요소를 다루는 가장 실용적인 출발점입니다.

6장: 모델 선택과 데이터 처리 위치

1장에서 확정한 여덟 항목 가운데 이 장이 다루는 것은 다섯 번째, 데이터 처리 위치 공개입니다. 에이전트를 도입하겠다는 결재를 올리는 실무자에게 가장 먼저 돌아오는 질문은 대개 "우리 회사 데이터가 어디로 나가느냐"이고, 그 답은 모델 하나를 고르는 문제로 끝나지 않습니다. 에이전트 프로세스를 사내 서버에 앉혀도 모델 호출과 검색 요청, 사용 정보, 메시지 알림은 각자 다른 경로로 바깥과 통신합니다. 이 장은 그 경로를 하나씩 짚습니다.

6.1 모델 제공자를 고르는 세 갈래 길

에이전트를 어떤 모델로 움직일지는 예산 담당자와 보안 담당자가 동시에 묻는 질문입니다. 답을 세 갈래 선택지로 정리하고, 그 선택지 사이를 오갈 때 실제로 치르는 비용까지 함께 봐야 결재 문서에 빠진 항목이 남지 않습니다.

6.1.1 세 제공자 경로

모델을 어디서 불러올지는 사내에서 예산과 승인 절차가 갈리는 지점입니다. 같은 작업이라도 호출이 앤트로픽(Anthropic) 자체 API 로 가는지, 여러 모델을 중계하는 라우팅 서비스로 가는지, 아니면 이 에이전트를 만든 회사가 운영하는 별도 포털로 가는지에 따라 계약 창구와 데이터 처리 조항이 달라지기 때문입니다.

공식 문서는 모델 제공자로 세 경로를 나열합니다. 첫째는 네이티브 앤트로픽 경로로, API 키를 앤트로픽에 직접 등록해 호출이 앤트로픽 서버로만 갑니다. 둘째는 오픈라우터(OpenRouter) 경로로, 여러 모델 제공자를 한 API 로 중계하는 서드파티 라우팅 서비스를 거칩니다. 셋째는 누스 포털(Nous Portal) 경로로, 에이전트 개발사가 직접 운영하는 모델 게이트웨이입니다[S6]. 세 경로는 우열 관계가 아니라 계약 형태가 다른 선택지입니다. 사내에 이미 앤트로픽 엔터프라이즈 계약이 있다면 첫째가 자연스럽고, 여러 모델을 실험적으로 바꿔가며 쓰고 싶다면 둘째가 편하며, 별도 계약 없이 빠르게 시작하고 싶다면 셋째가 진입 장벽이 낮습니다.

세 경로 가운데 어느 것을 고르느냐가 곧 6.2 에서 다룰 데이터 처리 경로 네 갈래 중 한 갈래(모델 제공자 경로)를 그대로 결정합니다. 즉 이 항의 선택은 뒤에 나올 데이터 흐름표의 한 행을 채우는 입력값입니다. 세 경로를 표로 정리하면 다음과 같습니다.

제공자 경로	호출 창구	계약 성격
네이티브 앤트로픽	앤트로픽 API 직접 호출	앤트로픽과 직접 계약, 별도 중계자 없음
오픈라우터	오픈라우터 API 경유	서드파티 라우팅 서비스가 여러 모델 제공자를 중계
누스 포털	에이전트 개발사 게이트웨이	개발사가 운영하는 별도 모델 게이트웨이 계약

세 경로 모두 공식 문서에 나열된 선택지이며[S6], 이 표 자체가 결재 문서에 그대로 옮길 수 있는 요약입니다.

다만 공식 문서는 세 경로 각각의 지역별 가용성이나 요금 체계 차이까지는 다루지 않습니다. 어느 경로를 택해도 기능 자체는 동일하게 동작한다는 전제 위에서 나열된 선택지일 뿐, 계약 조건이나 데이터 보관 기간 같은 세

부는 각 제공자의 별도 약관을 봐야 합니다. 그래서 이 표는 선택지를 좁히는 출발점이지 계약서를 대신하지 못합니다. 실제 결재를 준비하는 담당자는 세 경로 중 후보를 이 표로 추린 뒤, 후보로 남은 제공자의 서비스 약관과 데이터 처리 조항을 별도로 확인하는 절차를 이어 가야 합니다.

6.1.2 제공자를 바꿔도 남는 것과 사라지는 것

제공자나 모델을 언제든 바꿀 수 있다는 말은 절반만 맞습니다. 전환 명령 한 줄로 모델을 바꾸는 것은 실제로 가능하지만, 그 전환에는 눈에 보이지 않는 비용이 따라붙습니다. 이 비용을 모르고 예산을 짜면 실제 청구서에서 차이가 납니다.

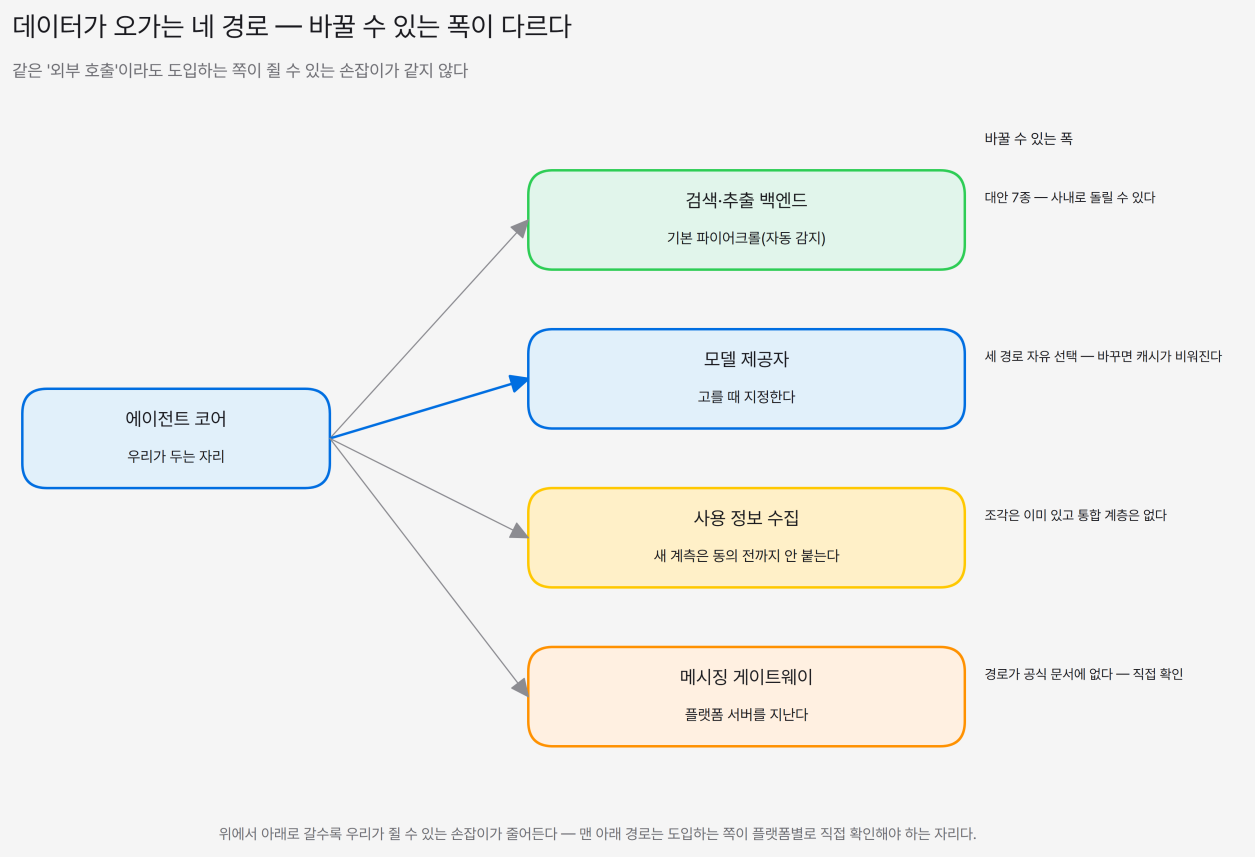
공식 문서는 모델을 전환할 때마다 프롬프트 캐시가 초기화된다고 명시합니다. 캐시 키에 모델 식별자가 포함되어 있어서, 전환 직후 첫 메시지는 캐시된 부분 없이 전체 대화 내용을 처음부터 다시 읽어 들이고, 이 첫 메시지는 전체 입력 가격으로 과금됩니다[S5]. 프롬프트 캐시는 이미 처리한 컨텍스트를 다시 계산하지 않도록 절약해 주는 장치인데, 모델을 바꾸는 순간 그 절약분이 사라지고 처음 상태로 돌아간다는 뜻입니다.

이 비용은 대화가 길수록, 그리고 전환 빈도가 잦을수록 커집니다. 짧은 세션 하나를 새로 시작할 때 모델을 한 번 바꾸는 정도라면 체감이 크지 않지만, 긴 절차 기억을 쌓아 온 세션에서 모델을 자주 오가면 매번 전체 재계산 비용이 쌓입니다. 반례로, 프로파일을 아예 새로 시작하거나 짧은 단발 질의만 던지는 용도라면 이 비용은 무시할 만한 수준입니다. 공식 문서가 이 사실을 굳이 FAQ 항목으로 남긴 이유도 여기에 있습니다. "모델은 언제든 바꿀 수 있다"는 유연성의 이면에 재계산 비용이라는 실비용이 있다는 점을, 완곡하게 돌려 말하지 않고 그대로 밝히고 있습니다[S5].

그래서 예산을 검토하는 담당자는 모델 전환을 "무료 옵션 변경"이 아니라 "전환 시점마다 한 번의 전체 재계산 비용이 드는 결정"으로 다뤄야 합니다. 여러 모델을 상시로 오가며 쓰는 운영 방식을 택할지, 아니면 프로파일이나 업무별로 모델을 고정해 캐시 이점을 유지할지는 조직이 미리 정해 둘 몫입니다.

6.2 데이터가 오가는 네 갈래 경로

에이전트를 드나드는 데이터는 한 갈래로만 흐르지 않습니다. 검색·추출 백엔드, 모델 제공자, 사용 정보 수집, 메시징 게이트웨이라는 서로 독립된 네 경로가 각자 다른 곳으로 데이터를 보내거나 받습니다. 이 네 경로를 하나씩 확인하지 않으면 "에이전트가 사내 서버에 있으니 데이터도 안 나간다"는 착각에 빠지기 쉽습니다. 에이전트 프로세스의 물리적 위치와 그 프로세스가 만드는 외부 호출은 별개의 문제입니다.



6.2.1 검색·추출 백엔드가 만드는 경로

에이전트가 웹에서 정보를 찾아오는 순간, 그 요청은 에이전트를 만든 회사가 아니라 검색·추출 기능을 실제로 처리하는 서드파티 서비스로 나갑니다. 이 서비스가 무엇이냐에 따라 검색어와 추출 대상 URL, 추출된 본문이 어느 회사의 인프라를 거치는지가 정해집니다.

기본값은 파이어크롤(Firecrawl)입니다. 별도로 백엔드를 지정하지 않으면 사용 가능한 API 키를 기준으로 자동 감지되어 파이어크롤이 요청을 처리하고, 이 기본값 외에도 서치엑스엔지(SearXNG)를 포함해 총 7종의 대안 백엔드로 바꿀 수 있습니다[S7]. 즉 검색·추출 경로는 고정된 것이 아니라 설정 한 항목으로 자체 호스팅 백엔드까지 전환 가능한 스왑형 구조입니다.

이 메커니즘 자체는 5장에서 이미 다뤘습니다. 이 장에서 같은 내용을 다시 설명하지는 않고, 데이터가 오가는 네 갈래 가운데 한 갈래로만 재배치해 표에 담습니다. 이 경로가 갖는 의미는 "검색 기능이 잘 작동하는가"가 아니라 "검색어와 검색 결과가 기본적으로 어느 회사를 거치는가"입니다. 자체 호스팅으로 전환하면 이 경로만큼은 조직의 인프라 안에 머무르게 할 수 있지만, 전환하지 않는 한 기본값인 외부 서비스로 나갑니다.

이 전환에도 한계는 있습니다. 백엔드를 자체 호스팅으로 바꾼다고 해서 검색 대상 웹사이트까지 통제할 수 있는 것은 아니며, 추출 결과가 담기는 저장소를 별도로 마련하지 않으면 자체 호스팅의 이점이 절반만 실현됩니다. 규제 산업처럼 외부 검색 자체를 아예 막아야 하는 조직이라면 백엔드 전환만으로는 부족하고, 검색·추출 기능 자체를 비활성화하는 별도 구성이 필요합니다. 결국 이 경로에서 조직이 실제로 손에 쥐는 선택지는 "외부로 보낼지, 자체 인프라로 돌릴지, 아예 끌지"라는 세 갈래이고, 그 결정은 기본값을 그대로 둔 채로는 내려지지 않습니다.

6.2.2 모델 제공자가 만드는 경로

6.1 에서 고른 제공자는 데이터가 오가는 두 번째 갈래를 그대로 채웁니다. 프롬프트 전체와 모델의 응답, 그리고 대화 맥락이 이 경로를 통해 오갑니다.

네이티브 앤트로픽을 고르면 프롬프트와 응답은 앤트로픽 서버로만 오갑니다. 오픈라우터를 고르면 오픈라우터라는 중계 서비스를 거쳐 최종 모델 제공자에게 전달되고, 누스 포털을 고르면 에이전트 개발사가 운영하는 게이트웨이를 거칩니다[S6]. 세 경로 모두 6.1.1 에서 다룬 선택지와 동일하므로 여기서 다시 설명하지 않고, 데이터 흐름표의 한 행으로만 요약합니다.

이 갈래가 중요한 이유는 단순합니다. 제공자 선택은 그 자체로 데이터 처리 위치 선택입니다. "어떤 모델을 쓸까"라는 질문과 "우리 프롬프트가 어디로 가는가"라는 질문은 사실상 같은 질문이고, 6.1 에서 이미 답한 선택이 이 갈래의 답이기도 합니다.

6.2.3 사용 정보 수집 경로

세 번째 갈래는 가장 오해하기 쉬운 지점입니다. "텔레메트리가 있는가 없는가"를 하나의 답으로 뭉뚱그리면 나중에 그 답을 뒷받침할 근거를 대지 못합니다. 정책상 원칙과 이미 존재하는 계측 조각은 서로 다른 층위의 사실이므로 나눠서 봐야 합니다.

정책상 원칙부터 보면, 개발 가이드 문서는 신규 텔레메트리나 제3자 식별자 태깅, 귀속 태그를 사용자 대상 opt-in 게이트(설정 게이트, 설치 시 안내, 토글) 없이는 병합하지 않는다는 방침을 명시하고 있습니다[S8]. 이것은 개발팀이 코드를 머지하는 기준이며, 앞으로 새로 추가되는 계측 기능은 사용자가 켜고 끌 수 있는 장치 없이는 들어가지 않는다는 약속입니다.

그런데 이 정책이 "현재 배포판에 사용 정보 수집이 전혀 없다"는 뜻은 아닙니다. 공개 이슈 트래커에는 토큰 회계, 예상 비용, 세션 길이, `/insights` 같은 부분적 사용 정보 조각이 이미 흩어져 존재하며, 이를 하나로 묶는 통합 텔레메트리 계층은 없다는 설명이 올라와 있습니다[S9]. 다시 말해 "통합된 감시 체계는 없다"는 사실과 "아무 계측도 없다"는 주장은 다른 이야기입니다. 이미 존재하는 조각들이 로컬에만 머무르는지, 일부라도 외부로 나가는지는 이슈 본문에서도 확정되지 않았습니다.

두 사실을 합쳐 "수집 안 함"이라고 단정하면 나중에 감사나 실사 과정에서 그 단정을 뒷받침할 근거를 대지 못하는 상황에 놓입니다. 정책은 앞으로 추가될 신규 기능에 대한 약속이고, 기존 계측 조각의 범위와 목적지는 별개로 확인해야 할 현황입니다. 이 두 층위를 나눠 쓰는 것이 이 장에서 가장 조심해야 할 서술입니다.

6.2.4 메시징 게이트웨이가 만드는 경로

네 번째 갈래는 메시징 플랫폼 연동입니다. 텔레그램 같은 채널로 에이전트와 대화하거나 알림을 받는 구성을 쓰면, 메시지는 그 플랫폼의 서버를 거쳐 에이전트로 전달됩니다.

공식 문서는 이 기능을 "메시징 플랫폼 연동"이라고만 언급할 뿐, 채널별로 메시지가 정확히 어느 서버를 거쳐 에이전트에 도달하는지 세부 경로는 밝히지 않습니다[S6]. 이것은 확인에 실패했다는 뜻이 아니라, 공식 문서 자체에 그 수준의 세부가 담겨 있지 않다는 사실입니다.

세부가 없다는 사실 자체가 도입을 검토하는 조직에는 확인 대상이 됩니다. 메시징 연동을 쓰기로 결정했다면, 어느 플랫폼을 통해 메시지가 오가는지, 그 플랫폼의 서버가 어느 지역에 있는지, 메시지 내용이 플랫폼 쪽에 얼마나 오래 남는지를 도입하는 조직이 해당 플랫폼의 약관과 공식 문서를 별도로 확인해야 합니다. 이 항목은 8장의 확인 질문으로 그대로 이어집니다.

네 갈래를 한 번에 비교할 수 있도록 정리하면 다음과 같습니다. 표의 "전환 가능" 열은 기본값을 그대로 두는 대신 다른 경로로 바꿀 수 있는지를 나타내고, "성격" 열은 그 경로가 확정된 사실인지 정책인지 확인이 필요한 미상인지를 구분합니다.

경로	기본값	전환 가능	성격
검색·추출 백엔드	파이어크롤(자동 감지)[S7]	서치엑스엔지 등 7종 대안으로 전환 가능[S7]	확인된 기본값, 자체 호스팅 전환 가능
모델 제공자	선택 시점에 지정(네이티브 앤트로픽/오픈라우터/누스 포털)[S6]	세 경로 중 자유 선택, 단 전환 시 프롬프트 캐시 초기화[S5]	확인된 선택지, 전환에 실 비용
사용 정보 수집	신규 텔레메트리는 opt-in 게이트 전까지 미병합(정책)[S8]	이미 토큰 회계·비용 추정·세션 길이 등 조각 존재, 통합 계층 없음[S9]	정책과 현황이 분리된 사실, 목적지 미확인
메시징 게이트웨이	연동 시 해당 플랫폼 서버 경유(세부 미명시)[S6]	채널 선택에 따라 달라지나 공식 문서에 경로 명시 없음[S6]	확인 필요, 도입 조직이 플랫폼별로 별도 확인

이 표가 이 장에서 가장 값나가는 한 쪽입니다. 자체 호스팅을 근거로 결재를 받아야 하는 담당자에게는, 에이전트 프로세스의 물리적 위치와 이 네 갈래 각각의 목적지가 서로 다른 질문이라는 점을 보여 주는 것이 표 한 장의 역할이기 때문입니다. "자체 호스팅이면 데이터가 안 나간다"는 문장은 자동으로 참이 되지 않습니다. 에이전트 프로세스를 사내 서버나 사내 클라우드에 올려도, 검색·추출 백엔드를 기본값 그대로 두면 그 요청은 외부 서비스로 나가고, 모델 제공자를 외부 API 로 두면 프롬프트도 외부로 나갑니다. 자체 호스팅이 통제할 수 있는 것은 에이전트 코어가 도는 위치이지, 코어가 만드는 네 갈래 호출까지 전부를 자동으로 사내에 가두는 것은 아닙니다.

6.3 제품이 대신하는 것과 도입하는 쪽 몫

앞선 6.2 에서 확인한 네 갈래는 제품이 기본값으로 이미 처리해 주는 부분과, 도입하는 조직이 스스로 확인하고 결정해야 하는 부분으로 나뉩니다. 이 구분은 5.3 과 같은 형식이며, 8장의 확인 질문으로 그대로 이어집니다.

6.3.1 제품이 기본값으로 처리하는 부분

별도 설정 없이 이미 적용되어 있는 값부터 정리합니다. 검색·추출 백엔드는 파이어크롤이 기본값으로 자동 감지되어 동작하므로, 별도 구성 없이 바로 쓸 수 있습니다[S7]. 사용 정보 수집 쪽에서는 신규 텔레메트리나 제3자 식별자 태깅을 opt-in 게이트 없이 병합하지 않는다는 정책이 이미 개발 기준으로 자리 잡고 있습니다[S8].

이 두 가지는 6.2 에서 이미 확인한 사실을 넘어서는 새로운 정보가 아니라, 그 확인 결과를 "이미 되어 있는 것"으로 다시 묶은 것입니다. 도입 조직 입장에서는 검색 백엔드를 반드시 바꿔야만 쓸 수 있는 구조가 아니고, 신규 계측 기능이 조직 동의 없이 슬그머니 들어오는 구조도 아니라는 뜻입니다. 별도 승인 절차 없이 시작할 수 있는 지점이 여기까지입니다.

6.3.2 도입 조직이 직접 확인해야 하는 부분

반대로 공식 문서만으로는 다 채워지지 않는 부분이 있고, 이 부분은 도입 조직이 직접 확인해야 합니다.

첫째, 제공자 전환 비용입니다. 모델을 바꿀 때마다 프롬프트 캐시가 초기화되어 전환 직후 첫 메시지가 전체 입력 가격으로 처리된다는 사실[S5]을 조직의 운영 방식에 대입해, 얼마나 자주 모델을 오갈 계획인지 미리 따져 봐야 합니다. 우리 조직은 모델을 얼마나 자주 바꿀 예정이고, 그 전환 빈도가 예산에 어느 정도 영향을 주는지 확인할 질문입니다.

둘째, 이미 존재하는 계측 조각의 실제 범위입니다. 토큰 회계와 비용 추정, 세션 길이 같은 조각이 어디까지 기록되고 그 데이터가 로컬에만 남는지 확인해야 합니다[S9]. 우리가 쓸 배포 환경에서 이 조각들이 어디에 저장되고 누가 접근할 수 있는지 확인할 질문입니다.

셋째, 메시징 경로의 세부입니다. 공식 문서가 채널별 경로를 명시하지 않으므로[S6], 실제로 도입할 메시징 플랫폼의 약관과 데이터 보관 정책을 별도로 확인해야 합니다. 우리가 연동할 플랫폼이 메시지를 어디에 얼마나 보관하는지 확인할 질문입니다.

이 세 질문은 8.2.3 의 확인 목록으로 그대로 넘어갑니다. 이 장이 정리한 네 갈래 경로와 기본값·전환 가능 여부를 바탕으로, 8장에서는 도입 전에 실제로 던져야 할 질문 형태로 다시 정리합니다.

7장: 권한 분리와 안전장치

이 장은 1장이 정한 여덟 항목 가운데 ⑥ 권한 분리와 ⑦ 위험 명령 승인 두 가지를 다룹니다. 이 두 항목은 공식 보안 문서가 밝히는 여덟 계층 안전장치 체계 안에 자리 잡고 있어서, 계층 전체를 먼저 펼쳐 놓아야 두 항목의 위치가 분명해집니다. 다만 미리 밝혀 둘 사실이 하나 있습니다. 여덟 계층 가운데 어느 하나도 실행이 끝난 뒤 그 결과가 실제로 맞는지 확인하지 않습니다. 즉 1장이 정한 여덟 항목 중 ⑧ 실행 결과 검증에 해당하는 계층은 이 안전장치 체계에 없습니다. 이 장은 그 사실을 감추지 않고 마지막 절에서 정면으로 다룹니다.

7.1 여덟 계층 전체 지도와 신원·명령 통제

7.1.1 안전장치 여덟 계층 한눈에 보기

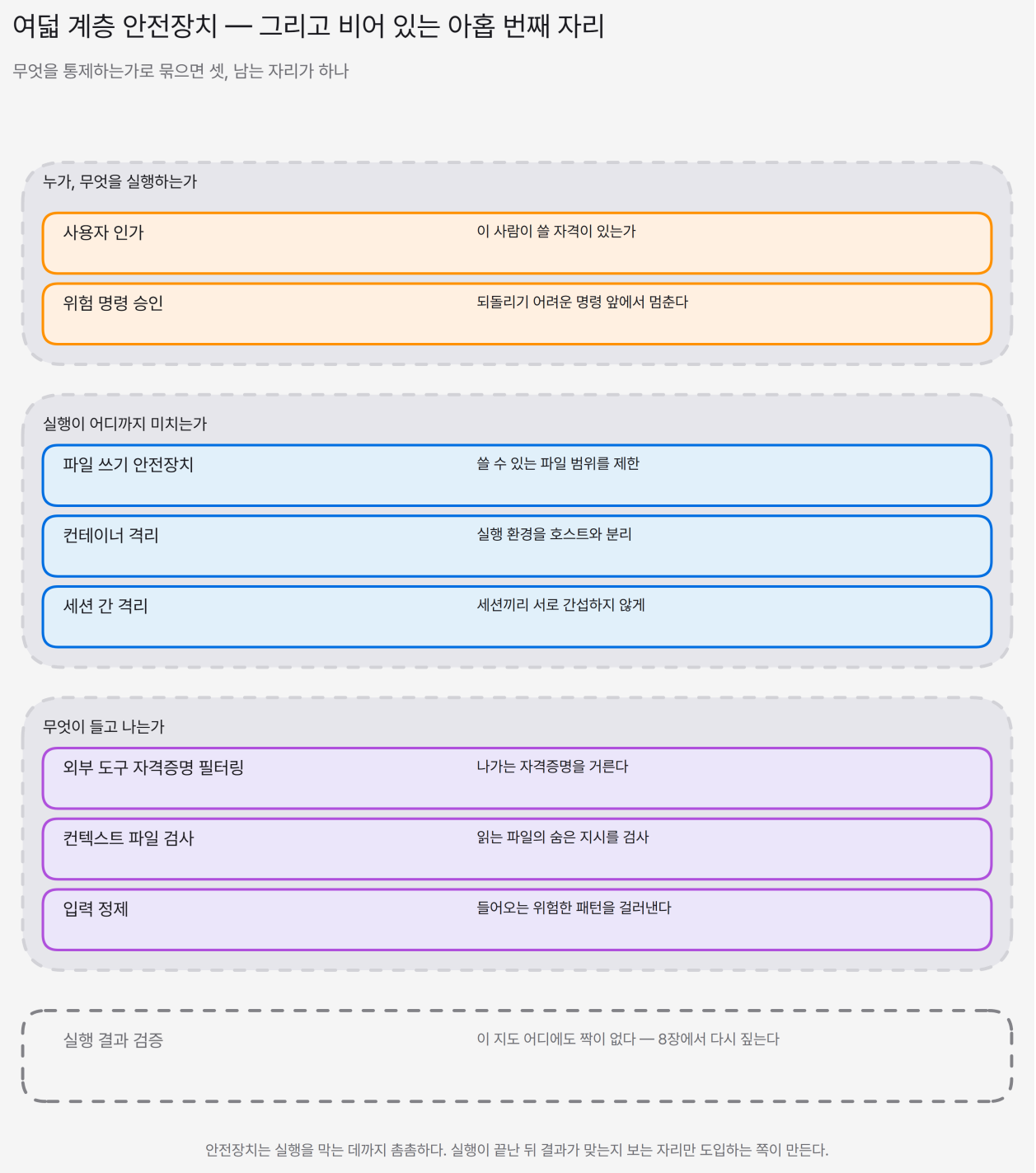
Hermes Agent 공식 보안 문서는 안전장치를 여덟 계층으로 나눠 설명합니다[S3]. 계층은 사용자 인가, 위험 명령 승인, 파일 쓰기 안전장치, 컨테이너 격리, 외부 도구 자격증명 필터링, 컨텍스트 파일 검사, 세션 간 격리, 입력 정제 순으로 이어집니다[S3]. 여덟 개라는 숫자가 낯설게 느껴질 수 있는데, 이는 계층 하나하나가 서로 다른 시점에서 서로 다른 대상을 검사하기 때문입니다. 어떤 계층은 사람이 누구인지 확인하고, 어떤 계층은 명령이 실행되기 직전에 개입하며, 어떤 계층은 파일이나 자격증명처럼 구체적인 대상을 들여다봅니다.

이 여덟 계층을 각각 문단으로 풀어 설명하기 전에, 계층 번호와 역할을 표로 먼저 정리해 두면 전체 그림을 잡는 데 5분이면 충분합니다. 이후 절들은 이 표의 순서를 그대로 따라가며 계층을 하나씩 좁혀서 다룹니다. 여덟 줄을 한 번에 훑어보면, 뒤에 나오는 절 하나하나가 이미 이 표 안 어느 자리를 설명하는지 미리 알 수 있어서 낯선 용어가 나와도 길을 잃지 않습니다.

계층	계층명	한 줄 역할
1	사용자 인가	이 요청을 보낸 사람이 에이전트를 쓸 자격이 있는지 확인
2	위험 명령 승인	삭제·전송처럼 되돌리기 어려운 명령을 실행 직전에 멈추고 승인을 요구
3	파일 쓰기 안전장치	에이전트가 쓸 수 있는 파일 범위를 제한
4	컨테이너 격리	실행 환경을 호스트 시스템과 분리
5	외부 도구 자격증명 필터링	외부 연동 도구로 나가는 자격증명이 새지 않도록 거름
6	컨텍스트 파일 검사	에이전트가 읽어들이는 파일에 숨은 지시가 섞였는지 검사
7	세션 간 격리	서로 다른 세션의 실행이 서로 간섭하지 않도록 분리
8	입력 정제	채널로 들어오는 입력에서 위험한 문자열·패턴을 걸러냄

표를 보면 여덟 계층 중 앞의 두 계층은 누가, 어떤 명령을 실행하는지를 통제하고, 가운데 세 계층은 실행이 벗어날 수 있는 범위를 가두며, 나머지 세 계층은 들고 나는 데이터를 검사합니다. 이 장은 7.1에서 앞의 두 계층을, 7.2에서 가운데 세 계층을, 7.3에서 나머지 세 계층과 빠진 계층을 순서대로 다룹니다.

이 여덟 계층은 1장에서 정리한 네 가지 실행 단계 — 누가 시작하는가, 무엇을 만지는가, 어디로 나가는가, 누가 확인하는가 — 와도 그대로 맞물립니다. 앞의 두 계층은 "누가 시작하는가"에 해당하고, 가운데 세 계층은 "무엇을 만지는가"에, 나머지 세 계층은 "어디로 나가는가"에 가깝습니다. 다만 "누가 확인하는가"라는 네 번째 단계는 절차 기억 쪽에만 짝이 있고, 실행 결과 검증 쪽 짝은 이 여덟 계층 지도 어디에도 남아 있지 않습니다. 이 빈자리는 7.3에서 다시 짚습니다.



7.1.2 사용자 인가와 위험 명령 승인

4장은 채널로 들어온 요청이 사람의 추가 승인 없이 끝까지 처리되는 무인 실행을 다뤘습니다. 이 서술과 "위험한 명령은 승인을 거친다"는 이 항의 내용이 서로 부딪히는 것처럼 보일 수 있는데, 실제로는 두 계층이 서로 다른 범위에서 작동하기 때문에 공존합니다. 사용자 인가 계층은 애초에 이 요청을 보낸 사람이 에이전트를 쓸 자격이 있는지를 확인하는 문턱이고, 위험 명령 승인 계층은 그 문턱을 통과한 요청 안에서도 되돌리기 어려운 특정 명령이 실행되기 직전에만 별도로 개입하는 좁은 통로입니다[S3].

이 두 계층이 실제로 어떻게 뚫릴 수 있는지는 2026년 7월에 보고된 사례에서 드러납니다. 태국 재무부 네트워크에 침투한 공격자는 Hermes Agent를 침투 경로가 아니라 침투 이후의 자동화 도구로 사용했습니다. 초기 침투 경로가 무엇이었는지는 보도에서도 확인되지 않았지만, 공격자는 이미 내부에 들어와 있었고 그 뒤에 에이전트를 무인으로 돌렸습니다[S11]. 이때 공격자는 소프트웨어에 실제로 있는 이름인 YOLO 모드 — 승인 없이 위험한 명령까지 그대로 실행하게 하는 설정 — 를 켜서, 위험 명령 승인을 요구하는 기본 설정을 껐습니다[S11]. 보도가 전하는 사실관계를 그대로 따라가면 이 사건은 제품 결함으로 보기 어렵습니다. 침투 지점은 에이전트가 아니라 이미 뚫려 있던 네트워크였고, 승인 설정을 끈 것은 공격자의 선택이었기 때문입니다[S11].

같은 조사에서 별도로 확인된 것은 사용자 인가 계층이 아예 켜지지 않은 채 인터넷에 노출된 대시보드의 규모입니다. 보안 조사기관이 노출 지문을 이용해 인터넷을 훑는 자체 스캔 데이터베이스를 조회한 결과, 이 계층을 인증 없이 외부 바인딩으로 띄운 패널이 한 달 동안 약 5,900건 잡혔고, 별도로 인증 없이 결과 파일을 그대로 내주는 노출 디렉터리 색인도 575건 검색되었습니다[S11]. 이 숫자는 침해 건수가 아니라 노출 면의 크기를 말합니다. 실제로 뚫린 사건의 수가 아니라, 인증 없이 열려 있어서 조사기관의 스캔에 잡힌 인스턴스의 수라는 뜻입니다. 이 노출이 이어진 배경에는 인증 없이 대시보드를 외부에 띄울 수 있게 해 주던 우회 설정이 있었고, 2026년 6월의 보안 강화 조치 이후 개발사는 이 우회 설정을 사실상 폐기하고 외부 바인딩에는 인증 게이트가 자동으로 개입하도록 바꿨습니다[S13].

이 두 계층에는 사고 사례와는 별개로 구조적인 한계도 있습니다. 공개된 이슈 트래커에는 메신저 게이트웨이로 들어오는 사용자를 인가하는 방식이 전부 아니면 전무라는 지적이 올라와 있습니다. 인가된 사용자는 모든 슬래시 명령과 모든 도구, 전체 터미널 접근 권한을 함께 받고, 위험 명령 승인을 요구하는 설정도 관리자와 일반 사용자를 가리지 않고 똑같이 적용됩니다[S12]. 역할별로 권한 단계를 나누는 기능은 이 이슈가 올라온 시점을 기준으로 아직 검토 대기 상태입니다[S12].

이 한계를 제품 결함으로 단정할 필요는 없습니다. 권한을 잘게 나누는 기능이 아직 없다는 사실은 현재 시점의 설계 범위이지, 숨겨진 결함이 아니기 때문입니다. 다만 여러 사용자가 한 인스턴스를 나눠 쓰는 조직이라면, 도입 전에 "관리자와 일반 사용자를 실제로 구분해 운영할 수 있는가"를 직접 확인해야 한다는 뜻이기도 합니다. 인가만 받으면 누구나 같은 권한을 갖는 구조에서는, 인가 대상을 어디까지 좁힐지가 사실상 유일한 통제 지점이 됩니다.

두 사건과 이 구조적 한계가 함께 말하는 것은 하나입니다. 사고의 원인은 모델의 판단력이 아니라 노출 면과 승인 게이트, 그리고 그 게이트를 얼마나 세밀하게 나눌 수 있는가라는 설정의 문제였습니다. 세 가지 모두 기본값이 아니라 운영자가 조정할 수 있는 지점으로 존재하므로, 도입하는 조직이 확인할 몫은 "이 기능이 있는가"가 아니라 "우리 환경에서 이 설정이 기본값대로 켜져 있고, 우리 조직 구조에 맞게 세분화할 수 있는가"입니다.

7.2 실행 경계를 지키는 계층

7.1이 누가 시작할 수 있는지를 다뤘다면, 이 절은 시작된 실행이 어디까지 번질 수 있는지를 가두는 계층을 다룹니다. 컨테이너 격리·세션 간 격리·파일 쓰기 안전장치 세 계층이 여기 속합니다. 이 중 컨테이너 격리와 파일 쓰기 안전장치는 5장에서 메커니즘 수준으로 이미 다뤘으므로 이 절은 여덟 계층 지도 안에서 그 위치만 확인하고, 세션 간 격리는 이 장에서 처음 자리를 짚습니다.

7.2.1 컨테이너 격리와 세션 간 격리

여덟 계층 지도에서 컨테이너 격리는 4번째 자리를, 세션 간 격리는 7번째 자리를 차지합니다[S3]. 두 계층 모두 실행 환경 자체를 가두는 역할이라는 공통점이 있지만, 가두는 대상은 다릅니다. 컨테이너 격리는 에이전트 프로세스 하나가 호스트 시스템 전체로 번지지 않도록 막고, 세션 간 격리는 같은 제품을 쓰는 여러 세션이 서로의 실행에 끼어들지 못하도록 막습니다.

이 구분은 여러 사용자나 여러 업무가 한 제품을 나눠 쓰는 조직에서 특히 중요해집니다. 컨테이너 격리만 있고 세션 간 격리가 없다면, 같은 컨테이너 안에서 서로 다른 두 세션의 작업이 뒤섞일 위험이 남습니다. 공식 보안 문서가 이 두 계층을 별도 항목으로 나눠 명시한 것은[S3], 하나가 다른 하나를 대신하지 않는다는 점을 인정한 결과로 읽힙니다.

이 차이가 두드러지는 시점은 배포 규모가 커질 때입니다. 단일 사용자가 하나의 세션만 여는 환경에서는 컨테이너 격리와 세션 간 격리가 사실상 같은 경계처럼 느껴집니다. 반면 여러 부서가 하나의 인스턴스를 나눠 쓰는 환경에서는 컨테이너는 하나이더라도 세션은 여러 개로 늘어나므로, 격리의 실제 단위가 컨테이너인지 세션인지에 따라 문제가 번지는 범위가 달라집니다.

이 장에서 두 계층의 내부 동작 방식을 다시 풀어 쓰지는 않습니다. 컨테이너 격리가 실제로 무엇을 어떻게 가두는지는 이미 앞선 장에서 메커니즘 수준으로 다뤘고, 세션 간 격리는 공식 문서가 계층의 존재와 역할만 밝힐 뿐 세부 동작까지는 공개하지 않습니다. 여기서는 이 두 계층이 여덟 계층 지도 안에서 각각 어느 자리에 있는지만 확인하면 충분합니다. 의사결정권자가 기억해 둘 것은 순서 자체보다, 실행 환경과 세션이라는 두 가지가 서로 다른 계층에서 따로 관리된다는 사실입니다.

7.2.2 파일 쓰기 안전장치

여덟 계층 지도에서 파일 쓰기 안전장치는 3번째 자리에 있습니다[S3]. 이 계층은 에이전트가 손댈 수 있는 파일 범위를 제한하는 역할을 맡습니다.

앞선 장에서 이 계층이 무인 실행과 맞물려 어떻게 작동하는지 이미 구체적으로 다뤘으므로, 여기서는 같은 설명을 되풀이하지 않습니다.

다만 파일 쓰기 안전장치와 위험 명령 승인을 같은 장치로 혼동하지 않도록 다시 짚어 둘 만합니다. 파일 쓰기 범위를 제한하는 것과, 파일을 지우거나 옮기는 명령 자체를 승인 대상으로 묶는 것은 서로 다른 통제입니다. 전자는 애초에 손댈 수 있는 대상의 범위를 좁히고, 후자는 그 범위 안에서도 되돌리기 어려운 행위 앞에서 실행을 멈춰 세웁니다.

이 장에서 남기는 결론은 하나입니다. 파일 쓰기 안전장치는 컨테이너 격리·세션 간 격리와 나란히 실행 범위를 가두는 계층군에 속하며, 다음 절에서 다루는 데이터를 검사하는 계층군과는 성격이 다릅니다.

7.3 데이터·입력 방어와 빠진 계층

여덟 계층 지도에서 남은 세 계층은 데이터가 드나드는 경계를 검사합니다. 이 절을 마지막으로 여덟 계층 전체를 다 훑게 되며, 그다음에는 이 지도 어디에도 없는 자리 하나를 정면으로 짚습니다.

7.3.1 자격증명 필터링·컨텍스트 파일 스캔·입력 정제

여덟 계층 지도에서 남은 세 계층 — 외부 도구 자격증명 필터링, 컨텍스트 파일 검사, 입력 정제 — 은 각각 5번, 6번, 8번 자리를 차지합니다[S3]. 셋을 따로따로 이해할 필요는 없습니다. 세 계층 모두 에이전트로 들어오거나 나가는 데이터를 검사한다는 하나의 틀로 묶을 수 있기 때문입니다.

자격증명 필터링은 에이전트가 외부 연동 도구를 호출할 때 API 키나 토큰 같은 자격증명이 응답이나 로그로 새어 나가지 않도록 거릅니다. 컨텍스트 파일 검사는 에이전트가 작업 중 읽어들이는 파일 안에 사용자가 의도하지 않은 지시문이 숨어 있는지를 확인합니다. 입력 정제는 채널로 들어오는 메시지 자체에서 위험한 패턴을 걸러내는 가장 앞단의 계층입니다[S3]. 세 계층의 검사 시점은 다르지만, 무엇이 들어오고 무엇이 나가는가를 확인한다는 목적은 같습니다.

이 세 계층이 중요한 이유는 앞서 1장에서 정리한 OWASP 위험 범주 중 공급망과 임의 코드 실행이 바로 이 지점, 즉 데이터가 드나드는 경계에서 발생하기 때문입니다. 5장에서 다룬 웹 검색·추출 백엔드처럼 외부 연동이 늘어날수록, 그 연동을 타고 오가는 자격증명과 응답 내용을 검사하는 계층의 역할도 함께 커집니다.

의사결정권자가 확인할 지점은 세 계층이 각각 무엇을 걸러내는지 외우는 것이 아니라, 자격증명·파일·메시지라는 세 가지 데이터 통로 모두에 검사 지점이 있는지를 자기 조직의 위협 모델과 대조해 보는 일입니다.

7.3.2 여덟 번째 계층, 검증의 부재

지금까지 짚은 일곱 계층을 다시 살펴보면 공통점이 하나 있습니다. 사용자 인가부터 입력 정제까지, 일곱 계층은 모두 누가, 무엇을, 어디까지 할 수 있는가를 통제합니다[S3]. 반면 실행이 끝난 뒤 그 결과가 실제로 맞았는가를 확인하는 계층은 이 여덟 계층 목록 어디에도 없습니다.

이 공백은 3장에서 이미 예고된 것입니다. 절차가 잘못 연결된 채로 스킬 문서에 굳어져도, 에이전트는 멈추지 않고 그럴듯하게 다듬어진 결과를 계속 내놓을 수 있다고 짚은 바 있습니다. 위험 명령 승인 게이트는 이런 절차가 되돌리기 어려운 명령을 실행하려는 순간에는 개입하지만[S3], 절차를 따라 완성된 산출물 자체가 사실과 맞는지는 확인하지 않습니다. 명령이 안전하게 실행되었다는 것과 결과가 옳다는 것은 다른 질문입니다.

이 차이를 좀 더 구체적으로 보면, 위험 명령 승인 게이트가 걸리는 대상은 삭제나 전송처럼 정해진 패턴에 맞는 명령입니다. 반면 보고서 한 편을 요약하거나 데이터를 잘못 집계하는 것처럼 명령 자체는 안전하지만 내용이 틀린 산출물은 이 게이트가 다루는 범위 바깥에 있습니다. 여덟 계층 중 어느 것도 사람이 결과물을 마지막에 읽어 보는 절차를 대신하지 않습니다.

이 공백을 제품의 결함으로 읽을 필요는 없습니다. 여덟 계층은 애초에 누가 실행을 시작하고 어디까지 손댈 수 있는가라는 통제를 설계 범위로 삼았고, 그 범위 안에서는 각 계층이 맡은 역할을 하고 있습니다. 결과 검증은 처음부터 이 통제 체계가 다루기로 한 문제가 아니었을 뿐입니다.

그래서 이 공백은 이 백서에서 가장 정직하게 남겨야 할 정보이기도 합니다. 제품이 대신해 주는 몫은 여덟 계층 중 일곱, 즉 누가 시작할 수 있는지와 어디까지 손댈 수 있는지를 통제하는 부분입니다. 도입하는 조직이 직접 채워야 할 몫은 나머지 하나, 즉 실행이 끝난 뒤 결과가 맞는지 확인하는 절차입니다. 그 설계는 8장에서 이어받아 다룹니다.

8장: 도입 판단

8.1 우리 조직 규모에 맞는 구성인가

8.1.1 설치 전제 조건을 우리 인프라가 충족하는가

Hermes Agent 도입을 검토할 때 가장 먼저 확인할 것은 화려한 기능표가 아니라 설치 조건입니다. 공식 문서가 명시하는 사전 요구사항은 세 가지뿐입니다[S2]. 이 세 가지가 우리 인프라에 이미 있는지 확인하는 일이 도입 판단의 첫 단추입니다.

- Python 3.11 이상
- Node.js
- ripgrep(파일 검색 유틸리티)

이 목록이 짧다는 것 자체가 눈에 띄는 사실입니다. CPU 코어 수, 메모리 용량, 디스크 여유 공간 같은 정량 스펙은 공식 문서 어디에도 나오지 않습니다[S2]. README 는 "5달러짜리 VPS(가상 사설 서버)부터 GPU 클러스터, 서버리스 인프라까지 돌아간다"는 표현으로 지원 범위를 밝힐 뿐, 각 환경에 필요한 최소 사양을 숫자로 못박지는 않습니다[S2].

이것을 벤더의 게으름으로 읽을 필요는 없습니다. 에이전트가 실제로 쓰는 자원은 실행하는 스킬의 종류, 동시에 여는 세션 수, 호출하는 모델 제공자에 따라 크게 갈리기 때문에 하나의 숫자로 정리하기 어려운 영역이라고 보는 편이 더 정확합니다. 다만 그 반례를 인정한다고 해서 확인 부담이 사라지지는 않습니다. 정량 기준이 없다는 사실 자체가 확인 항목입니다. 문서를 더 뒤진다고 나오지 않고, 직접 설치해서 우리 워크로드로 며칠 돌려보는 것 외에는 확정할 방법이 없습니다.

그러므로 도입 전 점검표에는 "사양을 문서에서 찾는다"가 아니라 "테스트 환경에 설치해 우리 업무 규모로 실측한다"가 올라가야 합니다. 이 판단을 건너뛰고 바로 도입 규모를 정하면, 운영 단계에서 자원 부족을 뒤늦게 발견하는 위험을 조직이 그대로 떠안게 됩니다. 실측을 건너뛰고 인프라 규모부터 결정하면, 나중에 서버를 다시 산정하거나 계약을 재협상해야 하는 상황으로 이어질 수 있습니다. 그러니 세 가지 사전 요구사항을 갖춘 시험 환경 하나를 먼저 마련하고, 실제로 검토 중인 반복 업무 한두 가지를 그 위에서 돌려 본 뒤에 규모를 결정하는 순서가 안전합니다.

8.1.2 여덟 항목 중 기본 제공과 직접 구성을 우리가 구분했는가

이 백서 1장이 정리한 필수 기능 여덟 항목은 이론적 체크리스트가 아니라 실제로 대볼 수 있는 표입니다. 그 값은 각 항목이 "제품이 기본으로 주는 것"인지 "도입하는 조직이 채워야 하는 것"인지를 우리 조직 스스로 표시해 볼 때 나옵니다.

절차 기억(스킬 시스템)은 에이전트가 실행 뒤 돌아가는 자기개선 리뷰에서 스스로 스킬을 작성하는 방식으로 기본 제공됩니다[S4]. 채널 통합과 무인 실행, 실행 격리 역시 문서화된 기본 동작입니다[S3]. 데이터 처리 위치는 검색·추출 백엔드 선택지가 문서에 명시되어 있다는 점에서 공개 수준은 기본 제공에 가깝지만, 실제로 어느 백엔드를 쓸지는 조직이 골라야 합니다[S7]. 권한 분리와 위험 명령 승인은 보안 문서가 명시한 여덟 계층 안에 들어 있는 기본 기능입니다[S3]. 실행 결과 검증만은 다릅니다. 이 항목은 제품이 절차를 정해 두지 않은 채 남겨 둔 자리입니다.

아래 표에 우리 조직 이름으로 직접 체크해 보십시오. 마지막 열이 비어 있는 항목은 이번 도입 프로젝트의 실제 작업 목록에 올라가야 합니다.

항목	기본 제공(제품 몫)	직접 구성 필요(도입 몫)	우리 조직 확인 여부
① 절차 기억	O	—	☐
② 채널 통합	O	—	☐
③ 무인 실행	O	실행 범위 조정	☐
④ 실행 격리	O	—	☐
⑤ 데이터 처리 위치 공개	O(선택지 공개)	백엔드 선택	☐
⑥ 권한 분리	O	—	☐
⑦ 위험 명령 승인	O(절차 존재)	승인 담당자 지정	☐
⑧ 실행 결과 검증	—	검증 절차 전부	☐

이 표가 보여주는 것은 실망스러운 결핍이 아니라 작업 범위입니다. 일곱 항목은 설치만 하면 기본으로 따라오고, 여덟 번째 항목만 조직이 처음부터 설계해야 한다는 뜻이기 때문입니다. 다만 "범위 조정"이나 "담당자 지정"처럼 표에 짧게 적힌 말들도 실제로는 별도의 결정을 요구합니다. 이 결정들을 8.2 절에서 하나씩 물음으로 풀어보겠습니다.

8.2 도입 전에 확인할 물음들

8.2.1 무인 실행 범위를 우리 기준으로 좁힐 수 있는가

Hermes Agent 의 무인 실행은 승인 없이 여러 단계를 이어서 처리하는 능력에서 나옵니다. 이 능력이 매력적인 이유는 반복 업무를 사람이 매번 확인하지 않아도 되게 만들어 주기 때문입니다. 그런데 이 편의는 뒤집으면 그대로 위험의 크기이기도 합니다. 승인 없이 도는 범위가 넓을수록 사람이 개입할 틈도 좁아집니다.

공식 보안 문서는 위험한 명령에 대해서는 승인 단계를 두고 있다고 밝힙니다[S3]. 하지만 "위험하다"의 경계선을 어디에 그을지, 무인 실행이 허용되는 작업의 범위를 얼마나 좁힐지는 제품이 아니라 도입하는 조직이 정해야 하는 값입니다. 문서는 이 범위를 조정하는 설정 지점이 있다는 사실은 알려주지만, 우리 조직에 맞는 값이 무엇인지까지 알려주지는 않습니다.

그러므로 이 항은 결론이 아니라 물음으로 남습니다. 우리 조직의 반복 업무 중 무인 실행을 맡겨도 되는 범위는 어디까지인가, 그 범위를 설정으로 실제로 좁힐 수 있는지 우리 환경에서 확인했는가. 이 두 물음에 아직 답하지 못했다면, 도입은 그 답을 찾은 뒤로 미루는 편이 안전합니다.

이 범위는 업무마다 다르게 그어질 수밖에 없습니다. 정형화된 조회·요약 업무는 무인 실행 범위를 넓게 두어도 위험이 크지 않지만, 외부로 나가는 메시지 발송이나 시스템 설정을 바꾸는 명령이 섞인 업무라면 범위를 좁게 두고 승인 단계를 더 촘촘히 거치게 하는 편이 안전합니다. 결국 이 조정은 제품이 대신 해 주지 않는, 업무 성격을 가장 잘 아는 조직 스스로의 판단입니다.

8.2.2 위험한 명령의 승인은 누가 어떤 절차로 하는가

앞 항에서 남긴 승인 지점은 제품이 만들어 놓은 자리일 뿐, 그 자리에 앉을 사람까지 제품이 정해 주지는 않습니다. 보안 문서가 밝히는 여덟 계층 가운데 위험 명령 승인은 두 번째 계층으로 명시되어 있습니다[S3]. 이 계층이 실제로 작동하려면 승인 요청이 뜨는 순간 누가 그것을 검토하고 판단할지가 조직 안에 정해져 있어야 합니다.

여기서 흔히 놓치는 지점이 있습니다. 승인 절차가 시스템에 있다는 사실과, 그 절차를 담당할 사람이 조직에 지정되어 있다는 사실은 서로 다른 이야기입니다. 전자는 제품이 갖추고 있지만, 후자는 온전히 도입하는 조직의 몫입니다. 승인 요청이 떴을 때 담당자가 정해져 있지 않으면, 알림은 계속 쌓이고 아무도 판단하지 않는 상태로 방치되기 쉽습니다. 이렇게 되면 승인 계층은 이름만 존재하는 빈 절차가 됩니다.

그러므로 도입 전에 답해야 할 물음은 이렇습니다. 위험 명령 승인 알림이 오면 누구에게 가는가, 그 사람이 부재중일 때 대신 판단할 사람은 정해져 있는가, 판단 기준은 문서로 남아 있는가. 이 물음들에 이름과 절차로 답할 수 있어야 승인 계층이 실제로 작동합니다.

8.2.3 검색·추출 백엔드와 사용 정보 수집 범위를 우리 환경에서 확인했는가

이 항은 두 개의 서로 다른 사실을 섞지 않고 각각 확인하라는 물음입니다. 첫째는 검색·추출 백엔드입니다. 기본값은 Firecrawl 이고, 웹 백엔드 설정을 지정하지 않으면 사용 가능한 API 키를 기준으로 자동 감지되며, SearXNG 를 포함해 대안 백엔드로 전환할 수 있습니다[S7]. 어떤 백엔드를 쓰느냐에 따라 검색·추출 요청이 실제로 거치는 서버가 달라지므로, 이 선택은 데이터가 어디를 지나가는지를 가르는 실질적인 결정입니다.

둘째는 사용 정보 수집 범위입니다. 개발 정책 문서는 통합된 사용자 대면 opt-in(선택적 동의) 게이트 — 설정 게이트, 설치 시 안내, 도구 토글 — 가 갖춰지기 전까지는 새로운 외부 전송 텔레메트리(사용 정보 원격 수집)나 제3자 식별자 태깅을 병합하지 않는다고 명시합니다[S8]. 이것은 정책이지 "현재 아무 계측도 없다"는 사실 확인은 아닙니다. 실제로 열려 있는 이슈는 토큰 회계, 비용 추정, 세션 길이 등 부분적인 계측 조각이 이미 존재하며, 다만 이를 하나로 모으는 통합 계측 계층이 없다고 밝히고 있습니다[S9]. 이 계측 조각들이 로컬에만 남는지, 아니면 어딘가로 전송되는지는 공개된 자료만으로는 확정되지 않습니다.

두 사실을 하나로 뭉치면 "텔레메트리가 전혀 없다"는 과장된 결론에 이르기 쉽습니다. 정확한 확인 항목은 두 갈래로 나뉩니다. 하나는 우리가 선택한 검색·추출 백엔드가 어떤 서버로 요청을 보내는지 우리 환경에서 직접 확인했는가이고, 다른 하나는 이미 존재하는 토큰·비용·세션 계측 조각이 우리 조직의 데이터 처리 정책과 충돌하지 않는지 검토했는가입니다.

8.3 공개 사례로 규모를 가늠하기

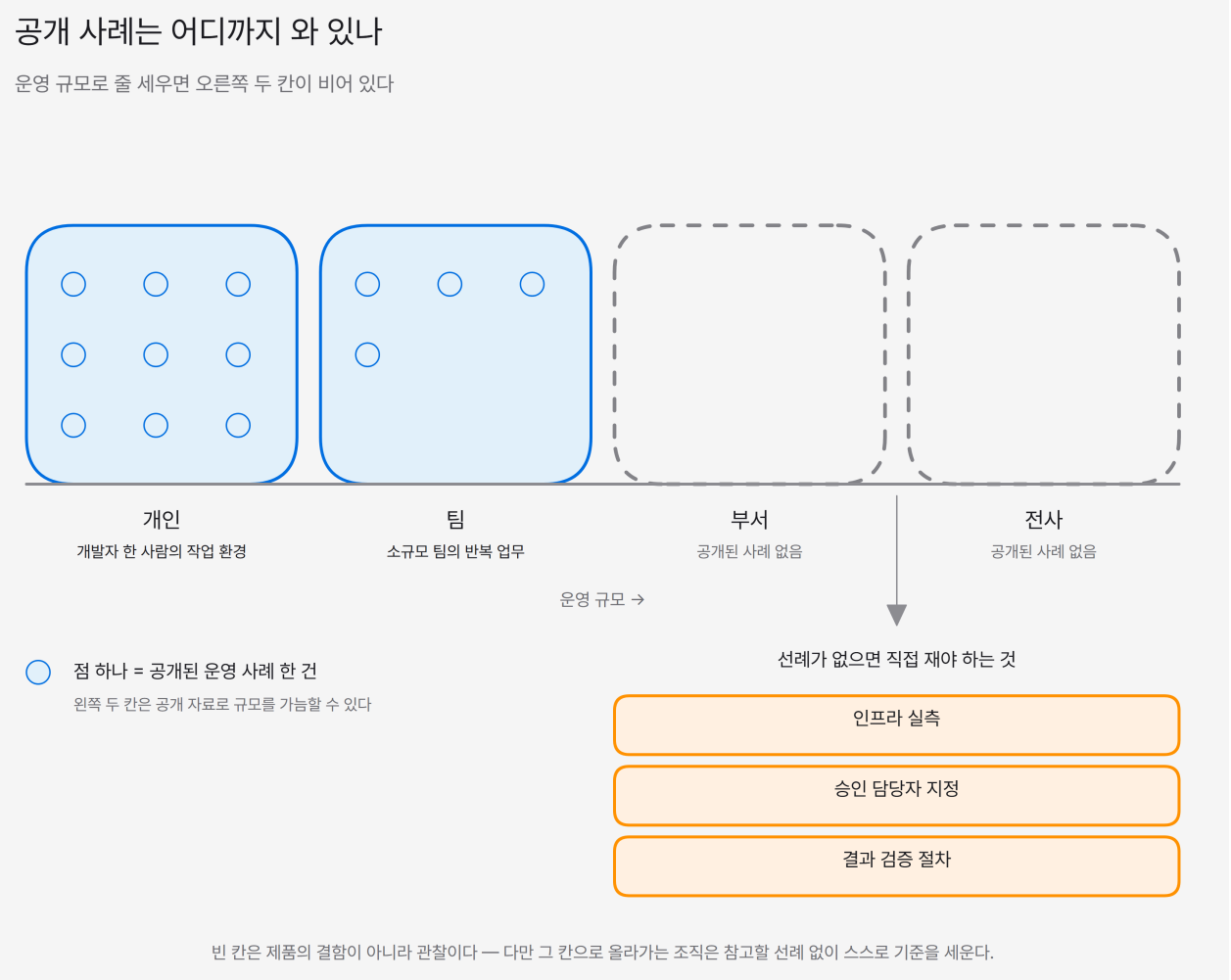
8.3.1 공개된 사례는 어떤 규모에 몰려 있는가

8.1 절과 8.2 절의 물음에 전부 답했다고 해도 마지막 질문 하나가 남습니다. 이 구성으로 실제로 운영해 본 조직이 있는가, 그 규모는 우리와 비슷한가 하는 질문입니다. 이 질문에 답하려고 공개된 활동을 살펴보면, 몰려 있는 지점이 뚜렷합니다.

사전조사에서 확인되는 공개 활동은 저장소의 커뮤니티 이슈 논의 같은 개인 개발자·소규모 단위의 언급에 그칩니다[S9]. 조직명이 붙은 전사 규모 도입 사례, 이를테면 "어느 기업이 몇 명 규모 팀에 이 구성을 붙여 몇 개월째 운영 중이다"는 식의 공개 사례는 조사 범위 안에서 확인되지 않았습니다.

이 사실을 부족함으로 읽을 필요는 없습니다. 이것은 공개된 사례의 규모 분포가 지금 이렇다는 관찰일 뿐입니다. 다만 관찰이라고 해서 무게가 가벼워지는 것은 아닙니다. 의사결정권자가 스스로 물어야 할 질문은 분명합니

다. 전사 규모의 공개 사례가 없다면, 우리 조직이 그 규모의 첫 사례가 되는 것은 아닌가. 이 물음에 그렇다고 답하게 된다면, 도입은 참고할 선례 없이 스스로 기준을 세우는 프로젝트가 됩니다.



8.3.2 지원 범위 선언과 고객 운영 사례를 구분했는가

공식 자료를 읽을 때 흔히 걸려 넘어지는 지점이 하나 더 있습니다. README 는 "5달러짜리 VPS, GPU 클러스터, 서버리스 인프라 어디에서도 돌아간다"고 밝힙니다[S2]. 이 문장은 제품이 어떤 인프라 형태를 지원하도록 설계되었는지에 대한 벤더의 지원 범위 선언입니다.

이 문장은 실제로 그런 규모에서 운영 중인 고객이 있다는 사례 보고가 아닙니다. 지원 범위 선언과 운영 사례는 서로 다른 종류의 진술이며, 이 백서의 사전조사 범위 안에서는 GPU 클러스터나 서버리스 규모로 이 제품을 실제로 운영하고 있다는 공개된 고객 사례를 확인하지 못했습니다.

두 문장을 접속사로 이어 "다양한 규모에서 돌아가니 우리 규모에서도 검증되었을 것이다"라는 하나의 결론으로 읽으면, 도입 판단의 근거가 실제보다 부풀려집니다. 지원 범위는 제품이 기술적으로 감당하도록 설계된 폭이고, 운영 사례는 누군가 그 폭 안에서 실제로 오래 버텨 냈다는 증거입니다. 이 둘을 각각 따로 확인하는 습관이 이 백서가 마지막으로 남기는 확인 항목입니다.

이 장 전체를 관통하는 물음은 결국 여덟 가지로 정리됩니다. 도입을 결정하기 전에, 우리 조직의 이름으로 아래 물음에 하나씩 답해 보십시오.

- 우리 인프라에 Python 3.11, Node.js, ripgrep 을 설치할 수 있는가, 그리고 우리 업무 규모로 실측해 볼 테스트 환경이 있는가
- 여덟 항목 표에서 비어 있는 칸은 몇 개이고, 그 칸을 채울 책임자는 누구인가
- 무인 실행 범위를 우리 기준으로 좁힐 수 있다는 것을 직접 확인했는가
- 위험 명령 승인 알림은 누구에게 가고, 그 사람이 부재중일 때는 누가 대신하는가
- 우리가 선택한 검색·추출 백엔드는 어떤 서버로 요청을 보내는가
- 이미 존재하는 토큰·비용·세션 계측 조각이 우리 조직의 데이터 처리 정책과 충돌하지 않는가
- 실행 결과를 누가, 어떤 절차로, 무엇을 기준으로 검증할 것인가. 이 답을 우리 조직이 직접 설계했는가
- 우리와 비슷한 규모의 공개 운영 사례가 없다면, 그 첫 사례가 되어도 괜찮은가

이 여덟 개 물음에 전부 답할 수 있을 때, 도입 판단은 비로소 우리 조직의 것이 됩니다.

Appendix

References

[S1] Hermes Agent 공식 문서 (2026). Documentation Home. https://hermes-agent.nousresearch.com/docs/
[S2] NousResearch/hermes-agent (2026). README.md. https://raw.githubusercontent.com/NousResearch/hermes-agent/main/README.md
[S3] Hermes Agent 공식 문서 (2026). Security — User Guide. https://hermes-agent.nousresearch.com/docs/user-guide/security
[S4] Hermes Agent 공식 문서 (2026). Skills System — User Guide. https://hermes-agent.nousresearch.com/docs/user-guide/features/skills
[S5] Hermes Agent 공식 문서 (2026). FAQ & Troubleshooting. https://hermes-agent.nousresearch.com/docs/reference/faq
[S6] Hermes Agent 공식 문서 (2026). Features Overview — User Guide. https://hermes-agent.nousresearch.com/docs/user-guide/features/overview
[S7] Hermes Agent 공식 문서 (2026). Integrations. https://hermes-agent.nousresearch.com/docs/integrations/
[S8] NousResearch/hermes-agent (2026). AGENTS.md. https://raw.githubusercontent.com/NousResearch/hermes-agent/main/AGENTS.md
[S9] NousResearch/hermes-agent GitHub Issues (2026). feat(observability): unified telemetry + analytics for latency, cost, and completion/failure rates (Issue #6642). https://github.com/NousResearch/hermes-agent/issues/6642
[S10] OWASP Gen AI Security Project (2025). OWASP Top 10 for Agentic Applications for 2026. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/
[S11] Hunt.io (2026). Thailand's Ministry of Finance Targeted With Hermes AI Agent Running Unattended, Hades Implant Staged. https://hunt.io/blog/thailand-ministry-finance-targeted-with-hermes-ai-agent
[S12] NousResearch/hermes-agent GitHub Issues (2026). Feature: Gateway Permission Tiers — Role-Based Access Control (Owner/Admin/User/Guest) for Messenger Platforms (Issue #527). https://github.com/NousResearch/hermes-agent/issues/527
[S13] Hermes Agent 공식 문서 (2026). Docker — User Guide. https://hermes-agent.nousresearch.com/docs/user-guide/docker

함께 읽을 자료 — 개념 배경을 짧게 정리한 공개 글입니다. 본문의 확인 근거는 위 [S...] 목록입니다.

[R1] MSAP.ai (2026). AI 에이전트와 챗봇의 차이 — 자율성·도구·메모리로 가르는 5가지 기준. https://www.msap.ai/blog-home/ai-blog/ai-agent-vs-chatbot/
[R2] MSAP.ai (2026). Agentic AI(에이전틱 AI)란 무엇인가 — 생성형 AI의 다음 단계. https://www.msap.ai/blog-home/ai-blog/what-is-agentic-ai/
[R3] MSAP.ai (2026). AX(AI 전환)란 무엇인가 — 정의부터 DX와의 차이, 추진 4단계까지.

https://www.msap.ai/blog-home/blog/what-is-ax/
[R4] MSAP.ai (2026). AX와 DX — 디지털 전환과 AI 전환, 무엇이 다른가. https://www.msap.ai/ax/ax-vs-dx/

Glossary

본문에서 처음 나올 때 풀어 쓴 낱말을 한자리에 모았습니다. 본문을 읽다 낱말이 막히면 여기로 돌아오시면 되고, 검토 회의에서 용어를 맞출 때도 이 표를 그대로 쓰실 수 있습니다.

용어	정의
agentskills.io	에이전트 스킬을 위한 공개 규격 사이트로, 스킬 문서의 형식이 특정 제품에 갇히지 않고 호환되도록 하는 공개 표준
DM 짝짓기 인증	메신저 계정과 에이전트 사용 권한을 연결할 때 거치는 인증 절차. 8자리 비모호 알파벳 코드, 1시간 TTL, 사용자당 10분 1회 요청 제한, 실패 5회 시 1시간 잠금으로 구성
Firecrawl	웹 페이지 검색·추출 요청을 처리하는 기본 백엔드 서비스
MCP(Model Context Protocol)	도구·서비스를 표준화된 방식으로 연결하는 프로토콜. 이 프로토콜을 통해 오가는 자격증명을 거르는 계층이 안전 장치 체계에 포함됨
Nous Portal	에이전트 개발사가 직접 운영하는 별도의 모델 게이트웨이 경로
OpenRouter	여러 모델 제공자를 하나의 API로 중계하는 서드파티 라우팅 서비스
OWASP Top 10 for Agentic Applications 2026	에이전트형 애플리케이션에서 반복적으로 나타나는 위험을 열 개 범주로 정리한 OWASP Gen AI Security Project의 표준 기구 자료
TTL(Time To Live)	발급된 인증 코드가 만료되기까지 유지되는 유효 기간
VPS(Virtual Private Server)	가상 사설 서버. 에이전트 런타임을 소규모로 실행할 수 있는 최소 단위 환경의 예시로 문서에 언급됨
WSL(Windows Subsystem for Linux)	윈도우용 리눅스 하위 시스템. systemd 지원이 불안정해 상시 구동 환경으로 쓸 때 제약이 있다고 공식 문서가 밝힌 대상
YOLO 모드	승인 없이 위험한 명령까지 그대로 실행하게 하는 설정
검증(실행 결과 검증)	실행이 끝난 뒤 결과가 실제로 맞는지 확인하는 절차. 여덟 항목 가운데 제품이 기본 제공하지 않는 유일한 항목
권한 분리	에이전트의 실행 권한을 사용자·역할별로 나누어 통제하는 기능 요구

용어	정의
리그렙(ripgrep)	코드 저장소를 빠르게 검색하는 명령행 도구로, 설치 전제 조건 세 가지 중 하나
무인 실행	사람이 각 단계마다 개입하지 않아도 목표까지 실행을 이어가는 능력
세션 간 격리	서로 다른 세션의 실행이 서로 간섭하지 않도록 분리하는 안전장치 계층
스킬(skill)	코어가 필요할 때 불러 쓰는 온디맨드 지식 문서로, 절차 기억을 구현하는 메커니즘
실행 격리	도구가 시스템 자원에 접근하는 범위를 물리적·논리적으로 가두는 장치
온디맨드(on-demand)	필요한 시점에만 불러 쓰는 방식
원본 대조(origin hash)	번들로 제공되는 스킬이 로컬에서 수정되었는지 여부를 대조하는 값
위험 명령 승인	삭제·전송처럼 되돌리기 어려운 명령을 실행 직전에 멈추고 사람의 승인을 요구하는 안전장치 계층
입력 정제	채널로 들어오는 입력에서 위험한 문자열·패턴을 걸러내는 안전장치 계층
잠금(lock-in)	특정 벤더의 폐쇄적 형식에 자산이 갇혀 다른 도구로 옮기기 어려워지는 위험
절차 기억	같은 일을 다시 지시하지 않아도 되도록 수행 절차를 저장해 두는 능력
점진적 공개(progressive disclosure)	절차 전체를 한 번에 읽어들이지 않고, 필요한 부분만 그때그때 펼쳐 보는 방식
채널 통합	사람이 화면 앞에 없을 때도 요청이 들어올 수 있는 입구를 여러 메시징 플랫폼으로 연결하는 기능 요구
컨테이너 격리(Container isolation)	도구가 실행되는 환경을 컨테이너 안에 가두어 호스트 시스템 전체로 번지지 않도록 하는 안전장치 계층
코어(core)	대화를 주고받고 실제 작업을 실행하는 본체
파일 쓰기 안전장치(File write safety)	에이전트가 도구 실행 과정에서 손댈 수 있는 파일의 범위를 제한하는 안전장치 계층
프로파일	하나의 설치에서 분리 운용하는 여러 실행 단위

함께 읽을 자료

이 백서가 다루는 개념을 더 짧은 글로 먼저 확인하고 싶을 때 읽을 수 있는 공개 자료입니다. 검토 회의 전에 팀원과 배경을 맞출 때도 그대로 쓰실 수 있습니다.

주제	이 백서의 어디와 이어지나	링크
[R1] AI 에이전트와 챗봇의 차이 — 자율성·도구·메모리로 가르는 다섯 기준	1장 1.1 챗봇과 에이전트의 경계	https://www.msap.ai/blog-home/ai-blog/ai-agent-vs-chatbot/
[R2] Agentic AI(에이전틱 AI)란 무엇인가 — 생성형 AI의 다음 단계	1장·2장 제품 정의와 실행 방식	https://www.msap.ai/blog-home/ai-blog/what-is-agentic-ai/
[R3] AX(AI 전환)란 무엇인가 — 정의부터 추진 4단계까지	8장 도입 판단	https://www.msap.ai/blog-home/blog/what-is-ax/
[R4] AX와 DX — 디지털 전환과 AI 전환은 무엇이 다른가	8장 도입 판단	https://www.msap.ai/ax/ax-vs-dx/

기업이 AI 에이전트에 요구하는 여덟 가지 — Hermes Agent 는 어디까지 기본으로 주나

CONTACT

WEB
msap.ai
www.msap.ai/

EMAIL
hello@msap.ai

TEL
02-6953-5427
0269535427

YOUTUBE
@msaptv
www.youtube.com/@msaptv

LINKEDIN
linkedin.com/showcas...
www.linkedin.com/showcase/msap-ai/

FACEBOOK
facebook.com/opennaru
www.facebook.com/opennaru

SCAN