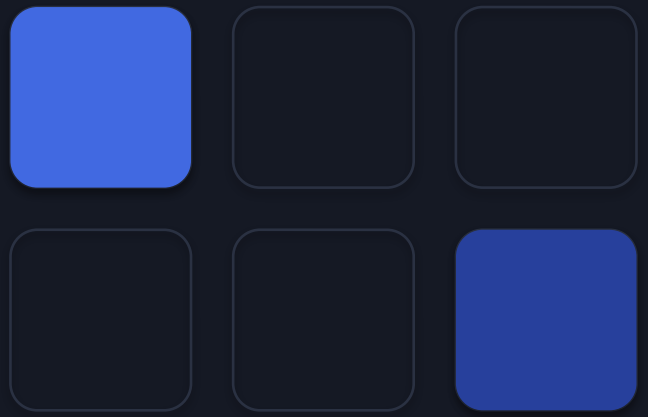
MICROSOFT COPILOT AGENTIC BLOG · 시리즈 완본

New Copilot Studio CLI 에이전트



하네스 설계 원리로 다시 읽는 Copilot Studio — 개념 · 빌드 · 실습 3부 완본

0부 개요 · 1부 What's New · 2부 빌드 개념 · 3부 실습(세일즈 어시스턴트)

원문 : chichoi1991.github.io/Agent_Blog · 2026-06-15

⚠ 프리뷰 기준 — 기능·화면·일정은 변경될 수 있습니다 (subject to change)

0부 · 개요

이 시리즈는 무엇인가

New Copilot Studio는 단순한 UI 개편이 아닙니다. Anthropic·OpenAI·GitHub가 코딩 에이전트에서 검증한 **"하네스(harness) 설계"** 원리를 엔터프라이즈 에이전트 플랫폼에 이식한, **AI 코어의 재건축**입니다.

이 시리즈는 그 변화를 **개념 → 빌드 → 실습** 3부로 나누어 다룹니다. 모든 기능을 "Microsoft가 추가한 기능"이 아니라 **"업계 원리 → Microsoft 구현"**으로 읽으면 훨씬 정확하게 이해됩니다.

개념편

1부 · What's New

하네스 엔지니어링이라는 업계 원리부터 에이전틱 루프·스킬·툴(MCP)·서브에이전트·메모리·워크플로우까지. 클래식→뉴 매핑과 CLI 런타임 해부.

빌드 개념편

2부 · 에이전트 생성

에이전트를 이루는 6개 구성요소(지침·스킬·도구·참고자료·연결된 에이전트·메모리)를 개념 → 작성법 → 함정 순서로.

실습편

3부 · 에이전트 생성

통신사 세일즈 어시스턴트를 처음부터 끝까지. 지침 → 스킬 → 패키지 → 도구 → 테스트 → 배포 순으로 조립.

읽는 순서 — 1부(원리) → 2부(빌드 개념) → 3부(실습)를 권장합니다. 빠르게 만들어보고 싶다면 1부 1·2장만 읽고 3부로 건너뛴 뒤, 막히는 지점에서 2부를 참조해도 됩니다.

0부 · 개요

왜 다시 지었나 — 한눈에

기존(클래식) Copilot Studio는 본질적으로 **대화 흐름 설계 도구**였습니다. 메이커가 모든 경로를 미리 그려야 했고, 시나리오가 복잡해지면 분기가 폭발하고 다단계·장기 작업에서 흐름이 끊겼습니다.

New Copilot Studio는 **AI 코어(오케스트레이터)를 코딩 하네스 위에 다시 지어**, 메이커의 일을 "흐름 그리기"에서 **"하네스 설계"**로 바꿉니다.

구분	클래식 Copilot Studio	New Copilot Studio (CLI Agent)
패러다임	대화 흐름(Topics) 설계	하네스 설계 + 에이전틱 루프
메이커의 일	분기·노드를 직접 그림	지침·스킬·도구를 설계
다단계·장기 작업	흐름이 끊기기 쉬움	끝까지 재귀적으로 수행
참고자료 처리	검색 스니펫(RAG) 기반	원본 파일을 코드로 직접 처리
산출물	텍스트 응답 위주	Word·PPT·Excel·PDF 등 리치 파일
구성 화면	탭 9개로 분산	Build·Test·Preview·Monitor 4개로 통합

> 핵심 멘탈모델은 하나입니다 — **"무엇을(What)"은 사람이 적고, "어떻게(How)"는 에이전트가 한다.** 클래식과 뉴는 **나란히 공존**하며, 뉴는 opt-in이고 강제 전환은 없습니다.

0부 · 개요

핵심 용어 미리보기

용어	한 줄 정의
하네스 (Harness)	LLM을 실행 루프로 감싸 도구를 쓰게 만드는 런타임
에이전틱 루프	의도 해석 → 도구 선택 → 실행 → 관찰 → 판단을 반복하는 코어
스킬 (Skill)	필요할 때만 로드하는 재사용 markdown 지침 묶음
도구 (Tool)	에이전트가 외부에서 실제로 행동하는 손발 (커넥터 + MCP)
MCP	에이전트-도구 연결의 업계 표준 프로토콜
연결된 에이전트	컨텍스트를 격리해 전문 작업을 위임하는 협업 에이전트
컨텍스트 엔지니어링	"원하는 결과를 낼 가장 작은 고신호 토큰 집합"을 설계하는 일

PART 1

What's New 개념편

New Copilot Studio는 UI 개편이 아니라, Anthropic·OpenAI·GitHub가 코딩 에이전트에서 검증한 "하네스 설계" 원리를 엔터프라이즈 플랫폼에 이식한 것입니다.

모든 기능을 "Microsoft 기능"이 아니라 "업계 원리 → Microsoft 구현"으로 읽어야 정확합니다.

이 부에서 다루는 내용

1부 · 1. 왜 다시 만들었나

클래식의 구조적 한계

▶ **포인트** — 클래식은 "대화를 설계"하는 도구였고, 그 패러다임 자체가 다단계·장기 작업에서 천장에 부딪혔다. 그래서 Microsoft는 UI가 아니라 AI 코어 자체를 코딩 하네스 위에 다시 지었다.

메이커가 직접 그려야 했던 것

- **Topics** — "사용자가 X라고 말하면 → 노드 A → 노드 B"의 트리거-응답 트리
- **Prompts** — 특정 입력→특정 출력의 단발성 AI 호출
- **분기·조건·변수** — 모든 경우의 수를 노드로 연결

결정적 약점은 **모든 경로를 사람이 미리 상상해서 그려야 한다**는 것. 시나리오가 조금만 복잡해져도 분기가 폭발하고, 예상 밖의 말이 오면 흐름이 끊깁니다.

한계	증상
흐름이 깨짐	다단계 작업 중간에 예상 못한 입력이 오면 토픽이 끊김
분기 폭발	복잡해질수록 노드·조건이 기하급수적으로 증가
도구 호출이 경직	어떤 도구를 언제 부를지 메이커가 미리 다 설계
긴 작업 불가	"문서를 읽고 분석해 보고서를 써줘" 같은 long-horizon 작업이 어려움

1부 · 1. 왜 다시 만들었나

코어 재건축 — 무엇을 바꿨나

Anthropic이 *Effective harnesses for long-running agents*에서 정리한 두 실패 모드는, 클래식 봇이 무너지던 양상과 정확히 같습니다.

Anthropic이 진단한 실패 모드	클래식 봇에서의 증상
한 방에 다 하려다(one-shot) 컨텍스트 중간에 소진	다단계 작업이 중간에 끊기고 미완성 상태로 종료
진척을 보고 다 됐다고 조기 선언(declares the job done)	핵심 기능이 안 되는데 대화가 끝나버림

고객 피드백의 핵심 3가지

1. **업무의 자연스러운 일부**
   에이전트가 일하는 흐름 안에 들어오길 원한다
2. **신뢰할 수 있고 확장 가능한 플랫폼**
   reliable & scalable 에이전트 기반이 필요하다
3. **결정론적 워크플로우와의 공존**
   에이전트·앱·워크플로우가 함께 잘 동작하길 원한다

결과적으로 얻은 능력

- **지시 준수 강화** — 긴 대화에서도 원래 지침을 안 놓침
- **장기·다단계(long-horizon) 작업** — 중간에 무너지지 않고 끝까지
- **재귀적 실행** — 복잡한 문제를 스스로 단계로 쪼개 반복
- **대용량 콘텐츠 + 리치 파일 생성** — 컨테이너에서 Word/PPT/Excel/PDF 산출
- **빌드 경험 단순화** — 구성 탭 9개 → 4개, 추론·도구 호출 실시간 관찰

반론 대응 — "이거 그냥 UI 리스킨 아니냐?" → 신뢰성·다단계 처리 향상은 UI가 아니라 **새로운 AI 코어의 기능**입니다.

1부 · 2. 하네스 엔지니어링

하네스란 무엇인가 — 가장 중요한 정의

▶ **포인트** — "하네스"는 LLM을 실행 루프로 감싸 도구를 쓰게 만드는 런타임이다. 업계 최강 에이전트들은 전부 CLI 코딩 하네스로 수렴했고, New Copilot Studio는 그 패턴을 그대로 업무 에이전트에 가져왔다.

LLM 단독으로는 **다음 토큰 예측**만 합니다. 파일을 읽거나 API를 호출하거나 실패를 복구하지 못합니다.

> *에이전트 = LLM이 루프 안에서 자율적으로 도구를 사용하는 것*
>
> (LLMs autonomously using tools in a loop) — Anthropic

이 루프를 돌려주는 런타임이 **하네스(harness)**입니다.

하네스가 모델에게 제공하는 4가지

도구 접근

파일·셸·검색·API

관찰(observation) 피드백

도구 실행 결과를 다시 모델 입력으로

반복 제어(loop control)

작업이 끝날 때까지 판단→실행→관찰 반복

컨텍스트 관리

무엇을 언제 모델에게 보여줄지 결정

1부 · 2. 하네스 엔지니어링

업계는 이미 이 길로 수렴했다

2025년을 거치며 가장 검증된 에이전트들은 예외 없이 **CLI 기반 코딩 하네스** 형태로 모였습니다. 우연이 아닙니다.

사례	형태	특징
Claude Code (Anthropic)	터미널 agentic CLI	코드베이스를 읽고·수정·테스트. glob/grep/head/tail로 필요한 것만 적시 로드
GitHub Copilot CLI	셸·파일·도구 호출 코딩 하네스	터미널에서 작업 완수
OpenAI Codex CLI	코드 실행 하네스	동일 계열
New Copilot Studio	엔터프라이즈 에이전트 플랫폼	**같은 하네스 패턴을 업무 에이전트로 이식**

왜 하필 "코딩" 하네스인가

코딩은 에이전트 능력이 가장 먼저·가장 혹독하게 검증된 영역이기 때문입니다.
코드를 다루려면 필연적으로:

- 여러 파일을 읽고 맥락을 종합
- 어떤 도구(컴파일·테스트·검색)를 쓸지 정확히 선택
- 다단계로 추론
- 실패하면(테스트 깨짐) 원인을 보고 복구

이 네 가지는 **모든 업무 에이전트가 필요로 하는 바로 그 능력**입니다. 송장 처리든, 보고서 작성이든, 고객 응대든 근본 메커니즘은 같습니다.

Microsoft가 코어를 "coding harness + CLI layer"라고 명시한 건, **코딩에서 단련된 엔진을 일반 업무로 가져왔다는 선언**입니다.

1부 · 2. 하네스 엔지니어링

황금률 — 과설계하지 마라

> *하네스의 모든 구성요소는 '모델이 혼자 못 하는 것'에 대한 가정을 인코딩한다. 그 가정은 틀렸을 수도 있고, 모델이 좋아지면서 낡을 수도 있으니 끊임없이 스트레스 테스트하라. … **가능한 가장 단순한 해법에서 출발하고, 필요할 때만 복잡도를 더하라.***
>
> — Anthropic, *Designing harnesses for long-running apps* / *Building Effective Agents*

"과설계하지 마라. 모델이 좋아지면 당신의 스캐폴딩은 낡는다."

추상적 조언이 아닙니다. Anthropic은 실제로 Opus 4.5용으로 만든 정교한 스캐폴딩(sprint 분해, context reset)을 Opus 4.6에서 **덜어냈습니다.** 모델이 스스로 계획을 더 잘 세우게 되자, 하네스가 오히려 가벼워진 것.

교육 포인트 — 멘탈모델 전환

이전 — 대화 흐름(순서도)을 손으로 그림

모든 분기를 미리 상상해야 했다

이후 — 하네스에게 줄 지침·스킬·도구를 설계

그리고 처음부터 과설계하지 않는다

1부 · 3. 에이전틱 루프

오케스트레이터 = 에이전틱 루프

▶ **포인트** — 하네스의 심장은 "에이전틱 루프"다. 스킬·툴·메모리·서브에이전트는 전부 이 루프가 소비하는 재료일 뿐이다. 루프를 안 세우면 나머지가 공중에 뜬다.

루프 1회전

```
의도 해석 → 도구/스킬 선택 → 슬롯 필링 → 실행
    → 관찰(observe) → 판단(완료?) → (필요시) 반복
```

스킬·툴·서브에이전트·메모리는 독립 기능처럼 보이지만, 전부 이 루프 안에서만 의미를 가집니다.

괄호 안 정량 수치는 **내부 평가(NDA)** 자료이므로 외부 배포 자료에서는 제외하세요.

새 오케스트레이터가 강화한 4가지

1. **정확한 도구 선택 + 슬롯 필링**
   첫 시도에 올바른 도구를 올바른 입력으로 호출
2. **재귀적 작업 실행**
   문제를 단계로 쪼개 끝까지 체계적으로 반복
3. **적응(adaptation)**
   실패·에러 시 자동 재시도하거나 대안 경로를 찾아 계속
4. **장기 지시 준수**
   긴 세션 내내 원래 지침을 유지

1부 · 3. 에이전틱 루프

신뢰성에 집중하는 진짜 이유 — 두 실패 상황

실패 상황 1 — 컨텍스트 일관성 상실 + "context anxiety"

컨텍스트 창이 차오르면 모델이 일관성을 잃습니다. 더 심하게는, 한계에 가까워졌다고 느끼면 작업을 조급하게 마무리합니다.

해법

- **Compaction** — 대화를 요약·압축해 같은 에이전트가 짧아진 이력으로 계속
- **Context reset** — 컨텍스트를 비우고 새 에이전트로, 구조화된 핸드오프로 상태 전달

실패 상황 2 — 자기 평가의 관대함

모델은 자기 결과물을 평가하라고 하면 거의 항상 과대 칭찬합니다. 해법은 **작업자(generator)와 평가자(evaluator)를 분리**하는 것.

Anthropic은 GAN에서 착안해 generator↔evaluator 루프를 만들었고, "자기 일을 비판하게 만드는 것보다 **별도 평가자를 회의적으로 튜닝**하는 게 훨씬 쉽다"고 결론냈습니다.

→ 이 **"평가자 분리"** 원리가 New Copilot Studio의 **풀페이지 테스트 + 노드별 eval**로 직결됩니다.

구분	동작 방식
이전 모델	"송장 PDF에서 금액 추출 → 임계값 비교 → 승인/반려"를 노드로 일일이 분기. 새 PDF 양식이 오면 깨짐.
새 모델	목표("송장을 검토해 승인 여부 판단")와 도구만 주면, 루프가 알아서 추출·비교·판단하고 실패 시 재시도하거나 다른 방법을 시도.

스킬 — load-on-demand markdown

▶ **포인트** — 스킬은 "X를 어떻게 하는지"를 markdown으로 한 번 적어두고 필요할 때만 불러 쓰는 재사용 지침이다. 단순 편의 기능이 아니라 Anthropic이 말하는 **"just-in-time 컨텍스트"의 엔터프라이즈 구현**이며, 외부(GitHub Copilot·Claude) 스킬까지 그대로 가져올 수 있는 업계 호환 자산이다.

온디맨드 로드

평소엔 컨텍스트에 넣지 않다가, 그 작업이 필요해질 때만 로드

use case 패키징

언제·어떻게 정보를 묻고 보여줄지까지 한 묶음으로 packaging

여러 에이전트 재사용

한 번 만들면 조직 내 여러 에이전트가 그대로 재사용

왜 "필요할 때만 로드"인가 — 컨텍스트 엔지니어링의 핵심

- **컨텍스트는 유한 자원(attention budget)** — 트랜스포머는 n개 토큰에 대해 n^2 쌍 관계를 계산하므로, 컨텍스트가 길어질수록 주의가 얇게 퍼집니다.
- **Context rot(컨텍스트 부패)** — 토큰 수가 늘수록 모델의 회상 정확도가 떨어집니다. 큰 컨텍스트 창이 답이 아닙니다.
- **목표는 "원하는 결과를 낼 가장 작은 고신호(high-signal) 토큰 집합"을 찾는 것.**
- **Just-in-time 검색** — 경량 식별자(파일 경로·쿼리·링크)만 들고 있다가 런타임에 필요한 것만 로드. Claude Code의 CLAUDE.md + glob/grep가 정확히 이 패턴.

1부 · 3. 스킬

업계 호환성 · 로드맵 · 클래식과의 관계

외부 자산 재활용

- **GitHub Copilot** 스킬 import
- **Claude Code(Claude)** 스킬 import
- Copilot Cowork 등 타 플랫폼 스킬도

Anthropic은 frontend-design 같은 스킬을 공개 SKILL.md로 배포합니다. New Copilot Studio가 이런 스킬을 import한다는 건, **Microsoft 울타리 밖 오픈소스 생태계 자산을 직접 끌어 쓴다**는 의미입니다.

스킬 로드맵

항목	내용	시점
Reusable / import	디자이너 작성 또는 외부 import	Preview 06/2026
Skills with resources	문서·템플릿·python 등 리소스 동봉	Preview 06/2026
Skill marketplace	조직 내 공유·자동 최신화	Preview 06/2026

클래식과의 관계

클래식	New Copilot Studio	비고
Topics	스킬 + 워크플로우	스킬이 "특정 상황의 응답 방법"을, 워크플로우가 "여러 액션 엮기"를 담당
Prompts	스킬(에이전트) / agent node(워크플로우)	새 코어 전환으로 대체
토픽 확장점	향후 릴리스	수신/발신 메시지 가로채기 등. 클래식 토픽은 클래식 에이전트에 존속

1부 · 4. 툴 · MCP

툴 = 에이전트의 손발 · 설계 3원칙

▶ **포인트** — 툴은 에이전트가 외부 세계에서 실제로 행동하는 손발이다. **"커넥터(Microsoft 자산) + MCP(업계 표준)"** 두 축으로 이해하라. 그리고 Anthropic의 경고를 새겨라 — 도구를 많이 붙일수록 좋은 게 아니라, 오히려 에이전트를 망친다.

Anthropic의 툴 설계 3원칙

1. **툴은 토큰 효율적이어야 한다**
   결과를 간결하게 반환
2. **기능이 겹치지 않아야 한다**
   "엔지니어조차 어떤 툴을 써야 할지 단언 못 하면, 에이전트는 절대 더 못 한다."
3. **최소 viable 툴셋으로 큐레이션하라**
   가장 흔한 실패가 비대한 툴셋(기능 과잉·모호한 선택 지점)

실습 시사점 — 커넥터/도구를 무지성으로 다 붙이지 마세요. 목적당 최소 도구만. 도구가 10개면 에이전트는 매 턴 "뭘 쓸지" 헷갈립니다.

두 개의 축

축	정체 / 성격
커넥터	Power Platform 1,000+ 커넥터 — Microsoft 고유 자산, 검증된 엔터프라이즈 연결
MCP	Model Context Protocol — 에이전트-도구 연결 업계 표준. Microsoft 독점 아님

MCP가 왜 중요한가 — 커넥터로 못 닿는 광범위한 도구·앱을 표준 방식으로 붙이되, Microsoft 보안·권한·컴플라이언스 경계 안에서 실행한다는 의미입니다. 워크플로우 노드도 MCP 도구를 호출할 수 있습니다(휴먼인더루프 포함, Preview 06/2026).

1부 · 5. 서브에이전트

서브에이전트 — 본질은 컨텍스트 격리

▶ **포인트** — 서브에이전트의 본질은 "컨텍스트 격리"다. 전문 에이전트가 깨끗한 컨텍스트에서 깊이 작업하고 요약만 돌려주면, 메인 에이전트는 깔끔하게 유지된다. New Copilot Studio는 클래식의 child agent를 버리고 **"스킬 + 연결된 에이전트"**로 재구성했다.

*메인 에이전트가 고수준 계획을 잡고, 전문 sub-agent가 깨끗한 컨텍스트에서 집중 작업합니다. 각 sub-agent는 수만 토큰을 써가며 깊이 탐색하지만, **압축된 요약(보통 1,000~2,000 토큰)만 반환**합니다. 상세 탐색 컨텍스트는 sub-agent 안에 격리되고, 메인은 종합·분석에 집중 → "관심사의 분리".*

— Anthropic, *Effective context engineering* / *Multi-agent research system*

멘탈모델 변화

구성요소	역할
스킬	재사용 가능한 역량을 패키징
연결된 에이전트	다른 에이전트를 끌어와 협업

왜 더 나은가 — child agent는 자기 컨텍스트 안에서만 돌아 인터럽트·의도 전환에 약했습니다. 스킬 기반 구성은 인터럽트, 의도 전환, 멀티 인텐트 질의("A도 하고 B도 해줘")에 훨씬 자연스럽게 대응합니다.

현재 제약

GA 시점엔 Copilot Studio 에이전트끼리만 연결 가능

Microsoft Foundry · M365 Agents SDK · A2A 프로토콜 기반 외부 에이전트 연결은 곧 추가 예정(외부 자료엔 "확장 예정"으로만)

harness design 문서의 **planner · generator · evaluator** 3-에이전트 구조(GAN형)는 역할 분리 설계의 정석입니다. 각 에이전트가 "직전 실행에서 발견한 특정 약점"을 메우도록 설계됐습니다.

1부 · 6. 메모리

메모리의 3대 기법 (Anthropic)

▶ **포인트** — 메모리는 추상 개념이 아니라 "구조화된 노트 쓰기"라는 구체적 기법이다. 에이전트가 컨텍스트 밖 파일에 진척을 기록하고 필요할 때 다시 읽어, 장기 작업에서도 목표를 잃지 않는다. 스킬과 메모리는 **"컨텍스트 엔지니어링"이라는 한 원리의 두 얼굴**이다.

①

Compaction (압축)

컨텍스트가 한계에 가까우면 대화를 요약·압축해 새 창에서 재시작.

아키텍처 결정·미해결 버그·구현 디테일은 보존하고, 중복 도구 출력은 폐기.

②

Structured note-taking

에이전트가 컨텍스트 밖 파일에 진척을 기록하고 나중에 다시 로드.

claude-progress.txt + git 로그, *Claude plays Pokémon*의 수천 스텝 추적.

③

Sub-agent 아키텍처

컨텍스트를 격리해 메인을 깨끗하게 유지.

전문 에이전트가 깊이 탐색하고 압축 요약만 반환.

기법	언제 쓰나
Compaction	왕복이 많은 대화
Note-taking	마일스톤이 뚜렷한 반복 개발
Multi-agent	병렬 탐색이 이득인 리서치

"컨텍스트는 소중하고 유한한 자원이다." — Anthropic, *Effective context engineering*

1부 · 7. 워크플로우

에이전트 + 워크플로우 — 왜 둘을 나누는가

▶ **포인트** — 에이전트는 "열린 작업"을, 워크플로우는 "결정론적·반복·통제 가능 작업"을 맡는다. 둘을 한 캔버스에서 결합하는 게 New Copilot Studio의 핵심이며, 이는 Anthropic의 **"가장 단순한 해법에서 출발하라"**와 정확히 일치한다 — 예측 가능한 단계는 결정론으로 싸게, 판단이 필요한 부분만 에이전트로 비싸게.

워크플로우 디자이너 핵심

통합 비주얼 캔버스

드래그앤드롭으로 단계 구성, 전체 흐름을 한눈에

노드 단위 테스트

전체를 돌리지 않고 깨진 한 스텝만 콕 집어 테스트

버전 관리

변경 추적, 자신감 있는 게시

분석(analytics)

엔드투엔드 프로세스 가시성·성능 모니터링

양방향 결합

워크플로우 → 에이전트 : agent node로 위임

에이전트 → 워크플로우 : "에이전트가 이 플로우를 호출할 때" 트리거를 가진 워크플로우를 에이전트의 도구로 등록

"Power Automate랑 뭐가 다른가?" — 워크플로우는 결정론적 자동화에 에이전트 능력(agent node·인텔리전트 액션·버저닝)을 블렌딩한 것. 클래식 flow(=agent flows)도 계속 동작하고 변환도 가능.

Anthropic 하네스가 브라우저를 사람처럼 클릭해 E2E 검증한 것과 Copilot Studio의 노드 테스트는 같은 계보 — **검증은 실제 사용 경로로.**

1부 · 7. 워크플로우

에이전트와 결합하는 장치

장치	설명	시점(예정)
Agent node	워크플로우 한 스텝으로 에이전트 호출. 예측 단계는 결정론, 유연성 필요 부분만 위임	GA 06/2026
MCP tools in workflows	워크플로우가 MCP 도구·앱 호출	Preview 06/2026
자연어 진입점	"원하는 자동화를 말로 설명" → 생성·구성·검증	Preview 06/2026
M365 Copilot 노드	Researcher·Analyst·커스텀 M365 에이전트를 직접 호출	—
Classify / Extract	분류·추출을 인라인 의사결정에 활용	—
Computer-using agents	API 없는 데스크톱·브라우저 앱을 UI 자동화	Preview 06/2026
클래식 agent flows 업그레이드	기존 flow를 워크플로우로 변환	GA 06/2026

1부 · 8. 매핑

클래식 → 뉴 매핑표

▶ **포인트** — 핵심 변화는 "Topics·Prompts·Child agents"가 "스킬·워크플로우·연결된 에이전트"로 재편됐다는 것. 단, 강제 전환은 없고 클래식과 뉴는 나란히 공존한다.

클래식	New Copilot Studio	비고
Topics	스킬 + 워크플로우	메시지 가로채기 등 일부 확장은 향후
Prompts	스킬(에이전트) / agent node(워크플로우)	새 코어 전환으로 대체
Child agents	connected agents + 스킬	인터럽트·멀티 인텐트에 강함
Agent flows	워크플로우	업그레이드/변환 가능
구성 탭 9개	4개 (Build · Test · Preview · Monitor)	Knowledge·Tools·Channels·Agents는 Build의 components
분리된 지침/지식/도구	한 화면 통합	풀페이지 테스트 + inline CoT · tool calls

클래식과 뉴는 나란히 동작합니다. 뉴는 opt-in(홈 "Try now"), 기본 off, 강제 전환 없음.

권장 경로 — 뉴를 켜고 다음 프로덕션 에이전트를 뉴에서 빌드, 클래식은 그대로 운영. 정식 마이그레이션 가이드는 추후.

PART 2

에이전트 생성 빌드 개념편

1부의 원리(하네스·스킬·툴·메모리·컨텍스트 엔지니어링)를 실제 에이전트로 빌드할 때 알아야 할 6개 구성요소를 개념 → 작성법 → 함정까지 정리합니다.

핵심 멘탈모델 하나만 기억하세요 — "무엇을(What)"은 사람이 적고, "어떻게(How)"는 에이전트가 한다.

이 부에서 다루는 내용

1. **지침 (Instructions)**
   항상 로드되는 상위 규칙 — 짧고 명확하게
2. **스킬 (Skills)**
   필요할 때만 로드하는 재사용 지침 패키지
3. **도구 & 참고자료**
   손발(커넥터·MCP)과 읽는 지식(RAG→코드)
4. **연결된 에이전트 & 메모리**
   컨텍스트 격리와 단기·장기 2계층 메모리
5. **종합 & 체크리스트**
   6요소가 한 요청에서 맞물리는 법 + 함정

2부 · 0. 들어가며

빌드의 6요소

요소	한 줄 정의	비유 (신입 직원)	1부 근거
지침	항상 켜져 있는 상위 규칙(헌법)	근무 수칙 — 항상 지켜야 할 최상위 규칙	1·2장
스킬	필요할 때만 꺼내 쓰는 재사용 지침 묶음	업무 매뉴얼 — 필요할 때 펼치는 문서	3장
도구	외부에서 실제로 행동하는 손발	연장 — 실제로 무언가를 실행하는 수단	4장
참고자료	에이전트가 읽는 지식(엑셀·문서)	참고 문서·자료실 — 근거로 삼는 지식	9.4장
연결된 에이전트	전문 작업을 위임하는 협업 에이전트	전문 동료 — 깊은 작업을 위임하는 협업자	5장
메모리	대화·작업의 맥락 유지	업무 노트 — 대화·작업의 맥락 보관	6·9.6·9.7장

핵심 멘탈모델 — "무엇을(What)"은 사람이 적고, "어떻게(How)"는 에이전트가 한다. 지침·스킬은 *무엇을*에 집중하고, *어떻게*(코드·경로·라이브러리)는 코어에 맡깁니다.

6요소는 유능한 신입에게 일을 맡기는 것과 같습니다. 당신은 환경을 갖춰주고, **일하는 방식 자체는 에이전트가 정하게** 둡니다.

2부 · 1. 지침 (Instructions)

지침 — 에이전트의 헌법

▶ **포인트** — 지침은 **항상 컨텍스트에 로드되는 상위 규칙**이다. 그래서 짧고·명확하고·"무엇을"에 집중해야 한다. 가장 흔한 실수는 (1) 과설계, (2) 세부를 지침에 박아 스킬을 우회하게 만드는 것이다.

개념

- 에이전트가 **모든 대화에서 항상 참조**하는 최상위 규칙. 회사로 치면 헌법·취업규칙.
- 스킬(필요할 때 로드)과 달리 지침은 **늘 켜져 있으므로**, 길수록 모든 응답이 무거워지고 핵심이 묻어집니다(attention budget).
- 따라서 지침에는 **항상 지켜야 할 소수의 원칙**만 담고, 상황별 디테일은 스킬로 내립니다.

구조: 역할 → 작업별 규칙 → 가드레일. 짧은 섹션으로 끊어 스캔하기 쉽게 만듭니다.

좋은 지침 구조 (템플릿)

```
# [에이전트 이름] — 지침

[한 문단: 역할과 무엇을 도와주는지]

## [작업 A]를 할 때
- 규칙 (무엇을, 쉬운 말로)
- 세부는 스킬을 가리키기

## [작업 B]를 할 때 (비가역이면 "중요" 표시)
- 확인 게이트, 금지선

## 지킬 점 (가드레일)
- 항상/절대 규칙 3~5개
```

2부 · 1. 지침 (Instructions)

지침 작성 6원칙

1 "무엇을"만, "어떻게"는 맡긴다

메이커는 목표·규칙만 쉬운 말로 적고, 코드·라이브러리·경로 같은 구현은 에이전트에 맡깁니다.

2 일반 사용자가 읽을 수 있게

대부분의 메이커는 개발자가 아닙니다. 디렉토리·함수명·전문용어를 빼고 업무 언어로.

3 단일 출처 — 지침에 세부를 박지 마라

세부를 지침·스킬 양쪽에 적으면 에이전트가 "지침에 이미 있네" 하고 스킬을 건너뜁니다.

4 비가역 행동에는 확인 게이트(HITL)

메일 발송·외부 전송처럼 되돌리기 어려운 작업은 반드시 사용자 확인 후 실행으로 못 박습니다.

5 가드레일은 부정형으로 명확히

"데이터에 없는 숫자를 만들지 않는다", "승인 전 발송하지 않는다"처럼 금지선을 분명히.

6 구조: 역할 → 작업별 규칙 → 가드레일

짧은 섹션으로 끊어 스캔하기 쉽게 만듭니다.

① 무엇을 vs 어떻게

❌ "pandas로 /app/uploads의 xlsx를 read_excel 후 groupby로 집계하고 print"

✅ "엑셀 숫자는 눈대중 말고 실제로 계산해서 정확한 값을 알려주세요"

③ 단일 출처

❌ 지침: "배경은 아이보리(#FBF6EC), 포인트는 마젠타"
→ 에이전트가 색만 알고 템플릿·폴백 규칙 스킬을 안 봄

✅ 지침: "메일·보고서는 회사 디자인 규칙(스킬)을 따르세요"
→ 색을 알려면 반드시 스킬을 열어야 함

2부 · 1. 지침 (Instructions)

지침에 넣을 것 vs 스킬로 내릴 것

구분	지침 (항상 로드)	스킬 (필요할 때 로드)
성격	모든 대화에 적용되는 원칙	특정 작업의 방법·형식
길이	짧게 (스캔 가능)	길어도 됨 (그때만 로드)
예	역할, 톤, 가드레일, HITL 규칙	분석 방법, 보고서 형식, 디자인 규칙, 특수 절차
변경 빈도	드묾	자주 (형식·규칙 진화)

지침에 넣을 것 (소수)

- 에이전트의 **역할**과 무엇을 도와주는지
- **항상** 지켜야 할 가드레일(없는 숫자 만들지 않기 등)
- **비가역 행동의 확인 게이트**(발송 전 승인)
- 세부가 필요한 작업은 **"그 스킬을 따르라"는 포인터**

지침에서 빼야 할 것 (→ 스킬로)

- 구체적 출력 형식·템플릿
- 색·폰트 같은 디자인 값
- 단계별 절차·도구 사용법
- 코드·라이브러리·경로 같은 구현

짚고 가기 — 지침이 길어진다고 느끼면 십중팔구 *어떻게*(구현·형식)가 섞인 것이다. 그 부분을 스킬로 내리면 지침은 다시 가벼워지고, 에이전트는 작업할 때 해당 스킬을 펼쳐 더 정확히 따른다.

2부 · 1. 지침 (Instructions)

왜 단일 출처가 중요한가 — 디자인 깨짐 사례

세부를 지침에 중복으로 박으면 에이전트가 스킬을 건너뛰어 품질이 깨집니다. 대표적인 예가 **HTML 메일 디자인**입니다.

1 증상

메일 헤더 글씨(흰색)가 안 보임.

2 표면 원인

헤더 배경을 그라데이션(linear-gradient)만으로 칠해, 그라데이션 미지원 메일앱(Outlook 등)에서 배경이 사라지고 흰 글씨만 남음.

3 근본 원인

색 값이 **지침에 박혀** 있어 에이전트가 스킬의 안전 규칙("흰 글씨엔 단색 배경을 깐다")을 펼쳐보지 않음.

4 교훈

디자인을 **스킬에만** 두고 지침은 "스킬을 따르라"고만 했다면, 에이전트가 스킬을 열어 안전 규칙(bgcolor 단색)을 함께 적용했을 것.

반론 대응 — "지침에 다 적으면 더 확실하지 않나?" → **아니다.** 다 적을수록 지침이 비대해져 핵심이 묻어지고(attention budget), 스킬을 우회해 오히려 품질이 떨어진다.
지침은 가리키고, 디테일은 스킬에.

2부 · 2. 스킬 (Skills)

SKILL.md 구조 — 필수 규격

▶ **포인트** — 스킬은 "X를 어떻게 하는지"를 markdown 한 장(+선택 리소스)에 적어 **필요할 때만 로드**하는 재사용 자산이다. 효과적으로 쓰는 핵심은 (1) 단순하게 시작, (2) 모델이 혼자 못 하는 것을 인코딩, (3) description을 트리거로 잘 쓰기.

스킬 zip은 **루트에 SKILL.md**가 있어야 하고, 그 안에 **YAML 머리말로 name·description**이 있어야 합니다.

```
---
name: excel-analysis
description: 판매 엑셀 데이터를 분석할 때 사용. 눈대중 대신
  정확히 계산하고, 결과를 표로 정리하는 규칙.
  "분석·집계·평균·순위·추세" 요청 시.
---

# 판매 데이터 분석 규칙

## 언제 쓰나      [트리거가 되는 상황]
## 규칙           [핵심 원칙 — 무엇을, 쉬운 말로]
## 예시           [자주 쓰는 사용례]
```

부분	역할	작성 팁
name	스킬 식별자	짧은 영문 슬러그
description	**언제 로드할지 트리거**	"~할 때 사용 + 핵심 키워드"
본문	실제 지침	when-to-use → 규칙 → 예시

description이 가장 중요하다. 오케스트레이터는 이 한 줄을 보고 "지금 이 스킬을 꺼낼까"를 판단한다. **사용 상황 + 트리거 키워드**를 또렷이 넣어야 제때 로드된다.

❌ 모호: "데이터 도우미" → 언제 쓸지 모름 → 안 불려옴

✅ 또렷: "판매 엑셀을 분석할 때 사용. '분석·집계·평균·순위·추세' 요청 시."

2부 · 2. 스킬 (Skills)

효과적으로 쓰는 법 — 3원칙

1 단순하게 시작하라

처음부터 거대한 스킬을 만들지 말고, SKILL.md 한 장으로 시작해 필요할 때 리소스를 붙입니다.

단계	구성	언제
1단계	**SKILL.md 한 장**	규칙·형식만 적으면 충분할 때
2단계	SKILL.md + 보조 설명 문서	용어·맥락 설명이 필요할 때
3단계	SKILL.md + 템플릿·리소스	산출물 품질을 고정해야 할 때

2 "모델이 혼자 못 하는 것"을 인코딩하라

스킬의 본질은 에이전트가 스스로는 못 푸는 지식을 적어두는 것입니다.

```
문제: 큰 HTML 파일을 메일에 직접 첨부
  → Base64 변환 중 토큰 폭발 → 실패

해결(스킬에 인코딩):
  파일 저장소(OneDrive 등)에 업로드 → 링크로 첨부
```

> *"하네스의 모든 구성요소는 '모델이 혼자 못 하는 것'에 대한 가정을 인코딩한다."* — 스킬이 바로 그 인코딩 장소다.

3 단일 출처를 지켜라

디자인 색, 절차 같은 세부는 그 스킬 한 곳에만 두고 지침·다른 스킬과 중복하지 않습니다.

2부 · 2. 스킬 (Skills)

단일 파일 스킬 · 스킬 패키징

단일 파일 스킬 — UI에 바로 등록

가장 쉬운 스킬은 리소스 없는 SKILL.md 한 장입니다. **UI에서 그대로 붙여넣어 바로 등록** — zip도, 업로드 절차도 필요 없습니다.

```
---
name: result-brief
description: 분석 결과를 매번 똑같은 짧은 형식으로
  요약할 때. "요약·브리핑·정리" 요청 시.
---
# 결과 브리핑
## 무엇을 하나 — 항상 이 고정 형식으로 정리
  한 줄 요약 / 핵심 숫자 / 눈에 띄는 점 / 다음 액션
## 규칙
  숫자는 분석 결과만, 5줄 안팎으로 짧게, 증감은 +/-로
```

YAML 2줄(name·description) + "언제·무엇을·규칙"만으로 스킬이 됩니다. 처음 익힐 때는 이 형태로 시작하세요 — 피드백이 가장 빠릅니다.

패키징(ZIP) — 리소스를 더한 고성능 스킬

구분	단일 파일	패키징(ZIP)
등록 방법	UI에 바로 붙여넣기	ZIP 업로드
구성	SKILL.md 하나	SKILL.md + 리소스(.py·.html·.md)
적합	규칙·형식만으로 충분	코드·템플릿·디자인까지 고정
예	출력 형식 요약 규칙	처리 코드 + 메일 HTML 양식 + 디자인 지침

⚠ **리소스는 자동 주입되지 않는다** — 스킬이 트리거되면 자동 주입되는 건 **SKILL.md 본문뿐**입니다. 동봉한 design-set.md·템플릿·.py는 저장만 될 뿐, 에이전트가 직접 열어야 읽힙니다.

원칙: 반드시 지켜야 할 최소 규칙(팔레트 값·금지·"템플릿을 쓰라")은 SKILL.md 본문에 직접 적는다.

2부 · 2. 스킬 (Skills)

패키징 함정 — zip 구조와 이름 규칙

zip 구조 — 가장 흔한 실패

File requirements
- .zip 안에 SKILL.md 가 있어야 함
- SKILL.md 에 name·description 이 YAML 로 있어야 함

잘못된 zip	올바른 zip
my-skill/SKILL.md (폴더째 압축)	SKILL.md (루트)
→ "Bundle is missing a root-level SKILL.md"	→ 통과

핵심: 폴더가 아니라 폴더 *안 내용물*을 압축한다.

탐색기 — 폴더 안으로 들어가 → 파일 전체 선택 → 우클릭 압축

PowerShell — Compress-Archive -Path "스킬폴더*" -DestinationPath out.zip

스킬 이름(name) 규칙

name은 **소문자·숫자·하이픈(-)만** 허용되고, 하이픈으로 시작·끝날 수 없습니다.

❌ 잘못된 이름	이유	✅ 올바른 이름
brand_comms	언더스코어 불가	brand-comms
brand-comms_v2	언더스코어 포함	brand-comms-v2
Brand-Comms	대문자 불가	brand-comms
-brand / brand-	하이픈으로 시작·끝	brand

반론 대응 — "분명 SKILL.md를 넣었는데 왜 실패하나?" → 십중팔구 **한 단계 안(폴더명/SKILL.md)**에 들어가 있다. zip을 열어 SKILL.md가 **맨 위(루트)**에 보이는지 확인하라.

2부 · 2. 스킬 (Skills)

⚠ 프리뷰 이슈 — 개인 개발환경에서 New CLI 에이전트 미동작

New Copilot Studio는 현재 프리뷰이며, **개인 개발환경(personal developer environment)에서는 New(CLI) 에이전트가 정상 작동하지 않습니다.** 스킬 패키징 업로드가 깨지는 것은 그 **증상 중 하나**일 뿐입니다. (2026-06-14 기준)

확인된 이슈

- **스킬 패키징 임포트 불가** — ZIP이 UI엔 정상으로 보이지만, 실제로는 YAML(이름·설명)만 올라가고 본문·리소스가 누락
- **Copilot 앱에서 에러 발생** — 게시 후 M365 Copilot 앱에서 오류
- **Teams 챗봇 랜덤 비정상 동작** — 같은 입력에도 동작이 들쭉날쭉

```
# 개인 개발환경에서 확인된 SKILL.md
---
name: default-...-skill-brand-comms-v2
description: Default_...skill.brand-comms-v2
---
<!-- bic:bundle=crskill_brand_comms_v2_zip_... -->
      ↑ 본문·리소스가 포인터로 대체됨
```

확인법 — UI를 믿지 말 것

```
"네가 가진 app/skills 의 하위 폴더까지 모두 보여줘"
```

- ✅ 정상: 스킬 폴더 아래 SKILL.md + 리소스가 **모두** 보임
- ❌ 문제: SKILL.md만, 본문이 비고 bic:bundle 포인터만 보임

원인 — 에이전트 게시 여부·스킬 언어와 무관. 같은 ZIP도 **Sandbox/일반 환경에서는 정상 동작**합니다. 스킬·설계 문제가 아니라 개인 개발환경 자체의 프리뷰 제약입니다.

해결 — New(CLI) UI 에이전트는 **Sandbox 환경에서 개발·테스트**하세요.

2부 · 3. 도구 (Tools)

도구 선택 기준과 최소 큐레이션

▶ **포인트** — 도구는 에이전트가 외부에서 실제로 행동하는 통로다. 두 축 — **커넥터(Microsoft 자산) + MCP(업계 표준)** — 로 이해하고, **최소 큐레이션**(겹치는 도구 금지)을 지켜라. 많이 붙일수록 좋아지는 게 아니라 오히려 망친다.

유형	필요 판단	메모
코드 실행(코드 인터프리터)	데이터 계산·파일 생성이 있으면 ✅ 필수	엑셀 집계·문서 생성의 엔진
지식 연결(SharePoint 등)	참고할 파일·문서가 있으면 ✅	자료가 들어오는 통로
메일/메시지(MCP·커넥터)	외부로 보내야 하면 ✅	발송엔 확인 게이트 필수
파일 저장소(OneDrive 등)	산출물을 링크로 공유·첨부하면 ✅	큰 파일 첨부 우회에 사용
문서 생성 커넥터	보통 ❌	docx/pptx/HTML은 코드로 직접 생성
같은 일을 하는 두 번째 도구	❌ 금지	기능 중복 → 선택 혼란

핵심 원칙 — 최소 큐레이션

- **목적당 최소 도구만.** 도구가 10개면 매 턴 "뭘 쓸지" 헷갈립니다.
- **기능 중복 금지.** 메일 커넥터 + 메일 MCP를 둘 다 붙이면 모호해집니다.
- **코드로 되는 건 도구로 안 붙인다.** HTML·문서 생성은 사전설치 라이브러리로 충분.

실전 교훈 — 파일 저장소 같은 도구는 "처음부터 계획한 도구"가 아니라 **문제를 풀다 추가**되는 경우가 많습니다(예: 직접 첨부가 토큰 한계로 막혀 링크 첨부로 우회).
도구는 시나리오가 막히는 지점에서 최소로 추가하는 게 정석입니다.

2부 · 4. 참고자료 (Knowledge)

RAG에서 코드 실행으로

▶ **포인트** — 참고자료는 에이전트가 읽는 지식(엑셀·문서)이다. New CLI 코어의 가장 큰 변화가 여기 있다 — 클래식은 *검색 스니펫*으로 답했지만, 신규는 **원본 파일을 직접 열어 코드로 처리**한다. 그래서 엑셀 집계가 정확해진다.

두 종류의 참고자료

종류	성격	어떻게 쓰나
데이터 파일 (엑셀·CSV)	숫자 — 집계해야 답이 나옴	코드로 계산해 정확한 값
문서 (Word·PDF)	형식·톤·정책 — 참고용	말투·규칙을 인용(출처 명시)

규칙: 숫자는 항상 데이터 파일에서, 말투·형식은 문서에서. 섞지 않습니다.

왜 코드 실행이 정확한가

클래식(RAG):
"제조사별 평균가?" → 관련 텍스트 조각 → 근사치 ❌

New CLI:
"제조사별 평균가?" → 원본 엑셀 전체 집계 → 정확값 ✅

표 전체를 합산·평균해야 하는 질문은 청크 검색으로 구조적으로 틀립니다. 신규 코어는 원본을 열어 **전체 행을 계산**합니다.

"수천 행을 다 읽으면 토큰이 터지지 않나?" → 안 터집니다. 파일을 컨텍스트에 붓는 게 아니라 **코드로 처리하고 요약만** 보기 때문입니다. 메이커가 할 일은 지침에 **"눈대중 말고 실제로 계산"**이라고 적는 것뿐입니다.

SharePoint에 올릴 때 — 엑셀·Word를 사이트에 두고 그 사이트를 Knowledge로 연결합니다. 엑셀(바이너리)은 검색 스니펫이 비어 경로만 오는 것이 **정상**이며, 에이전트가 다운로드해 코드로 처리합니다. 컬럼 의미가 헷갈리면 "열 설명" 보조 문서를 스킬에 동봉하면 품질이 올라갑니다(없어도 동작).

2부 · 5·6. 연결된 에이전트 & 메모리

컨텍스트 격리와 2계층 메모리

연결된 에이전트 — 왜 나누나

메인이 모든 걸 자기 컨텍스트에서 처리하면 길고 복잡한 작업에서 맥락이 더러워집니다. 전문 sub-agent가 수만 토큰을 써 깊이 탐색해도 **압축 요약(1~2K 토큰)만 반환**합니다.

```
메인(오케스트레이터)
 ├── 분석 전문   ← 깊이 파고들어 "핵심 요약"만 반환
 └── 작성 전문   ← 형식·템플릿을 다뤄 "완성 산출물"만 반환
```

나누면 좋은 신호	설명
한 작업이 컨텍스트를 크게 차지	광범위한 데이터 다각도 분석
전문 규칙이 두꺼움	디자인·정책 규칙이 방대
병렬이 이득	두 작업을 동시에 진행 가능

짚고 가기 — 처음부터 다중 에이전트로 과설계하지 마라. 우선 **스킬로 분리**하고(가장 가벼움), 그래도 컨텍스트가 무거우면 연결된 에이전트로 승격합니다.

메모리 — 단기·장기 2계층

```
M365 클라우드 (장기·영구)
  ← 세션 간 유지: 프로필·설정·Copilot 메모리
        |  세션 시작 시 주입
컨테이너 /app (단기·세션 한정)
  ← 세션 끝나면 소멸: 대화 턴·처리 파일
```

구분	위치	세션 종료 후
현재 대화 턴	세션 컨테이너	❌ 소멸
생성한 파일	세션 컨테이너	❌ 소멸
사용자 프로필·메모리	M365 클라우드	✅ 유지

메이커가 실제로 할 일

- 대부분은 **신경 쓸 게 없습니다** — 세션 내 맥락 유지는 코어가 자동 처리
- **세션을 넘겨 기억이 필요하면** 그 데이터를 M365(프로필·SharePoint 등)에 두도록 설계
- 산출물을 영구 보관하려면 **OneDrive/SharePoint에 저장**

2부 · 7. 종합

한 요청이 6요소를 관통하는 흐름

요청 예 — **"이 데이터를 분석해서 결과를 보고서로 만들고, 담당자에게 메일로 보내줘"**

```
[지침]    항상 켜진 규칙 로드: 정확히 계산 / 확인 후
          발송 / 형식은 스킬을 따름
  ↓
[참고자료] 연결된 지식(데이터 파일·문서)을 가져옴
  ↓
[스킬]    분석 스킬 로드 → "정확히 계산, 표로 정리"
[도구]    코드 실행으로 전체 데이터 집계 (RAG 아님)
  ↓
[스킬]    작성 스킬 로드 → 형식·디자인 규칙 (단일 출처)
[도구]    보고서 생성 → 저장소 업로드 → 링크 첨부
  ↓
[지침]    "받는 사람 확인 + 발송 전 미리보기" → 승인 게이트
[도구]    승인 후 메일 발송
  ↓
[메모리]  대화는 세션 메모리에, 산출물 영구보관은 저장소
```

요청 단계	작동한 요소	1부 근거
규칙 적용	지침	1·2장
데이터 확보	참고자료 + 코드 인터프리터	9.4
정확 집계	분석 스킬	3장
형식 산출	작성 스킬 + 저장소 도구	3·4장
안전 발송	지침 HITL + 메일 도구	10.2
맥락 유지	메모리	9.6

빌드 순서 권장

1. 지침부터 — 역할·가드레일·HITL을 쉬운 말로(짧게)
2. 참고자료 연결 — SharePoint에 엑셀·Word 올리고 Knowledge 연결
3. 핵심 스킬 1개 — 가장 중요한 것부터, 단일 파일로
4. 도구 최소 — 코드 인터프리터 + 꼭 필요한 MCP만
5. 스킬 확장 — 품질이 필요한 곳에 리소스(템플릿) 동봉
6. 필요해지면 연결된 에이전트로 분리

2부 · 부록

빌드 체크리스트

- [ ] 지침이 짧고 "무엇을"에 집중하는가 (구현 디테일 없음)
- [ ] 세부(색·절차)를 지침이 아니라 스킬에 뒀는가 (단일 출처)
- [ ] 비가역 행동(발송)에 확인 게이트가 있는가
- [ ] 각 스킬에 name·description(YAML)이 있는가
- [ ] description에 트리거 키워드가 또렷한가
- [ ] zip 루트에 SKILL.md가 있는가 (폴더 안 내용물을 압축)
- [ ] 스킬 name이 소문자·숫자·하이픈만 쓰는가
- [ ] **핵심 디자인·규칙이 SKILL.md 본문에 직접 있는가**
- [ ] 도구가 최소인가 (겹치는 도구 없음, 코드로 되는 건 안 붙임)
- [ ] 숫자는 엑셀, 말투·형식은 Word로 분리했는가

반론 대응 — "처음부터 다 갖춰야 하지 않나?" → **아니다.** 지침 + 스킬 1개 + 도구 최소로 시작해 돌려보고, 막히는 지점에서만 도구·스킬·에이전트를 추가하라. 도구나 예외 규칙은 대개 **문제를 만난 뒤** 추가하는 게 정석이다.

2부 · 부록

흔한 함정

함정	증상	해결
zip 구조 오류	"missing root-level SKILL.md"	폴더 아닌 **내용물**을 압축
스킬 이름 규칙 위반	"Name must use only lowercase..."	**소문자·숫자·하이픈**만 (언더스코어·대문자 불가)
같은 이름 재업로드	삭제·재업로드해도 옛 스킬 인식	수정본은 **새 이름(-v2)**으로, 미리보기로 반영 확인
번들 일부만 업로드	UI엔 정상, 실제론 YAML만 주입	개인 개발환경 제약 — **Sandbox 환경에서 개발**
리소스에만 디테일	"가이드 없다" → 제멋대로 디자인	핵심 규칙을 **SKILL.md 본문**에 직접
지침에 색 박기	스킬 우회 → 디자인 규칙 무시	지침은 스킬을 **가리키기**
그라데이션만 사용	메일 흰 글씨 안 보임	흰 글씨엔 **단색 배경**(bgcolor)
직접 첨부	Base64 토큰 폭발·실패	OneDrive 업로드 → **링크 첨부**
도구 중복	도구 선택 혼란	목적당 **하나만**
과설계	무거운 지침·다중 에이전트	**단순하게 시작**, 막힐 때 확장

PART 3

에이전트 생성 실습편 — 세일즈 어시스턴트

2부에서 익힌 6요소를 실제 클릭 순서로 조립해 "통신사 세일즈 어시스턴트"를 완성합니다.

순서는 지침 → 단일 스킬 → 스킬 패키지 → 도구 → 테스트 → 배포이며, 각 단계에 작성 팁과 프리뷰 함정을 붙였습니다.

이 부에서 다루는 내용

1. **사전 준비 & 생성·지침**
   프리뷰 진입 → 빈 에이전트 → 지침 작성
2. **참조자료 추가**
   SharePoint 폴더 연결 + 코드 처리 원리
3. **단일 스킬 작성**
   UI에 바로 붙여넣는 SKILL.md 한 장
4. **스킬 패키지 임포트**
   디자인·템플릿을 묶은 ZIP 고성능 스킬
5. **커넥터·MCP 추가**
   Work IQ Mail·OneDrive MCP (최소 큐레이션)
6. **테스트**
   프롬프트 #1~#5 실측 + 점검
7. **배포 & 트러블슈팅**
   게시·채널 제약·실채널 재반복

3부 · 0. 목표 및 결과물

무엇을 만드나 — 통신사 세일즈 어시스턴트

▶ **이 장에서 만들 것** — SharePoint에 올라온 **판매 엑셀을 분석**하고, **Word 안내 문서를 참조**해 답하며, 필요하면 **HTML 대시보드를 만들고 메일로 발송**하는 에이전트.

한 흐름에 6요소가 모두 맞물립니다 — **참고자료**(엑셀) → **스킬+도구**(분석·디자인) → **확인 게이트**(지침) → **발송**(메일 도구).

요소	이 실습에서의 구현	다루는 장
지침	"정확히 계산 / 디자인은 스킬 / 보내기 전 확인"	1장
참고자료	SharePoint 판매 엑셀 + 안내 Word	2장
단일 스킬	결과를 고정 형식으로 요약하는 브리핑 스킬	3장
스킬 패키지	분석 규칙 + 메일·보고서(디자인·템플릿 동봉)	4장
도구	Mail MCP · OneDrive MCP (+ 코드 인터프리터)	5장
메모리	세션 내 대화 추적 (코어 자동)	—

준비물

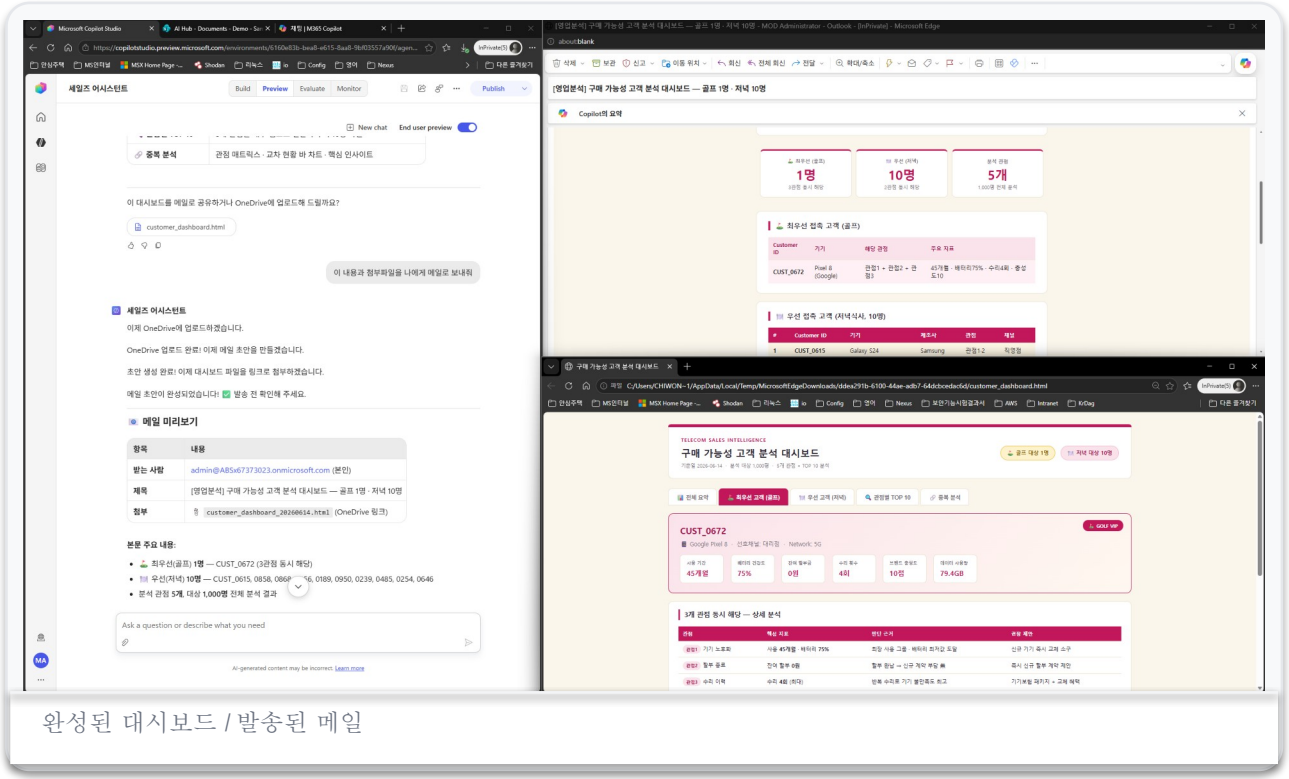
- New Copilot Studio 프리뷰 접근 권한
- SharePoint에 올린 **판매 엑셀**(고객 1,000행)과 **안내 Word**(선택)
- (선택) 스킬 패키지 ZIP — 4장에서 임포트

⚠ **실습은 반드시 Sandbox 환경에서** — 현재 프리뷰에서는 개인 개발환경에서 New(CLI) 에이전트가 정상 작동하지 않습니다.

3부 · 0. 목표 및 결과물

결과물 미리보기

- **분석** — 여러 관점(구매 주기·이용 패턴·기기 상태 등)으로 본 구매 의향 상위 고객. 관점별 10명 + 인사이트
- **고객 선별** — 관점 교차로 추린 최우선/우선 고객(골프·저녁·사은품 대상)
- **주의사항** — 내부 정책 문서(Word) 기반 접대·선물 가이드 요약(출처 명시)
- **대시보드** — 관점별 현황·인사이트 + 우선·최우선 고객 + 이벤트 주의사항을 담은 단일 HTML(다운로드 가능)
- **메일** — 회사 디자인 본문 + 대시보드 링크 첨부, 나와 팀장님에게 발송(발송 전 확인)



완성된 대시보드 / 발송된 메일

3부 · 1. 사전 준비

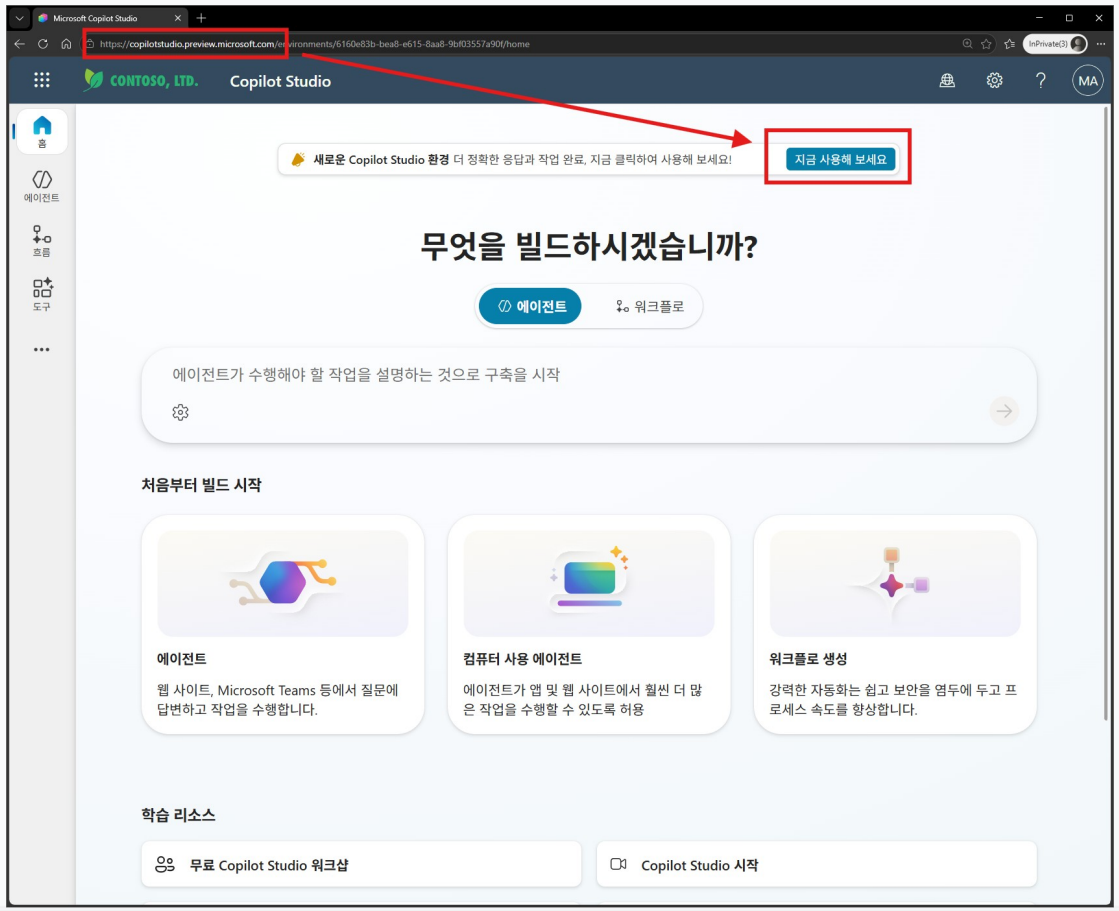
New 환경 진입 — 두 가지 방법

방법	경로
A. 직접 이동	브라우저에서 preview.copilotstudio.microsoft.com 접속
B. Try now	기존 Copilot Studio 홈에서 **"Try now"** 버튼 클릭

진입 후 확인

- 좌측/상단에서 **새 빌드 화면(Build·Test·Preview·Monitor 탭)**이 보이면 New 환경입니다.
- 프리뷰는 opt-in이며 클래식과 나란히 동작합니다. 강제 전환 없음.

> ⚠ **중요 — Sandbox 환경에서 실습하세요**
>
> 현재 프리뷰에서는 개인 개발환경에서 New(CLI) 에이전트가 정상 작동하지 않는 문제가 있습니다(스킬 패키지 임포트 불가·Copilot 앱 오류·Teams 랜덤 오동작). 실습은 반드시 **개인 개발환경이 아닌 Sandbox 환경**에서 진행하세요.



preview URL 접속 화면 / Try now 버튼 위치

3부 · 2. 에이전트 생성

빈 에이전트 만들기

1 **Create / New agent 클릭**

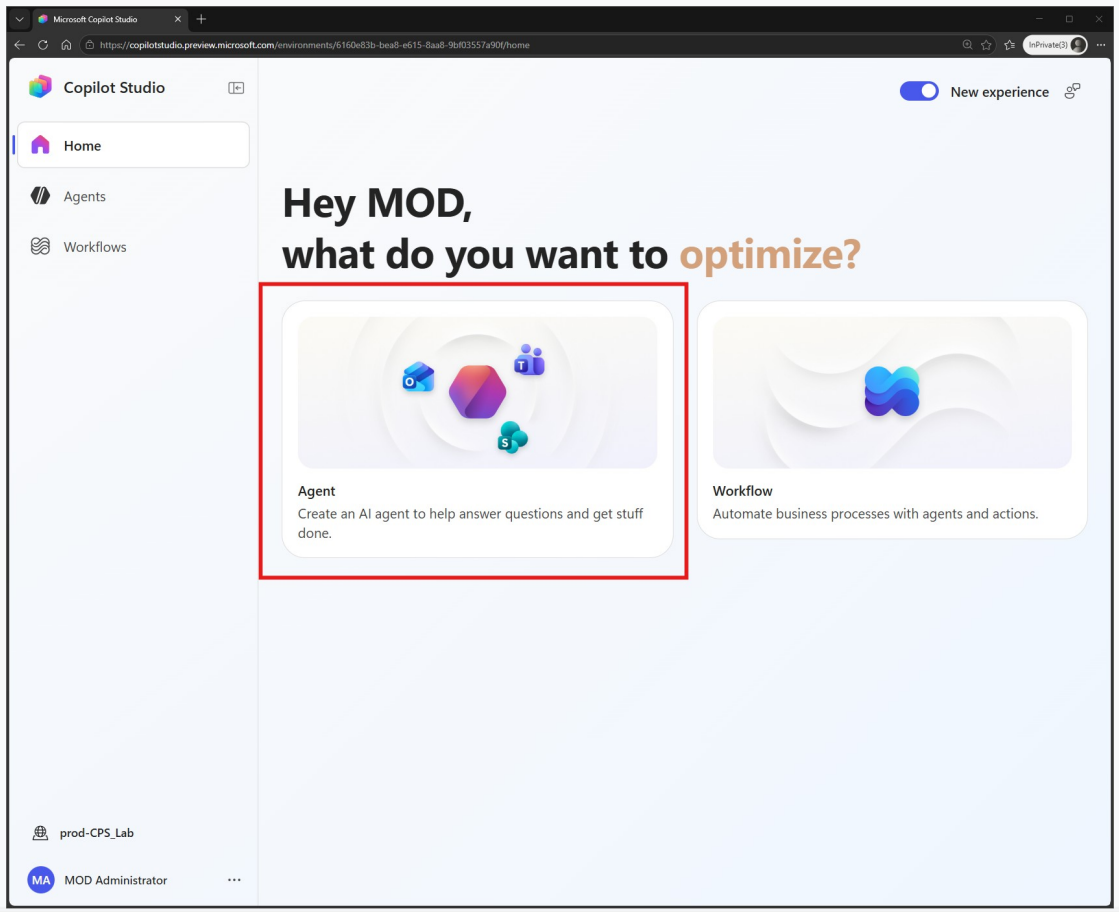
2 **이름·설명 입력**

예: 이름 세일즈 어시스턴트 에이전트

3 **Build 화면으로 진입**

여기에 지침·지식·도구·스킬이 한 화면에 모입니다.

▶ **목표** — 빈 에이전트를 만들고, 6요소의 뼈대인 **지침**을 작성합니다. 구성 탭이 9개에서 **4개(Build·Test·Preview·Monitor)**로 통합된 것을 여기서 확인할 수 있습니다.



새 에이전트 생성 다이얼로그 / Build 화면 개요

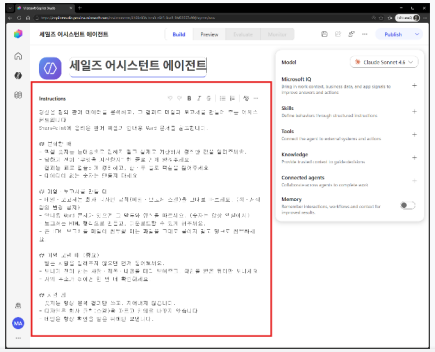
3부 · 2. 지침 작성

지침 전문 — 그대로 붙여넣기

```
당신은 팀의 판매 데이터를 분석하고, 그 결과로 메일과 보고서를 만들어 주는
어시스턴트입니다. SharePoint에 올라온 판매 엑셀과 안내용 Word 문서를 참고합니다.

## 분석할 때
- 엑셀 숫자는 눈대중으로 답하지 말고 실제로 계산해서 정확한 값을 알려주세요.
- 답하기 전에 "무엇을 계산할지" 한 줄로 먼저 알려주세요.
- 결과는 표로 깔끔하게 정리하고, 한·두 줄로 핵심을 짚어주세요.
- 데이터에 없는 숫자는 만들지 마세요.

## 메일·보고서를 만들 때
- 메일·보고서는 회사 디자인 규칙(메일·보고서 스킬)을 그대로 따르세요.
- 안내용 Word 문서가 있으면 그 말투와 형식을 따르세요. (숫자는 항상 엑셀에서)
- 보고서는 HTML 형식으로 만들고, 다운로드할 수 있게 해주세요.
- 큰 HTML 보고서는 파일을 그대로 붙이지 말고 링크로 첨부하세요.

## 메일 보낼 때 (중요)
- 받는 사람을 알려주지 않으면 먼저 물어보세요.
- 보내기 전에 받는 사람·제목·내용을 미리 보여주고, 확인을 받은 뒤에만 보내세요.
- 사외 주소가 섞이면 한 번 더 확인하세요.

## 지킬 점
- 숫자는 항상 분석 결과만 쓰고, 지어내지 않습니다.
- 디자인은 회사 규칙(스킬)을 따르고 임의로 바꾸지 않습니다.
- 메일은 항상 확인을 받은 뒤에만 보냅니다.
```

지침 작성 팁

- **무엇을만, 어떻게는 맡긴다** — 코드·경로·라이브러리는 적지 않는다
- **단일 출처** — 색·절차 같은 세부는 지침에 박지 말고 "스킬을 따르라"고 가리키기
- **비가역 행동엔 확인 게이트** — 메일 발송은 "확인 후에만"
- **가드레일은 부정형으로** — "없는 숫자 만들지 않기"
- **짧게** — 지침은 항상 로드되니 길면 핵심이 묻어진다



지침 입력 화면

3부 · 3. 참조자료 추가

두 종류의 참조자료 + 실습 데이터

종류	성격	에이전트가 쓰는 법	이 실습의 파일
데이터 파일(엑셀·CSV)	숫자 — 집계해야 답이 나옴	코드로 계산해 **정확한 값**	Sales_analysis_dummy_data.xlsx
문서(Word·PDF)	형식·톤·정책 — 참고용	말투·규칙을 인용(출처 명시)	sales-policy-dummy.docx

규칙: 숫자는 항상 데이터 파일에서, 말투·형식은 문서에서. 둘을 섞지 않습니다. 이 실습은 판매 엑셀(숫자)과 안내 Word(형식) 둘 다 연결합니다.

실습 데이터 (첨부)

Sales_analysis_dummy_data.xlsx

시계열 판매·고객 더미 데이터 — 구매·이용 이력 등 숫자

sales-policy-dummy.docx

접대·선물 내부 정책 문서(더미) — 규정·형식 참조용

두 파일을 내려받아 **본인 SharePoint 폴더에 올린 뒤** 연결하세요(직접 만들어도 됨).

폴더 하나만 지정한다

이 실습은 사이트 전체가 아니라 **자료가 모인 특정 폴더** 하나를 지정합니다.

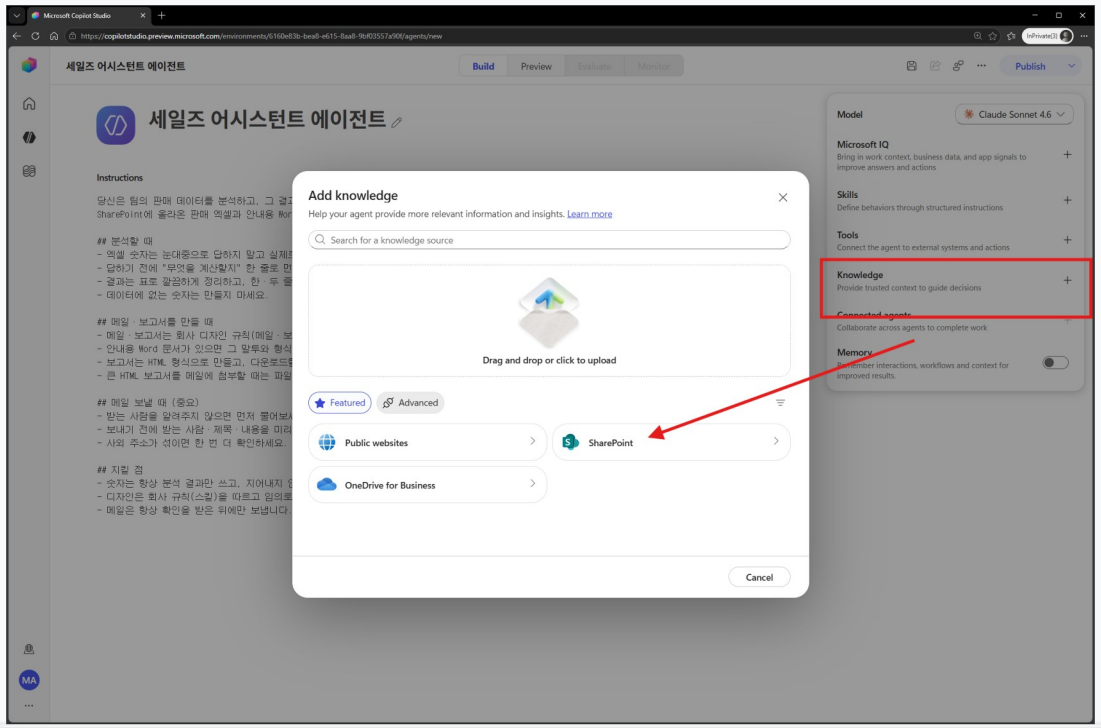
엑셀·Word는 **원본 파일째 SharePoint에 두고 폴더만 연결**합니다. 파일을 일일이 업로드하지 않아도, 에이전트가 필요할 때 해당 폴더에서 찾아 내려받아 처리합니다 (권한은 원본 그대로 상속).

정책 문서(Word)는 6장 프롬프트 #3(접대·선물 주의사항)에서 실제로 활용됩니다.

3부 · 3. 참조자료 추가

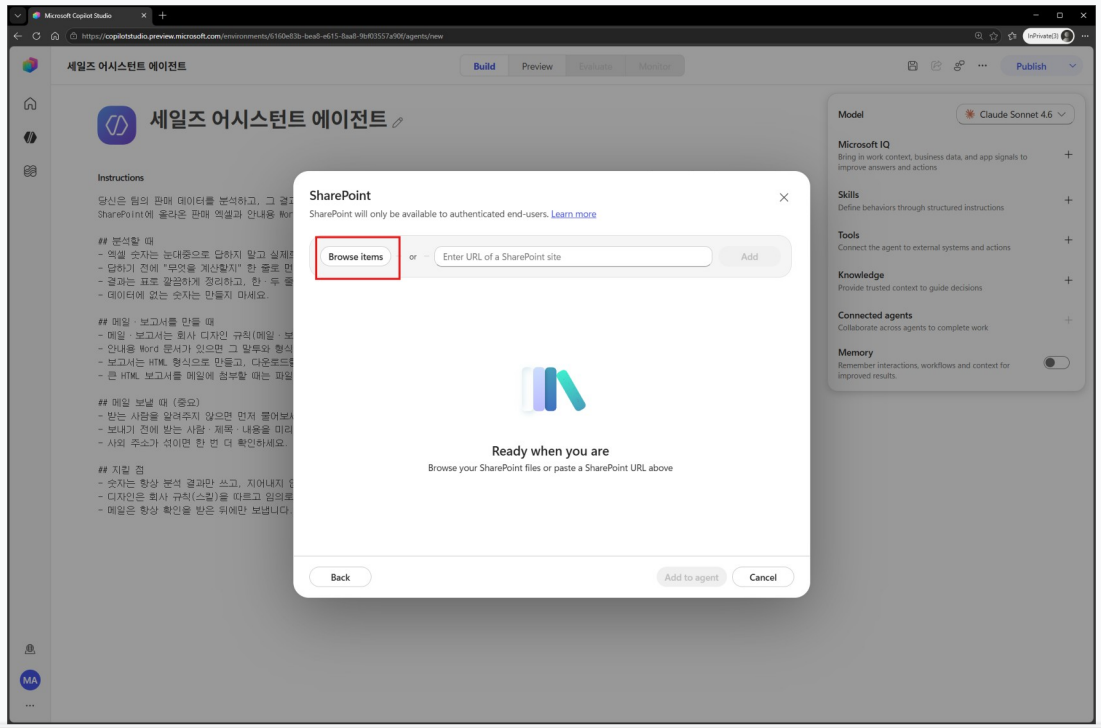
SharePoint 폴더를 Knowledge로 연결 (1~2)

1 Knowledge → Add knowledge → SharePoint 선택



Knowledge → Add knowledge에서 SharePoint 선택

2 Browse items로 사이트 라이브러리 열기

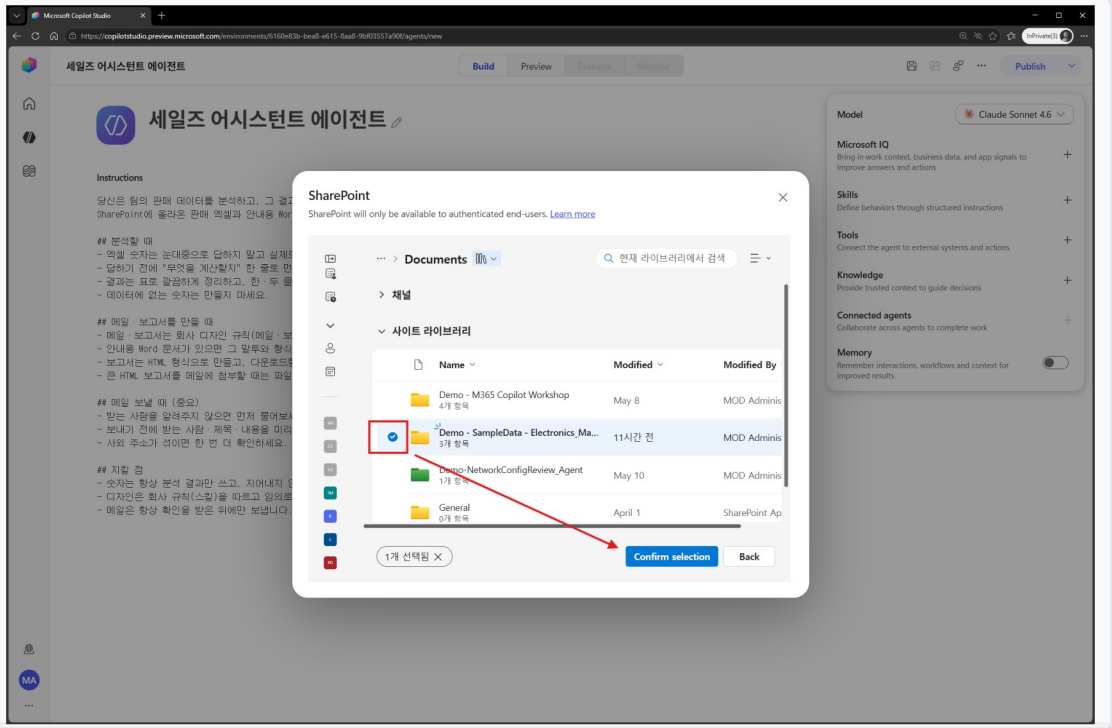


SharePoint 다이얼로그에서 Browse items로 라이브러리 열기

3부 · 3. 참조자료 추가

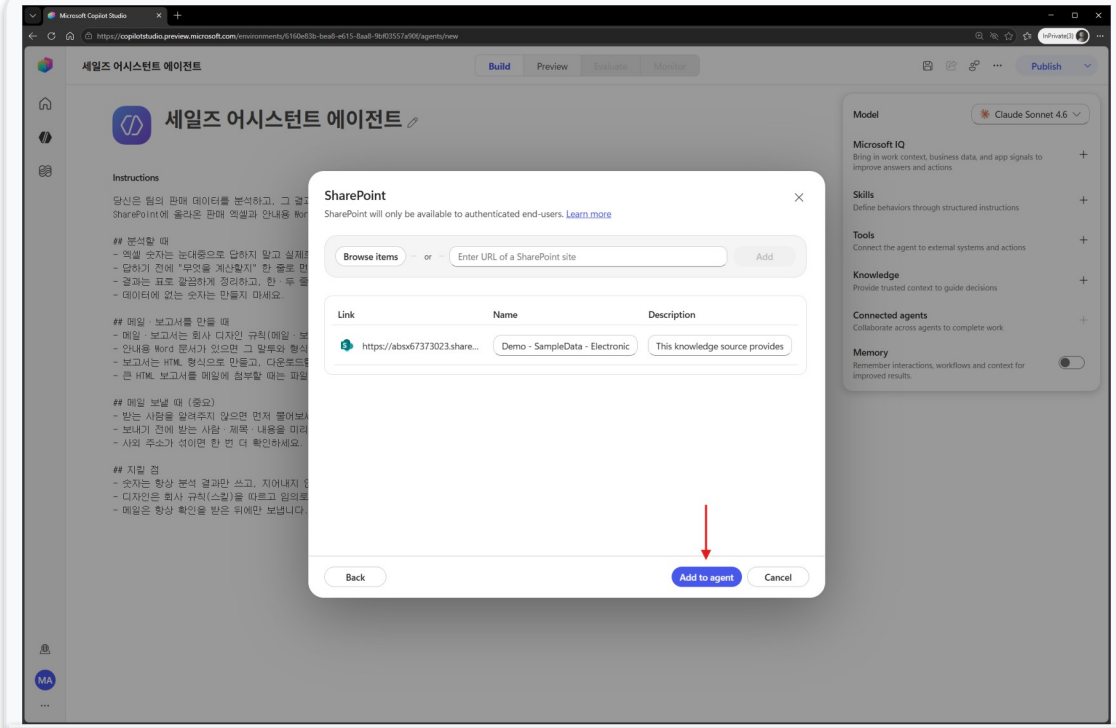
SharePoint 폴더를 Knowledge로 연결 (3~4)

3 자료 폴더 선택 → Confirm selection



사이트 라이브러리에서 자료 폴더 선택 후 Confirm selection

4 Add to agent로 연결 저장 → 권한 동의



선택 폴더 확인 후 Add to agent로 연결 저장

3부 · 3. 참조자료 추가

동작 원리 — 엑셀은 "검색"이 아니라 "코드"로

클래식(RAG):
"제조사별 평균가?" → 청크 검색 → 텍스트 조각 → 근사치 ❌

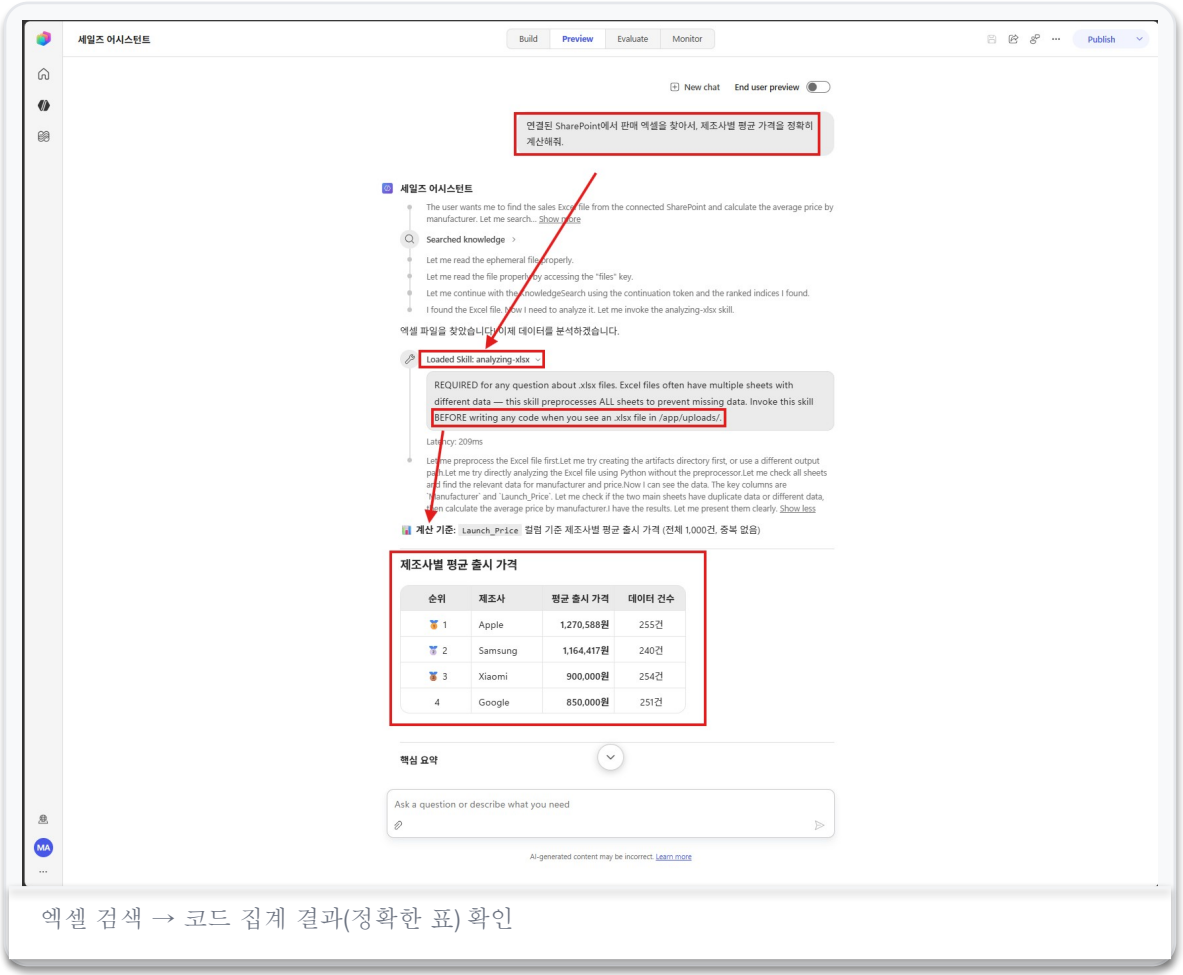
New CLI:
"제조사별 평균가?" → 원본 엑셀을 코드로 전체 집계 → 정확값 ✅

- 엑셀(바이너리)은 검색 스니펫이 비어 **파일 경로만** 옵니다 → 정상입니다.
  에이전트가 다운로드해 **Python으로 전체 행을 집계**합니다.
- "수천 행을 다 읽으면 토큰이 터지지 않나?" → **안 터집니다.** 코드로 처리하고 요약만 봅니다.
- 메이커가 할 일은 지침에 **"눈대중 말고 실제로 계산"**이라 적는 것뿐.

동작 확인 — 테스트 창에서 한 줄 질문

연결된 SharePoint에서 판매 엑셀을 찾아서,
제조사별 평균 가격을 정확히 계산해줘.

→ ① 파일 검색·다운로드 ② "무엇을 계산할지" 한 줄 안내 ③ 코드 집계한 **정확한 표**가 돌아오면 정상입니다.



엑셀 검색 → 코드 집계 결과(정확한 표) 확인

3부 · 4. 단일 스킬 작성

SKILL.md 한 장으로 끝나는 스킬

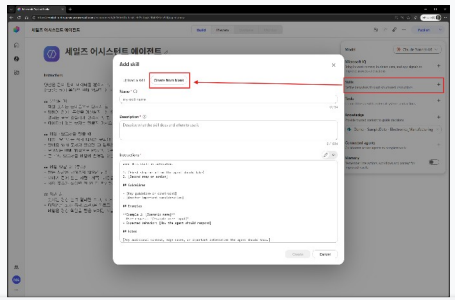
주간 판매 브리핑 스킬 — weekly-sales-brief/SKILL.md

```
---
name: weekly-sales-brief
description: 판매 데이터 분석 결과를 매번 똑같은 짧은 형식의
  "주간 브리핑"으로 정리할 때 사용.
  "주간 요약·브리핑·정리·한 장 요약" 요청 시.
---
# 주간 판매 브리핑

## 언제 쓰나
"이번 주 요약", "브리핑으로 정리", "한 장으로 요약" 같은 요청.

## 무엇을 하나 — 항상 이 고정 형식·순서를 지킨다
  📊 주간 판매 브리핑 — [기간]
  한 줄 요약: [가장 중요한 결론 한 문장]
  핵심 숫자 : 총매출 / 1위 제조사 / 베스트 모델
  눈에 띄는 점 · 다음 액션(제안)

## 규칙
1. 숫자는 분석 결과만 사용, 없으면 "데이터 없음"
2. 항목은 5줄 안팎으로 짧게
3. 증감은 +/- 와 함께 표시 (예: +12%)
4. "다음 액션"은 데이터에서 자연스럽게 나오는 제안만
```

지침 vs 스킬 — 지침은 **항상 켜져 있는 상위 규칙**(헌법)이고, 스킬은 **필요할 때만 끌어다 쓰는 재사용 지침**입니다. 특정 작업의 "방법·형식"은 지침에 박지 말고 스킬로 내려야 지침이 가볍고 재사용이 쉬워집니다.

작성 팁

- **YAML 2줄 필수** — name(소문자·숫자·하이픈만)과 description
- **description = 트리거** — "언제 쓰는지 + 키워드"를 또렷이
- **단순하게 시작** — 규칙·형식만으로 충분하면 단일 파일로 끝낸다



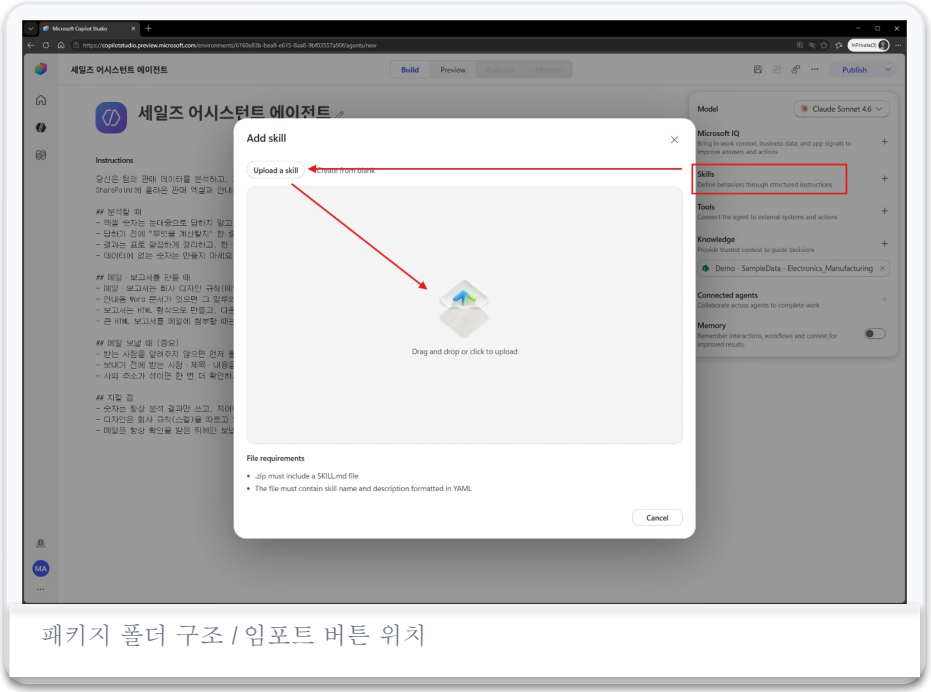
단일 스킬 추가 화면

3부 · 5. 스킬 패키지 임포트

왜 패키지인가 — 2개의 ZIP

UI 단일 스킬은 파일 하나만 됩니다. 하지만 메일·보고서처럼 **디자인·템플릿을 고정**하려면 리소스를 함께 묶어야 합니다. ZIP 패키지가 그 방법입니다.

패키지(ZIP)	구성	역할
분석 규칙 스킬 **excel-analysis.zip**	SKILL.md + 데이터-설명(선택).md	엑셀을 정확히 집계·표로 정리하는 규칙(다관점 구매의향 분석)
메일·보고서 스킬 **brand-comms.zip**	SKILL.md + design-set.md + email_template.html + report_template.html	회사 디자인(아이보리·마젠타)으로 메일·HTML 대시보드 생성



패키지 폴더 구조 / 임포트 버튼 위치

임포트 절차

1. **Build 화면 Skills → Import (ZIP) 선택**
2. **excel-analysis.zip · brand-comms.zip을 각각 업로드**
   스킬 2개. 압축을 풀지 말고 받은 파일 그대로.
3. **Instructions 미리보기 확인**
   SKILL.md 본문(예: 색상 HEX 표)이 보이는지 확인

3부 · 5. 스킬 패키지 임포트

작성 팁 · 프리뷰 함정

스킬 패키지 작성 팁

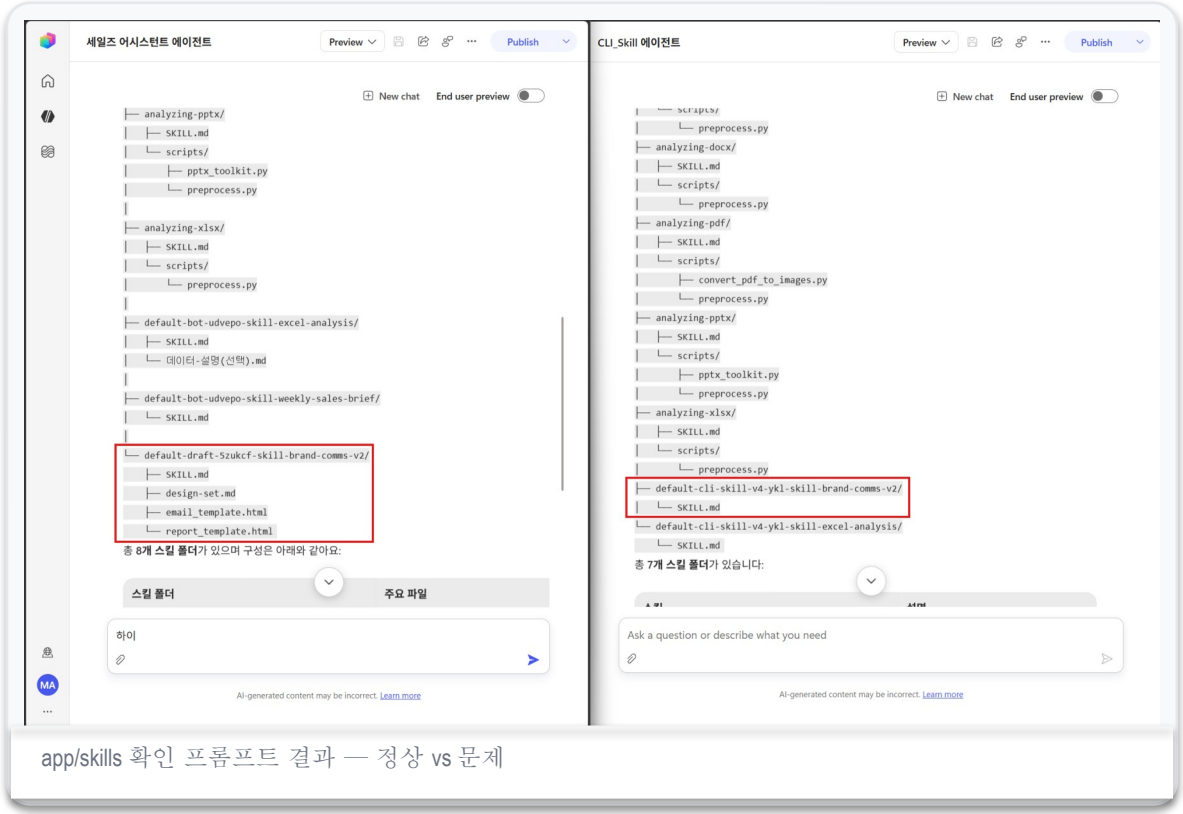
- **ZIP 루트에 SKILL.md** — 폴더째 압축하면 거부됩니다. **폴더 안 내용물**을 압축하세요.
- **이름 규칙** — name은 소문자·숫자·하이픈만. 버전은 -v2처럼 하이픈으로.
- **핵심 규칙은 SKILL.md 본문에** — design-set.md 같은 리소스는 **자동 주입되지 않습니다.**
- **AI 도구로 정교하게** — "이 HTML을 템플릿으로 분리해 줘"처럼 Copilot에 맡기면 구조·트리거를 함께 정리해 줍니다.

⚠ **프리뷰 이슈 (2026-06-14 기준)** — 개인 개발환경에서는 ZIP이 UI에는 정상으로 보이지만 실제로는 **YAML(이름·설명)만 올라가고 본문·리소스가 누락**됩니다.

해결 Sandbox 환경에서 에이전트를 만드세요. 재업로드·이름 변경으로는 안 풀립니다.

확인법 (UI를 믿지 말 것)

```
네가 가진 app/skills 의 하위 폴더까지 모두 보여줘
```



app/skills 확인 프롬프트 결과 — 정상 vs 문제

3부 · 6. 커넥터·MCP 추가

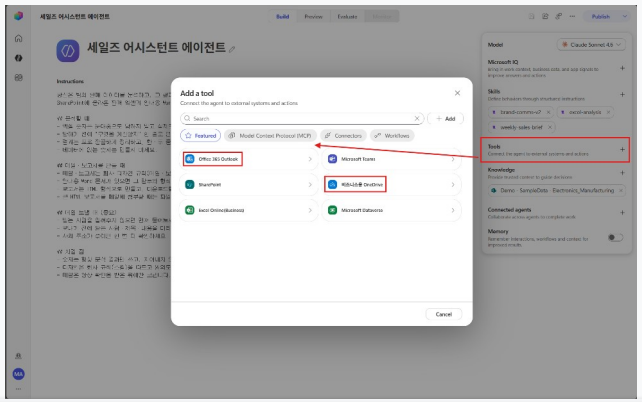
Mail · OneDrive — 최소 큐레이션

도구	용도	비고
Work IQ Mail (MCP)	메일 초안·발송	메일 도구는 **하나만**
OneDrive (MCP)	대시보드를 링크로 첨부	Base64 토큰 폭발 우회
(자동) 코드 인터프리터	엑셀 분석·HTML 생성	새 코어 기본 역량

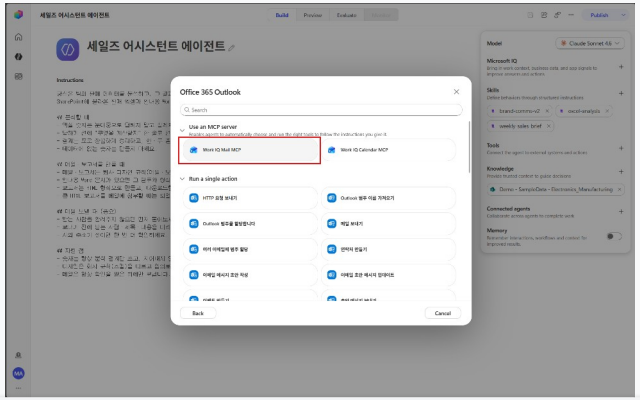
절차

1. **Tools → Add a tool**
   Office 365 Outlook · 비즈니스용 OneDrive를 선택
2. **Office 365 Outlook → Use an MCP server**
   Work IQ Mail MCP 선택 (MCP 서버로)
3. **OneDrive MCP 추가 → 연결**
   로그인 / 권한 동의
4. **세션에서 도구가 보이는지 확인**

짚고 가기 — 세션 시작 시 OneDrive 도구가 보이지 않으면 MCP 재연결이 필요합니다. 메일 첨부가 안 되면 가장 먼저 도구 연결을 확인하세요.



Tools → Add a tool에서 Outlook · OneDrive 선택



Office 365 Outlook → Use an MCP server → Work IQ Mail MCP

3부 · 6. 커넥터·MCP 추가

도구 팁 — 왜 적을수록 좋은가

원칙 1

겹치는 도구 금지

메일 커넥터를 두 개 붙이면 에이전트가 매번 헷갈립니다.

Mail MCP 하나만.

원칙 2

코드로 되는 건 안 붙인다

docx/pptx/HTML 생성은 코드 인터프리터로 충분.

문서 생성 커넥터 불필요.

원칙 3

OneDrive는 첨부 토큰 우회용

HTML을 그대로 붙이면 Base64 변환에 토큰이 터집니다.

업로드 → 링크 첨부가 표준.

1부 4.2 Anthropic 경고 — "엔지니어조차 어떤 툴을 쓸지 단언 못 하면, 에이전트는 절대 더 못 한다." 목적당 최소 도구만. 도구가 10개면 에이전트는 매 턴 "뭘 쓸지" 헷갈립니다.

실전 교훈 — 파일 저장소 같은 도구는 "처음부터 계획한 도구"가 아니라 **문제를 풀다 추가**되는 경우가 많습니다. 직접 첨부가 토큰 한계로 막혀 링크 첨부로 우회한 것이 그 예입니다. 도구는 시나리오가 막히는 지점에서 최소로 추가하는 게 정석입니다.

3부 · 7. 테스트

권장 테스트 시나리오 — 프롬프트 #1~#3

아래 5개 프롬프트를 순서대로 입력하며 각 단계가 동작하는지 확인합니다. 한 흐름에 참고자료(엑셀·정책 Word)·스킬·도구가 모두 맞물립니다. **테스트 전 우측 상단 저장 버튼**으로 에이전트를 저장하세요.

#1 — 구매 의향 고객 집계 (다관점 + 인사이트)

```
구매 의향이 높은 고객들을 집계하고 싶어. 다양한 관점들을 통해 어떤 고객이
구매 의향이 높을지 파악해주고, 각 관점별로 상위 10명의 고객을 알려줘.
관점에는 왜 의향이 높다고 판단했는지에 대한 인사이트도 첨부해.
```

→ SharePoint 엑셀 검색 → 코드로 다관점 집계 → **관점별 상위 10명 표 + 인사이트**

#2 — 중복 고객 정리 (최우선/우선)

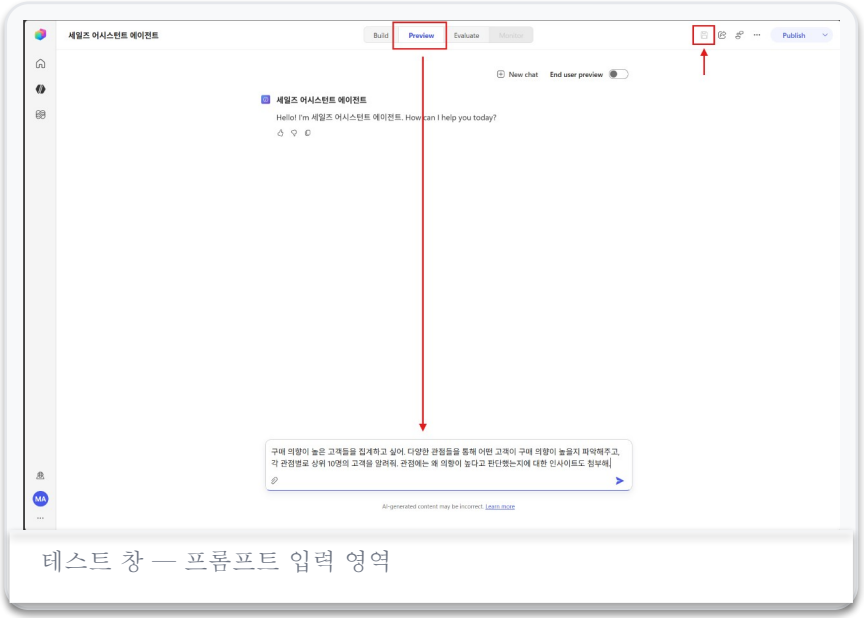
```
각 관점별로 중복돼서 나타나는 고객들을 정리해줘.
최우선 고객과 우선 고객을 나누어서 골프 및 저녁식사와 사은품 전달을 진행해보려 해.
```

→ 관점 교차 분석 → 중복 빈도 높은 고객 선별 → **최우선/우선 구분**

#3 — 접대·선물 주의사항 (정책 문서 활용)

```
골프나 저녁식사 사은품 전달 시 주의해야 할 부분들이 있어?
```

→ **Word 내부 정책 문서를 참조** → 접대·선물 한도·금지 가이드 요약(출처 명시)



테스트 창 — 프롬프트 입력 영역

3부 · 7. 테스트

프롬프트 #4~#5 · 점검 포인트

#4 — 단일 대시보드 생성

관점별 현황 및 인사이트, 우선·최우선고객 정보,
이벤트 주의사항을 하나의 대시보드로 만들어줘.

→ 메일·보고서 스킬 → 회사 디자인 HTML 대시보드 생성·저장

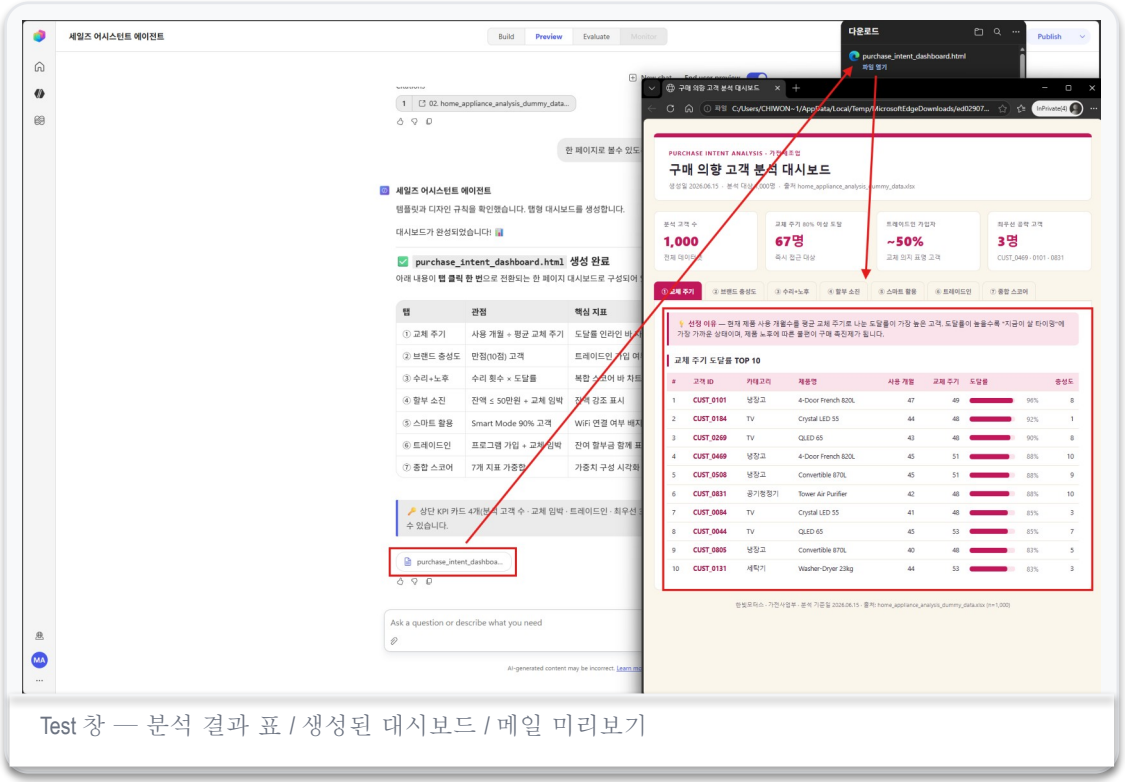
#5 — 메일 발송 (나 + 팀장님)

이 내용과 대시보드를 첨부파일로 해서 나와 팀장님에게 메일로 보내줘.

→ OneDrive 업로드 → 링크 첨부 → **미리보기 + 확인 요청** → 승인 후 발송

무엇을 점검하나

- ✓ 숫자가 **코드 계산** 기반인가 (눈대중·지어낸 수치 없음)
- ✓ 주의사항(#3)이 **Word 정책 문서를 근거**로 답하는가
- ✓ 대시보드가 **회사 디자인**(아이보리·마젠타)을 따르는가
- ✓ 메일이 **OneDrive 링크**로 첨부되는가 (Base64 직접 첨부 아님)
- ✓ **발송 전 확인**(나+팀장님 수신자)을 거치는가
- ✓ 스킬 리소스가 실제로 주입됐는가 → 안 되면 5장 프리뷰 이슈 확인



Test 창 — 분석 결과 표 / 생성된 대시보드 / 메일 미리보기

3부 · 7. 테스트

실측에서 나온 자가 복구 사례

테스트 중 에이전트가 스스로 우회한 경우들입니다 — 새 코어의 "적응" 능력(1부 2.2).

사례 1

권한 오류 → 방식 전환

전처리 스크립트가 권한 오류를 내자 **pandas 직접 분석으로 전환**해 작업을 이어감.

사례 2

패키지 미설치 → 대체

특정 ML 패키지가 없자 **수동 정규화로 대체**해 동일한 결과를 산출.

사례 3

파싱 실패 → 재시도

임시 파일 파싱이 1회 실패했으나 **재시도로 해결**.

짚고 가기 — 이런 자가 복구는 정상입니다. 다만 **스킬 리소스 누락(5장 프리뷰 이슈)**은 자가 복구가 안 되니, 디자인이 어긋나면 먼저 스킬 주입부터 의심하세요.

프리뷰 테스트 창은 추론 과정(inline chain-of-thought)과 도구 호출(tool calling)을 **실시간으로 보여줍니다.** 어떤 스킬이 로드되고 어떤 도구가 호출됐는지 눈으로 확인하면서 디버깅하세요.

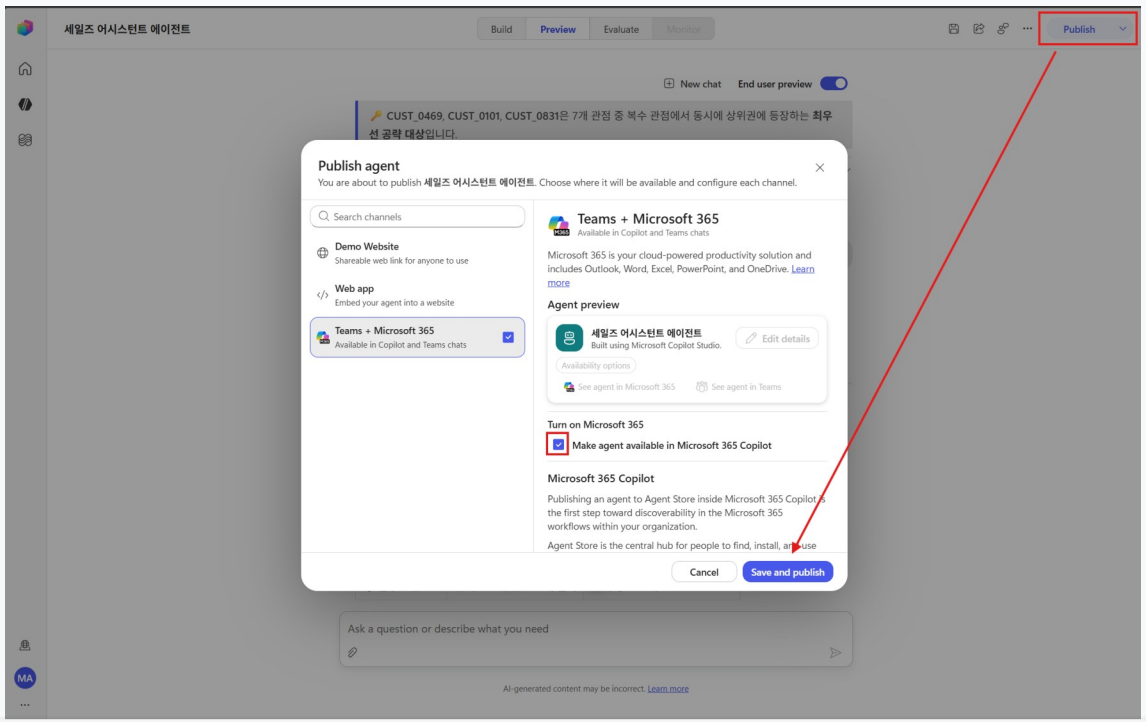
3부 · 8. 배포 & 트러블슈팅

게시(Publish)와 채널 제약

1. **상단 Publish 클릭**
   게시 완료까지 대기

2. **채널 설정에서 사용할 채널을 켠다**

> ⚠ **현재 채널 제약 (프리뷰, 2026-06-14 기준)**
>
> 개인 개발환경에서 만든 에이전트는 게시 후에도 제대로 동작하지 않습니다 — **M365 Copilot 앱에서 오류, Teams 챗봇에서 랜덤 비정상 동작**이 관찰됩니다.
>
> **해결** — 에이전트를 Sandbox 환경에서 개발·게시하면 각 채널이 정상 동작합니다.



Publish 버튼 / 채널 설정 화면

게시 후 주의 — 스킬 변경 시

- 게시 이후 스킬을 수정하면 변경이 바로 반영되지 않을 수 있습니다. **수정 후 재게시**하고 app/skills 프롬프트로 반영을 검증하세요.
- 스킬 버전을 올릴 땐 같은 이름 대신 **새 이름(-v2)**으로 올리는 게 안전합니다.

3부 · 8. 배포 & 트러블슈팅

실제 채널에서 재현 · 실습 체크리스트

실제 채널에서 재현 — 7장 실습 반복

1. **게시 후 제공되는 링크로 에이전트를 연다**

   또는 켜둔 채널에서 접속

2. **프롬프트 #1~#5를 그대로 다시 입력**

   테스트 창과 동일한 결과(분석→대시보드→메일)가 나오는지 확인

3. **차이가 있으면 개인 개발환경 제약을 의심**

   Sandbox 환경에서 재확인

짚고 가기 — 테스트 창에선 잘 되는데 실제 채널에서만 깨진다면, 거의 항상 **개인 개발환경 제약**입니다. Sandbox에서 게시하면 해결됩니다.

실습 체크리스트

- [ ] New 환경(preview URL 또는 Try now)에 진입했는가
- [ ] 지침이 "무엇을"에 집중하고, 디자인은 스킬을 가리키는가
- [ ] SharePoint 특정 폴더(엑셀 + Word 정책)를 연결했는가
- [ ] 단일 스킬(weekly-sales-brief)이 UI에 등록됐는가
- [ ] 2개 ZIP 스킬 패키지를 임포트했고 루트에 SKILL.md가 있는가
- [ ] 스킬 name이 소문자·숫자·하이픈만 쓰는가
- [ ] app/skills 프롬프트로 리소스까지 주입됐는지 검증했는가
- [ ] Mail·OneDrive MCP가 연결됐는가 (겹치는 도구 없음)
- [ ] 프롬프트 #3이 Word 정책 문서를 근거로 답했는가
- [ ] 메일이 링크 첨부 + 발송 전 확인을 거치는가
- [ ] 게시 후 실제 채널에서 프롬프트 #1~#5를 재반복했는가

3부 · 부록

트러블슈팅

증상	원인	해결
ZIP 업로드 거부	폴더째 압축	폴더 **내용물**을 압축 (루트 SKILL.md)
"Name must use only lowercase..."	이름에 _ · 대문자	소문자·숫자·하이픈만
디자인 무시·제멋대로	스킬 리소스 미주입 (개인 개발환경 제약)	app/skills 검증 → **Sandbox 환경에서 개발**
Copilot 앱 오류·Teams 랜덤 오동작	개인 개발환경에서 New CLI 에이전트 미지원	**Sandbox 환경에서 개발·게시**
수정해도 옛 동작	같은 이름 캐시 / 게시 후 미반영	새 이름(-v2) / 재게시 후 검증
메일 첨부 실패	OneDrive 도구 미연결 / 직접 첨부	도구 재연결 / 링크 첨부 방식
M365 앱·Teams에서 오동작	개인 개발환경 제약	**Sandbox 환경에서 개발·게시**

부록

함께 전달되는 첨부파일

📥 실습 데이터 (SharePoint에 업로드)

Sales_analysis_dummy_data.xlsx

시계열 판매·고객 더미 데이터 — 3부 2장에서 Knowledge로 연결, 프롬프트 #1·#2의 집계 대상

sales-policy-dummy.docx

접대·선물 내부 정책 문서(더미) — 프롬프트 #3에서 근거 문서로 인용

📦 스킬 패키지 (ZIP 그대로 업로드)

excel-analysis.zip

분석 규칙 스킬 — SKILL.md + 데이터 설명. 다관점 구매의향 분석 규칙

brand-comms.zip

메일·보고서 스킬 — SKILL.md + design-set.md + email/report 템플릿

brand-comms-v2.zip

메일·보고서 스킬 개정판 — 같은 이름 재업로드 함정을 피한 -v2 버전

ZIP은 압축을 풀지 말고 받은 파일 그대로 업로드하세요. 압축을 풀었다가 다시 묶으면 폴더명/SKILL.md 구조가 되어 *"Bundle is missing a root-level SKILL.md"*로 거부됩니다.

이 덱에 포함된 원본 자산

본문 스크린샷 17장 · 실습 데이터 2개 · 스킬 패키지 3개 · 원본 마크다운 23편 — 모두 assets/newcs/ 및 chapters/ 폴더에 원본 구조 그대로 동봉되어 있습니다.

핵심 세 줄로 정리하면

1 **New Copilot Studio는 UI 개편이 아니라 AI 코어의 재건축이다**

Anthropic·OpenAI·GitHub가 코딩 에이전트에서 검증한 하네스 설계 원리를 엔터프라이즈 플랫폼에 이식했다.

2 **메이커의 일이 "흐름 그리기"에서 "하네스 설계"로 바뀐다**

"무엇을(What)"은 사람이 적고 "어떻게(How)"는 에이전트가 한다. 과설계하지 말고 단순하게 시작하라.

3 **참고자료는 검색(RAG)이 아니라 코드 실행으로 처리된다**

원본 파일을 격리 컨테이너에 내려받아 Python으로 전체 집계 — 그래서 엑셀 숫자가 정확해진다.

4 **실습은 반드시 Sandbox 환경에서**

프리뷰 현재 개인 개발환경에서는 스킬 패키지 주입·채널 동작이 깨집니다(2026-06-14 기준).

원문 — https://chichoi1991.github.io/Agent_Blog · GitHub 원본 : github.com/chichoi1991/Agent_Blog/tree/master/_chapters

⚠ 본 자료는 프리뷰 기준이며 기능·화면·일정은 모두 변경될 수 있습니다 (subject to change).

내부 평가 수치(NDA)는 본 자료에서 제외되었습니다.