멀티 에이전트와 Auto-Healing

최종 프로젝트 적용 가이드 · 앞 과정에서 배운 것을 팀 서비스에 연결하기

AI 애플리케이션 개발 과정 · 최종 프로젝트 가이드

`Role · Tool · Contract` `max_steps` `retry → fallback` `incident log`

세 개념은 서로 다른 질문에 답한다

에이전트를 여러 개 만든다고 복구가 생기지 않고, 복구를 하려고 꼭 에이전트가 필요한 것도 아니다

멀티 에이전트

누가 어떤 순서로?

역할, Tool 권한, 입출력 계약이 서로 다른 에이전트가 정해진 순서로 일을 나눈다.

자기 성찰

답이 맞나?

응답은 왔지만 내용이 기준에 못 미칠 때, 평가하고 고쳐서 다시 낸다.

Auto-Healing

시스템이 돌아가나?

호출이나 서비스가 고장 났을 때, 감지하고 복구한 뒤 정상인지 확인한다.

자주 하는 오해 네 가지

오해	실제
프롬프트를 여러 번 호출하면 멀티 에이전트다	역할·권한·계약이 같다면 함수를 여러 번 부른 다단계 체인이다
LangGraph를 쓰면 멀티 에이전트다	프레임워크가 아니라 구조가 기준이다. 역할, 상태, 전이를 실제로 연결해야 한다
재시도를 붙이면 Auto-Healing이다	재시도는 복구 수단 하나다. 감지, 분류, 검증, 기록이 이어져야 한다
LLM에게 "다시 해 봐"라고 하면 복구된다	합격 기준은 코드가 검사하고, 복구 순서는 미리 정해 둔다

개념마다 대응하는 산출물이 있다

STEP 3과 STEP 4의 문서는 아래 질문에 대한 팀의 답을 적는 것이다

상태 전이 그래프 설계서
STEP 3
누가 어떤 순서로

→

자기 성찰 테스트 결과서
STEP 3
답이 맞나

→

협업 아키텍처 설계서
STEP 4
역할과 인계

→

보안 가드레일 설계 문서
STEP 4
해도 되는 일인가

→

Auto-Healing 구현 결과서
STEP 4
시스템이 돌아가나

평가는 문서가 있는지가 아니라 설계·구현·검증이 이어지는지를 본다. 같은 실행 ID와 같은 이름으로 문서를 잇는다.

에이전트, 도구, 오케스트레이터를 구분한다

구성 요소	하는 일	예
모델	판단과 생성에 쓰는 엔진	여러 역할이 같은 모델을 써도 된다
에이전트	맡은 역할에 따라 판단하고 결과를 만든다	요청 분석, 초안 작성, 근거 검토
도구 · API	정해진 입력으로 작업을 실행한다	DB 조회, 문서 검색, 예약 저장
오케스트레이터	순서, 상태, 전이, 상한을 통제한다	Python 함수, LangGraph
검증 코드	조건을 기계적으로 확인하고 차단한다	필수 필드, 권한, 합계

함수나 그래프 노드 하나가 곧 에이전트 하나는 아니다. 로그인, 파일 읽기, 합계 계산은 일반 코드로 충분하다.

에이전트를 나눠도 되는 네 가지 이유

이 중 하나로 설명되지 않으면 나누지 않는다. 에이전트가 늘면 비용, 지연, 오류 경로도 늘어난다

권한

Tool 권한이 다르다

조회만 하는 역할과 실행하는 역할을 나누면, 진단 담당이 복구를 실행할 수 없다.

병렬

동시에 할 수 있다

서로 독립인 조회를 동시에 실행하고 결과를 합친다.

검증

생성과 검증을 분리한다

글을 쓴 역할이 자기 글을 평가하지 않게 한다.

추적

책임을 추적해야 한다

누가 무엇을 넘겼는지 기록으로 남겨야 하는 업무다.

에이전트마다 네 가지를 정한다

협업 아키텍처 설계서의 역할 표가 이 네 칸으로 이루어진다

01

역할

무엇을 하는지와 함께, 무엇을 하지 않는지를 적는다.

02

입출력 계약

Pydantic 스키마로 정한다. 검증을 통과한 결과만 다음 역할로 넘긴다.

03

Tool 권한

역할마다 허용 Tool 목록을 둔다. 목록 밖의 호출은 코드가 거부한다.

04

종료 조건

최대 단계 수, 통과 조건, 실패했을 때의 행동을 정한다.

역할 카드와 계약을 코드로 적는다

- non_goals를 적어야 역할이 겹치지 않는다.
- allowed_tools 밖의 Tool은 호출할 수 없다.
- 근거가 없는 답은 계약에서 걸러진다.
- 앞 호출이 끝난 것과, 다음 역할이 쓸 결과가 준비된 것은 다르다.

agents.py (형식 예시)

```
from pydantic import BaseModel, Field

class DraftResult(BaseModel):
    answer: str = Field(min_length=1)
    evidence_ids: list[str] = Field(min_length=1)

WRITER = {
    "role": "초안 작성",
    "non_goals": ["품질 평가", "외부 시스템 변경"],
    "allowed_tools": frozenset({"search_docs"}),
    "output": DraftResult,
    "max_steps": 3,
}

def handoff(raw: dict) -> DraftResult:
    # 계약을 통과한 결과만 다음 역할로 넘긴다
    return DraftResult.model_validate(raw)
```

다섯 가지 협업 패턴에서 고른다

패턴	언제 쓰나	예
Sequential	앞 결과가 다음 역할의 입력일 때	분석 → 작성 → 검토
Parallel + Join	독립인 조회를 동시에 하고 합칠 때	날씨·장소·예산 → 일정
Router	요청 종류에 따라 한 곳으로 보낼 때	조회 / 위험 작업 / 일반 질문
Supervisor–Worker	관리자가 나누고 결과를 보며 다시 지시할 때	계획 → 위임 → 종합
Handoff	책임 자체를 다른 역할에 넘길 때	상담 → 전문 담당

LLM이 다음 역할을 고르는 구조라도, 허용되는 전이와 호출 상한과 종료는 코드가 통제한다.

일부만 실패했을 때의 정책을 미리 정한다

정책은 LLM이 즉석에서 정하지 않는다. 업무 요구사항으로 문서에 적어 둔다

Fail Fast

하나라도 실패하면 중단

모든 결과가 있어야 의미가 있는 업무에 쓴다. 결제, 예약 확정이 그렇다.

Best Effort

성공한 것으로 계속

일부가 빠져도 쓸모가 있을 때 쓴다. 빠진 정보는 결과에 표시한다.

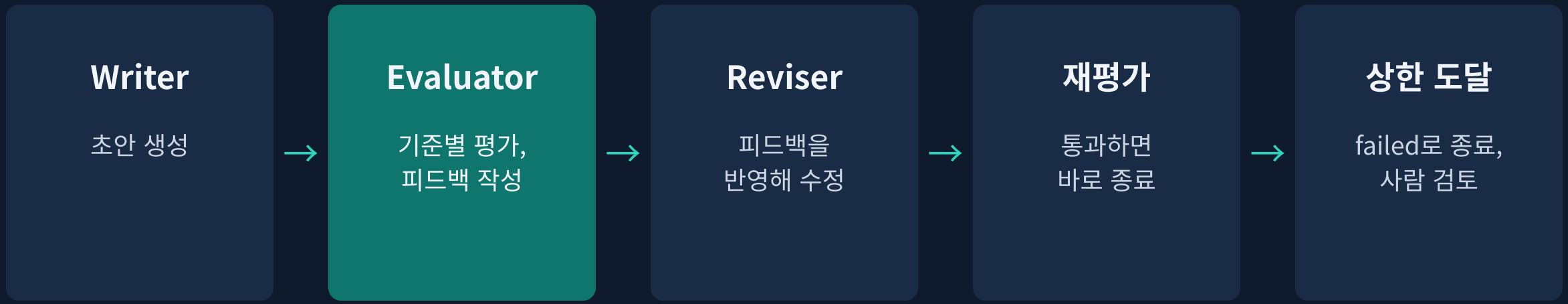
Required / Optional

필수만 성공하면 계속

역할마다 필수 여부를 정한다. 선택 결과 없이 진행한 것은 복구가 아니다.

자기 성찰은 생성과 평가를 분리한 루프다

Writer가 쓰고, Evaluator가 기준별로 평가하고, Reviser가 피드백을 반영한다



끝까지 통과하지 못하면 실패로 끝낸다. 통과하지 못한 결과를 성공인 것처럼 내보내지 않는다.

자기 성찰 루프의 네 가지 규칙

기준

평가 기준을 먼저 정한다

평균 점수 하나가 아니라, 어떤 기준에서 떨어졌는지 이름이 남게 한다.

검사

검사할 수 있는 것은 코드로

스키마, 근거 유무, 금지 표현은 Python이 확인한다. LLM 평가는 그다음이다.

피드백

피드백은 구체적으로

어느 부분이 왜 틀렸는지를 Reviser에게 넘긴다. "다시 해 봐"는 피드백이 아니다.

상한

반복 횟수를 코드로 제한

최대 횟수를 상수로 둔다. 끝나는 조건이 없는 루프는 버그다.

자기 성찰 테스트 결과서에 남길 것

항목	내용
평가 기준	기준의 이름과 합격 조건
시나리오	정상, 근거 누락, 잘못된 Tool 선택처럼 서로 다른 경우
라운드별 기록	초기 응답 → 평가와 피드백 → 수정 → 최종 판정
종료 사유	통과, 상한 도달, 치명 오류 중 무엇으로 끝났나
비용	반복 횟수, 토큰 사용량, 소요 시간

실행 로그나 추적 화면을 붙여, 표의 숫자가 실제 실행에서 나왔음을 보인다.

Auto-Healing은 여섯 단계의 운영 흐름이다

장애를 마법처럼 해결하는 기능이 아니다. 감지하고, 전략을 고르고, 결과를 확인하는 흐름이다

① 감지
timeout, 5xx, 빈 응답

→

② 분류
장애 유형 판단

→

③ 전략 선택
재시도, 대체, 차단, 이관

→

④ 실행
허용된 조치만 제한적으로

→

⑤ 검증
정상인지 다시 확인

→

⑥ 기록
전 과정을 로그로

검증에 실패하면 다음 전략으로 넘어가거나 사람에게 넘긴다. 복구 후 검증이 없으면 복구했다고 말할 수 없다.

분류가 먼저다: 모든 오류를 재시도하지 않는다

오류	다음 행동	이유
Timeout, 429, 5xx	재시도 후 대체 경로	일시적 오류다
401 인증 실패	즉시 중단	다시 해도 안 된다. 키와 설정을 확인한다
입력 누락	재계획, 추가 질문	입력이나 계획이 잘못됐다
권한 위반, Prompt Injection	차단	재시도는 정책을 반복해서 공격하는 셈이다
원인 불명	사람에게 이관	자동 복구 대상이 아니다

팀 서비스의 장애 유형표를 이 형식으로 먼저 만든다. Auto-Healing 구현 결과서의 첫 표가 된다.

복구는 정해진 순서로 한 계단씩 내려간다

어떤 순서로 무엇을 시도할지는 장애가 나기 전에 정해 둔다

① 재시도

같은 작업을
기다렸다 다시

→

② 제공자 전환

다른 LLM
제공자로

→

③ 대체 응답

캐시, 규칙,
검증된 템플릿

→

④ 사람 이관

실패를 기록하고
검토 요청

대체 응답을 썼으면 화면과 로그에 그렇게 표시한다. 정상 생성인 것처럼 보이게 하지 않는다.

복구 계단을 코드로 적는다

- 일시적 오류만 재시도한다. 다른 예외는 그대로 올라가 중단되거나 차단된다.
- 횟수와 대기 시간에 상한을 둔다.
- 어느 경로로 응답했는지 함께 돌려줘 화면에 표시한다.
- 모든 시도를 로그에 남긴다.

healing.py (형식 예시)

```
import time

class Transient(Exception):
    """timeout, 429, 5xx 같은 일시적 오류"""

def call_with_healing(providers, template, log, wait=0.5):
    for name, call in providers:        # primary, secondary
        for attempt in range(1, 4):     # 최대 3회
            try:
                result = call()
                log.append((name, attempt, "ok"))
                return result, name
            except Transient:
                log.append((name, attempt, "transient"))
                time.sleep(wait * 2 ** (attempt - 1))
    log.append(("template", 1, "fallback"))
    return template, "template"
```

무엇을 복구하는지 대상을 구분한다

복구 대상	예	복구 후 확인할 것
LLM · 도구 호출	타임아웃 재시도, 제공자 전환	계약, 근거, 업무 조건
작업 흐름	실패한 단계만 재실행, 저장된 상태에서 재개	성공 결과 보존, 중복 실행 방지
서비스 · 인프라	컨테이너 재시작, 트래픽 차단	Health와 실제 요청 처리

모든 팀이 서버 재시작을 만들 필요는 없다. 팀 서비스에서 실제로 일어날 실패를 고른다.
화면이 다시 연결된 것(SSE 재연결)은 화면의 복구다. 뒤에서 실패한 작업까지 복구된 것은 아니다.

위험한 작업은 자동으로 다시 실행하지 않는다

결제, 삭제, 재시작, 알림 발송처럼 되돌리기 어려운 작업에 적용한다

승인

사람이 먼저 승인한다

에이전트가 혼자 복구하지 않는다. 승인 ID가 없으면 실행이 거부되게 만든다.

중복 방지

이미 처리됐을 수 있다

응답이 끊겼어도 첫 요청은 처리됐을 수 있다. 다시 저장하기 전에 중복을 확인한다.

시뮬레이션

모의 실행부터 시작한다

처음에는 실제 재시작 대신 전략 선택까지만 실행한다. 모의 실행임을 결과에 밝힌다.

기록이 곧 증빙이다

- 복구에 성공했어도 장애가 있었다는 사실을 남긴다.
- 같은 실행 ID로 감지부터 검증까지 이어 볼 수 있게 한다.
- 대체 경로를 썼는지 표시한다.
- API Key, 비밀번호, 불필요한 개인정보는 기록하지 않는다.

incident 기록 (형식 예시)

```
{
  "incident_id": "inc-001",
  "detected_failure": "api_timeout",
  "selected_strategy": "retry_with_backoff",
  "retry_count": 1,
  "fallback_used": false,
  "recovery_status": "success",
  "validation_result": "health_check_passed",
  "duration_ms": 8200
}
```

복구율은 장애를 주입해서 숫자로 보인다

Auto-Healing 구현 결과서의 핵심은 재현할 수 있는 실험과 그 결과다

① 주입

일부러 고장 낸다

장애 유형마다 여러 건씩 주입한다. Mock 오류, 429 유발, 잘못된 입력을 쓴다.

② 측정

검증 통과만 센다

감지 시간과 복구 시간도 함께 잰다.

```
복구율 = 검증 통과 건수
        ÷ 자동 복구 대상 건수
```

③ 구분

섞어서 세지 않는다

정상 복구, 대체 응답, 사람 이관을 따로 적는다. 복구 실패 시나리오도 하나 이상 넣는다.

세 루프가 한 요청 안에서 만난다

요청 하나가 지나가는 길에 가드레일, 오케스트레이션, 자기 성찰이 차례로 놓인다

가드레일

입력 검증,
위반이면 차단

→

Orchestrator

순서와 전이
통제

→

Agents

역할별 실행,
계약 검증

→

Evaluator

품질 미달이면
수정 루프

→

응답

경로와 근거를
함께 표시

어느 단계에서든 호출이 실패하면 Auto-Healing이 분류, 복구, 검증, 기록을 맡는다. 복구하지 못하면 사람에게 넘긴다.

업무 유형별 적용 예시

업무	역할 분담	품질 개선	장애 복구
리서치 보고서	계획 → 근거 수집 → 작성·검토	근거 없는 주장을 수정	검색 실패 시 재시도, 다른 출처
상담·예약 지원	분류 → 규정 검토 → 예약안	규정·시간 위반을 수정	조회는 재시도, 저장은 중복 확인
장애 대응	증상 분석 → 조치 계획 → 검증	위험·롤백 누락을 수정	승인된 조치 후 Health 재검증
학습 자료·문항	자료 분석 → 생성 → 근거 검토	오답·보기 중복을 수정	제공자 전환, 검증 실패면 중단

특정 팀의 요구사항이 아닌 예시다. 팀의 사용자와 핵심 기능에 맞춰 바꿔 쓴다.

시연에서 보여 줄 다섯 장면

장면	보여 줄 것
정상 흐름	누가 어떤 입력을 받아 무엇을 다음 역할에 넘겼는가
내용 오류	검토가 무엇을 지적했고, 어떻게 수정하고 다시 평가했는가
복구 성공	장애 주입 → 감지 → 조치 → 재검증 → 종료를 같은 실행 ID로
복구 실패	상한을 넘으면 반복하지 않고 실패와 사람 이관으로 끝나는가
위험 경계	권한 부족이나 근거 누락을 성공으로 우회하지 않는가

개념 확인을 위한 수업용 점검 제안이다. 별도 배점이나 에이전트 개수 규정은 아니다.

적용 점검표

- [ ] 에이전트마다 역할·하지 않는 일·허용 Tool·입출력 스키마가 있다
- [ ] 장애 유형표 → 전략 → 검증 → 기록이 이어져 있다
- [ ] 에이전트를 나눈 이유를 권한·병렬·검증·추적으로 설명할 수 있다
- [ ] 401·권한 위반·Prompt Injection은 재시도하지 않는다
- [ ] 실행 순서와 종료 조건이 코드나 그래프에 있다
- [ ] 대체 응답을 쓰면 화면과 로그에 표시한다
- [ ] 자기 성찰에 평가 기준·최대 횟수·실패 처리가 있다
- [ ] 복구 실패 시나리오를 하나 이상 실험해 남겼다

지금 팀에서 할 일

넓게 벌이지 말고, 흐름 하나와 실패 하나를 끝까지 연결한다

① 핵심 흐름 하나

사용자 시나리오
하나 고르기

→

② 역할 표

책임, 입출력,
Tool, 종료 조건

→

③ 실패 하나

실제로 일어날
장애를 끝까지

→

④ 주입과 기록

일부러 고장 내고
로그로 증빙

앞 과정에서 실습한 제공자 전환, 평가·수정 루프, 계약 검증 코드를 다시 실행해 보고 팀 서비스에 맞게 결합한다.