마이크로서비스 아키텍처 · 2026 개정판

에이전트 시대의 마이크로서비스

바이브 코딩과 코딩 에이전트로 다시 쓰는 MSA



이즈잇소프트 채문창 · 2022 원본 → 2026 개정

2022 년 이후 , 무엇이 바뀌었나

2022 원본

사람이 코드를 쓴다

- 서비스 하나 = 개발팀 하나
- 병목은 구현 속도와 인력
- MSA 는 큰 조직의 전유물

2026 지금

에이전트가 코드를 쓴다

- 서비스 하나 = 사람 1 명 + 에이전트 N
- 병목은 명세 · 검증 · 운영
- 1 인 개발자도 MSA 를 만들 수 있다

MSA 의 원칙은 그대로 , 바뀐 것은 ' 누가 · 얼마나 빨리 ' 만드느냐입니다

오늘의 순서

2022 원본의 흐름을 유지하고 , 각 장을 에이전트 관점으로 개정했습니다

01

MSA 다시 보기

모놀리스 · 정의 · 장점 재평가

02

코딩 에이전트

바이브 코딩과 에이전트의 원리

03

에이전트로 설계

경계 찾기 · 계약 우선

04

에이전트로 구현

구현요소 6 가지 개정

05

문제점과 해결책

원본 8 가지 + 새 함정

06

팀 운영

사람 1 + 에이전트 N

07

실습

주문 시스템 3 개 서비스

08

정리 · 과제

2022 → 2026 한 장 요약

01

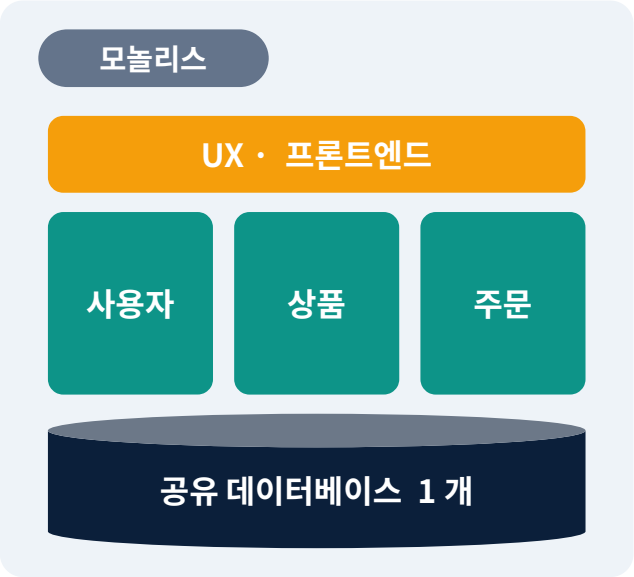
MSA 다시 보기

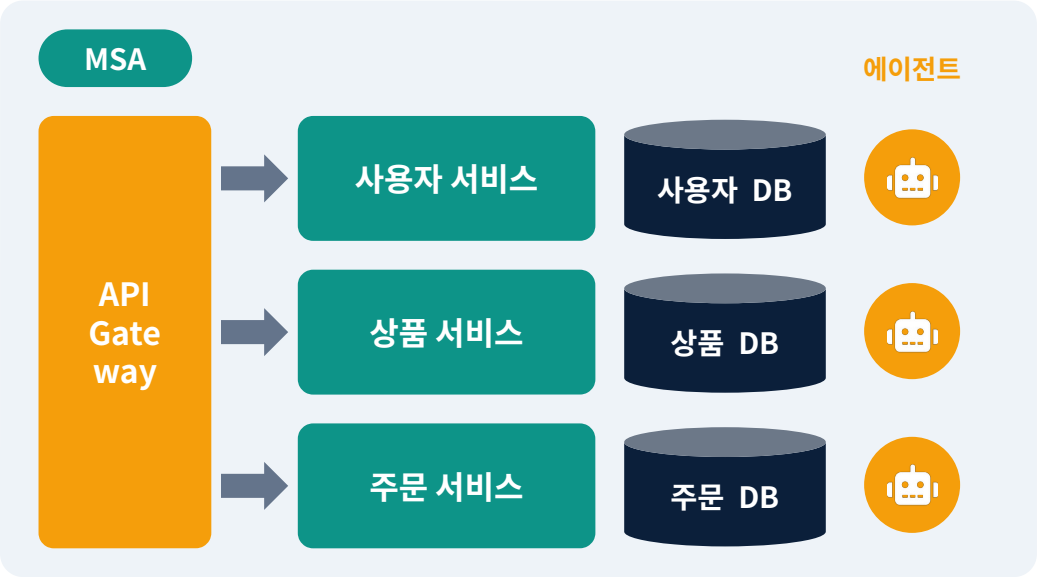
변하지 않은 원칙과 달라진 비용



모놀리스 vs MSA — 그리고 서비스마다 에이전트

원본의 사용자 · 상품 · 주문 예제를 그대로 가져왔습니다





서비스 경계 = 데이터 소유 경계 = 에이전트 한 세션이 맡는 작업 범위

MSA 정의 — 2022 원문과 2026 보완

원칙은 한 줄도 버리지 않았습니다 . 에이전트 시대에 맞는 해석만 덧붙였습니다

원칙	2022 원문	2026 보완
작은 서비스	작은 서비스의 결합으로 하나의 앱	에이전트 한 세션이 통째로 이해할 크기
독립 배포	완전 자동화된 개발 · 배포	에이전트 PR 을 CI 가 자동 검증 후 배포
중앙 관리 최소	최소한의 중앙 관리 체계	중앙엔 계약 (API 스키마) 과 규칙 파일만
폴리글랏	멀티 언어 · 멀티 저장소	언어 장벽은 낮아짐 , 리뷰 · 운영 부담은 그대로
독립 데이터	서비스별 데이터 저장소	변하지 않음 — 가장 중요한 원칙

MSA 장점 , 에이전트 시대에 다시 매기기

원본 8 쪽의 장점 목록을 세 가지로 다시 분류했습니다



강화

더 강해진 장점

작은 코드베이스 = 에이전트 정확도 상승 . 서비스별로 에이전트를 병렬로 돌려 개발 속도가 배로 늘어남 .



불변

그대로인 비용

네트워크 호출 지연 , 분산 데이터 정합성 , 배포 · 모니터링 대상 증가 . 에이전트가 줄여주지 않음 .



주의

새로 생긴 위험

만들기가 너무 쉬워 서비스를 남발 . 10 분 만에 ' 분산 모놀리스 ' 가 완성될 수 있음 .

그래서 처음부터 MSA? — 모놀리스 먼저

에이전트 덕분에 리팩터링 비용이 낮아져 '나중에 쪼개기'가 쉬워졌습니다

모듈러 모놀리스

한 저장소 · 한 배포, 폴더로 도메인 분리

경계 검증

모듈 간 호출은 공개 인터페이스로만

분리 신호 감지

부하 · 배포 주기 · 팀이 달라지는 모듈

필요한 것만 분리

에이전트가 모듈을 서비스로 추출

1 인 창업 · 초기 서비스는 모듈러 모놀리스로 충분한 경우가 대부분입니다 (Martin Fowler, MonolithFirst)

SoA · ESB 에서 에이전트 인터페이스까지

원본 10~11 쪽의 역사에 한 칸을 더했습니다

SOA · ESB

2000 년대 , 중앙 버스와 분산 트랜잭션 — 무거움

API Gateway

2010 년대 MSA, 입구 하나 · 서비스는 독립

서비스 메시

쿠버네티스 위 사이드카로 통신 · 보안 · 관측

MCP 인터페이스

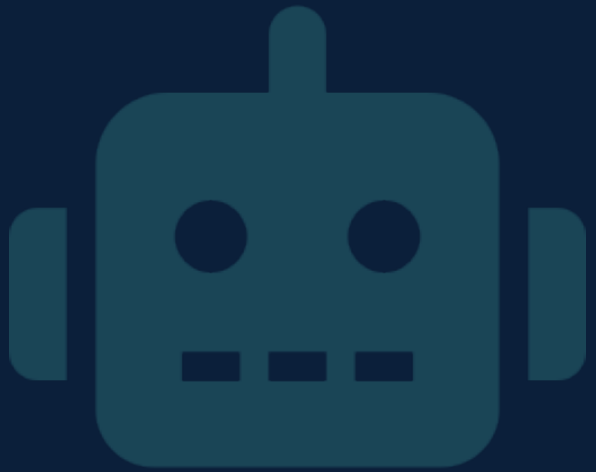
사람뿐 아니라 AI 에이전트가 서비스를 호출

교훈은 같습니다 : 똑똑한 중앙 (ESB) 이 아니라 ' 똑똑한 끝점 , 단순한 파이프 '

02

코딩 에이전트 이해

노드 안을 들여다보면 단순한 반복 루프



바이브 코딩 vs 에이전트 기반 엔지니어링

바이브 코딩



말로 만들고 결과만 본다

- 자연어로 요청 , 실행되면 수락
- 코드를 다 읽지 않아도 된다
- 프로토타입 · 1 회성 도구에 강함

VS

에이전트 엔지니어링



명세와 테스트로 지휘한다

- 계약 · 테스트 · 규칙 파일로 지시
- 변경 내용 (diff) 을 사람이 리뷰
- 운영되는 서비스에 필수

MSA 를 느낌으로 만들면 분산 모놀리스가 됩니다 — 프로토타입은 바이브로 , 서비스는 엔지니어링으로

코딩 에이전트 노드 안 들여다보기

에이전트 = LLM + 도구 (파일 · 셸 · git) + 끝날 때까지 도는 반복 루프



규칙 읽기

요청 + AGENTS.md · 명세



코드 탐색

관련 파일을 찾아 읽음



계획

수정할 파일과 순서 결정



수정 · 실행

코드 변경 , 테스트 실행



검증 · 반복

실패하면 원인 분석 후 재시도

그래서 테스트가 없으면 에이전트는 ' 성공 ' 을 판단할 근거가 없습니다 — 테스트가 곧 지시서

도구 지형 한눈에

도구 이름은 계속 바뀝니다 . 유형으로 기억하세요



터미널 에이전트

수업 실습

claude · codex · agy — 저장소에서 직접 읽고 , 실행하고 , 고침



IDE 에이전트

편집기

에디터 안에서 자동완성 · 대화 · 다중 파일 편집



원격 에이전트

비동기

이슈를 맡기면 클라우드에서 작업 후 PR 로 돌려줌



확장 표준

연결

MCP(도구 연결), AGENTS.md · 스킬 (규칙과 절차)

컨텍스트 엔지니어링 — 규칙 파일 배치

루트 AGENTS.md
전 서비스 공통 규칙 · 금지사항

서비스별 AGENTS.md
도메인 · 계약 위치 · 실행 명령

contracts/ 폴더
OpenAPI · 이벤트 스키마 = 단일 진실

docs/adr/
결정 이유를 남겨 에이전트도 따름

```
shop/
├─ AGENTS.md              # common rules
├─ CLAUDE.md              # -> see AGENTS.md
├─ contracts/
│  ├─ user.openapi.yaml
│  ├─ order.openapi.yaml
│  └─ events.asyncapi.yaml
├─ docs/adr/0001-db-per-service.md
└─ services/
   ├─ user/   AGENTS.md  app/  tests/
   ├─ product/AGENTS.md  app/  tests/
   └─ order/  AGENTS.md  app/  tests/
```

에이전트는 매 세션 기억이 없습니다 — 규칙 파일이 곧 에이전트의 장기 기억입니다

03

에이전트와 함께 설계하기

설계는 사람이 결정하고 , 에이전트는 초안과 검토를 돕는다



서비스 경계 찾기 — 에이전트와 이벤트 스토밍

BOUNDED CONTEXT



원본 20 쪽 개정

같은 ' 고객 ' 도 영업팀 · 배송팀 · 회계팀이 보는 모습이 다르다

1. **도메인 이야기 수집**
   요구사항 · 인터뷰 메모를 에이전트에게 제공

2. **도메인 이벤트 뽑기**
   ' 주문됨 ', ' 결제됨 ', ' 재고 부족 ' 같은 과거형 사건 목록 초안

3. **컨텍스트 맵 초안**
   이벤트를 묶어 후보 서비스와 소유 데이터를 제안받음

4. **사람이 결정 · ADR 기록**
   경계는 비즈니스 판단 — 결정 이유를 docs/adr 에 남김

프롬프트 예시 — 경계 초안 받기

역할과 산출물 형식 지정
표 · 목록 등 검토하기 쉬운 형태로

결정은 요청하지 않기
후보와 장단점만 , 선택은 사람이

반대 의견도 요청
' 모놀리스로 충분한가 ?' 를 꼭 묻기

결과는 ADR 초안으로
합의한 내용만 저장소에 커밋

```
너는 DDD 에 익숙한 아키텍트야 .
docs/requirements.md 를 읽고 다음을 만들어 줘 .

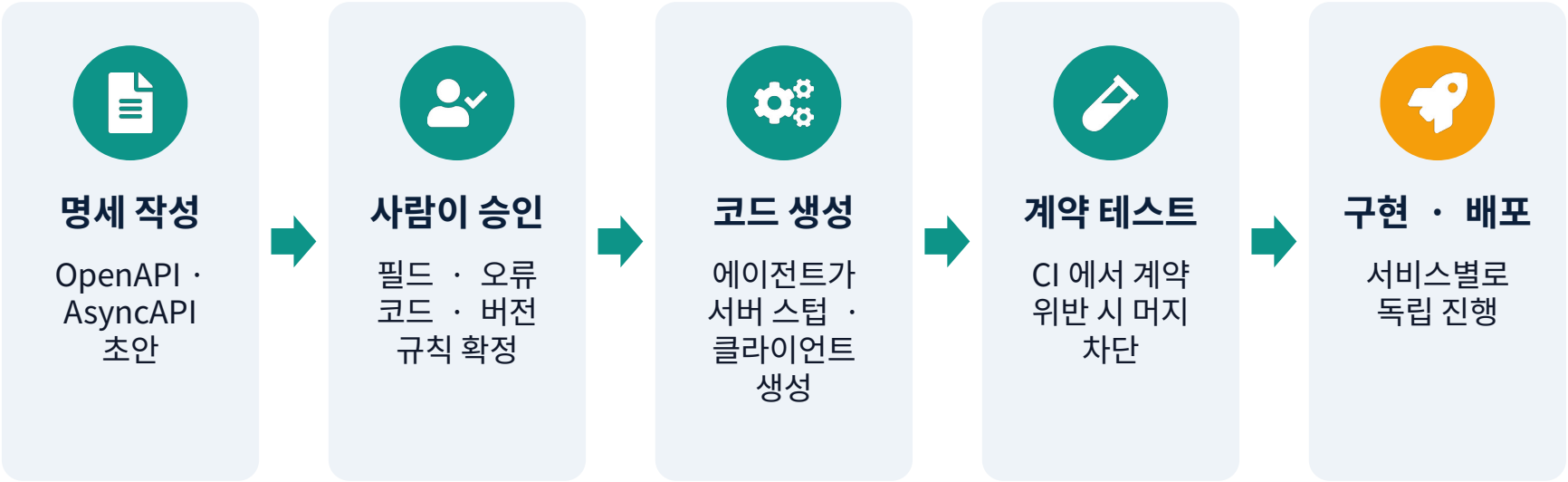
1. 도메인 이벤트 목록 ( 과거형 , 발생 주체 포함 )
2. 이벤트를 묶은 바운디드 컨텍스트 후보 2~3 안
3. 각 안의 서비스별 소유 데이터와 동기 / 비동기 호출
4. 각 안의 장단점 , 그리고
  ' 모듈러 모놀리스로 충분한 이유 ' 가 있다면 그것도

코드는 아직 작성하지 마 . 표로 정리해 .
```

설계 단계에서 코드 생성을 막아야 에이전트가 ' 빨리 만들기 ' 로 넘어가지 않습니다

계약 우선 — 인터페이스 불일치의 해법

원본 23 쪽 문제의 답 — 계약은 사람이 승인하고, 구현은 에이전트가 맡는다



규칙: 계약 파일 변경은 반드시 별도 PR + 사람 리뷰. 에이전트가 계약을 '맞춰서' 고치지 못하게

04

에이전트로 구현하기

원본 12~18 쪽 구현요소 6 가지의 2026 개정



구현요소 6 가지 , 2026 실무 선택

왼쪽은 원본 그대로 , 오른쪽 두 칸이 개정 내용입니다

2022 구현요소	2026 실무 선택	에이전트 활용 포인트
데이터 분산관리	서비스별 DB, Outbox, Saga	스키마 · 마이그레이션 · Saga 테스트 작성
RESTful / 비동기	REST · gRPC + Kafka · NATS · Redis Streams	계약에서 클라이언트 · 컨슈머 생성
폴리글랏	Python · Go · TypeScript 중 선택	언어 장벽 ↓ — 단 , 리뷰 가능한 언어로 제한
DevOps	GitOps, CI 가 에이전트 PR 검증	Dockerfile · 파이프라인 초안 작성
클라우드	컨테이너 + k8s / k3s (온프레미스 포함)	매니페스트 생성 , 리소스 설정 점검
Design for Failure	타임아웃 · 재시도 · 서킷브레이커 · 멱등성	장애 주입 테스트 , 트레이스 분석

데이터 분산관리 — 분산 트랜잭션 대신 Saga

원본 13 쪽 'Not 분산 트랜잭션 ' 을 구현하는 표준 패턴



Outbox 패턴 : DB 저장과 이벤트 발행을 한 트랜잭션에 묶어 ' 저장됐는데 이벤트 유실 ' 을 막음

비동기 이벤트 — 주문 서비스 예시

- **FastAPI + Redis Streams**
  실습 환경에서 가장 가벼운 조합
- **이벤트 이름은 과거형**
  order.created, stock.reserved
- **멱등성 키 필수**
  같은 이벤트가 두 번 와도 한 번만 처리
- **스키마는 contracts/**
  에이전트는 계약을 읽고 구현

```python
# services/order/app/main.py
import json, uuid, redis
from fastapi import FastAPI, Header

app = FastAPI()
r = redis.Redis(host="redis")

@app.post("/orders")
def create(body: dict,
           key: str = Header(alias="Idempotency-Key")):
    if not r.set(f"idem:{key}", 1, nx=True, ex=86400):
        return {"status": "duplicate"}
    oid = str(uuid.uuid4())
    # save PENDING order + outbox row
    evt = json.dumps({"id": oid, **body})
    r.xadd("order.created", {"data": evt})
    return {"id": oid, "status": "PENDING"}
```

이 코드도 에이전트가 계약 파일을 읽고 생성한 초안 — 사람은 멱등성 · 오류 처리를 리뷰합니다

서비스마다 에이전트 — git worktree 병렬 개발

- **서비스별 작업 폴더 분리**
  에이전트끼리 파일 충돌 없음
- **tmux 창 하나에 에이전트 하나**
  1 일차에 구축한 환경 그대로
- **각자 브랜치 → PR**
  lazygit 으로 diff 리뷰 후 머지
- **계약은 main 에서만 변경**
  worktree 에서 계약 수정 금지

```
# one worktree per service
$ git worktree add ../shop-user    -b feat/user
$ git worktree add ../shop-product -b feat/product
$ git worktree add ../shop-order   -b feat/order

# tmux: 3 panes, 3 agents
$ tmux new -s msa
  pane 1) cd ../shop-user    && claude
  pane 2) cd ../shop-product && codex
  pane 3) cd ../shop-order   && claude

# review & merge
$ lazygit
```

사람의 역할은 ' 동시에 세 명의 개발자에게 일을 나눠 주고 리뷰하는 팀장 ' 입니다

Design for Failure + 관측성

원본 18 쪽 — 실패를 전제로 설계하고, 실패를 '보이게' 만든다



타임아웃 · 재시도

기본

모든 외부 호출에 타임아웃, 재시도는 지수 백오프로



서킷 브레이커

차단

계속 실패하는 서비스는 잠시 끊어 장애 전파를 막음



멱등성

중복 방지

재시도와 중복 이벤트에도 결과가 한 번만 반영되게



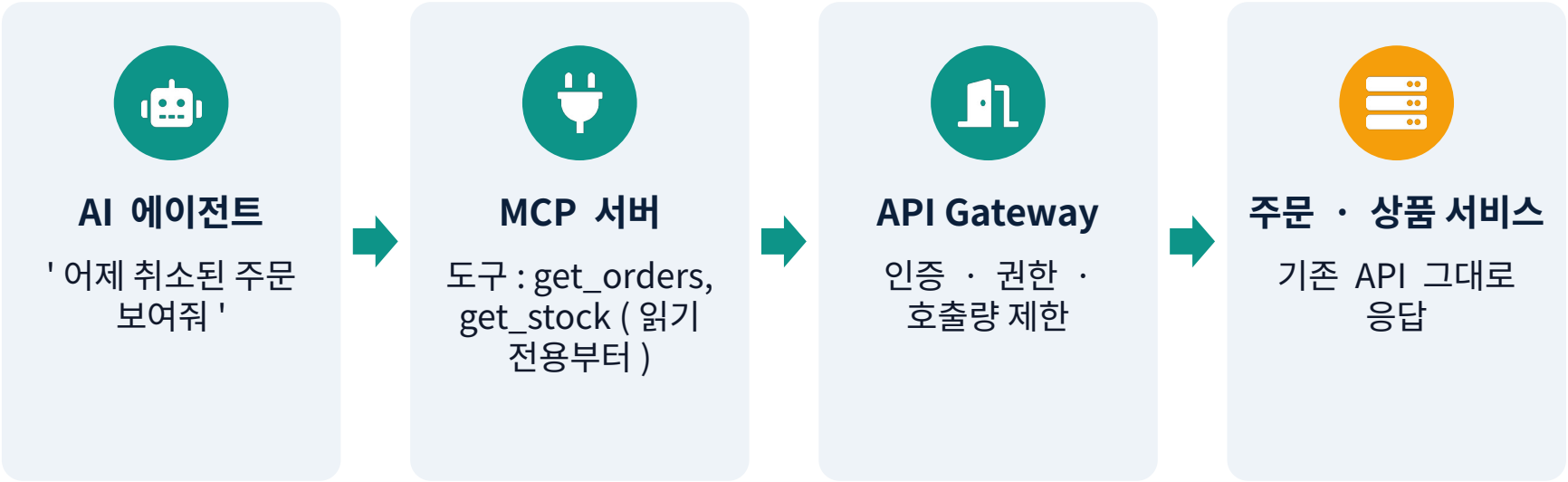
OpenTelemetry

추적

요청 하나의 전 구간 트레이스 — 에이전트에게 보여주면 원인을 함께 찾음

서비스를 에이전트에게 열기 — MCP

API Gateway 가 사람용 입구라면 , MCP 서버는 AI 에이전트용 입구입니다



쓰기 도구 (주문 취소 · 환불) 는 사람 승인 단계를 둡니다 — 원본의 ' 최소 권한 ' 원칙 그대로

05

문제점과 해결책 2026

원본 19~27 쪽 8 가지 문제의 개정판



문제점과 해결책 ① 설계 · 운영

2022 년 해결책은 여전히 유효합니다 . 에이전트가 실행 비용을 낮춰 줄 뿐입니다

문제	2022 해결책	2026 에이전트 활용
서비스 범위 설정	Bounded Context, DDD	에이전트와 이벤트 스토밍 , 결정은 사람이
레거시와 공존	SoA 연동 , API G/W	스트랭글러 피그 : 기능 단위 이전 + 회귀 테스트
운영 오버헤드	DevOps 배포 자동화	서비스 템플릿 + 에이전트가 CI/CD · IaC 생성
인터페이스 불일치	팀 간 커뮤니케이션	계약 우선 + CI 에서 계약 테스트 강제

문제점과 해결책 ② 코드 · 데이터 · 테스트

에이전트가 가장 크게 도움이 되는 영역은 ' 테스트 작성 ' 입니다

문제	2022 해결책	2026 에이전트 활용
코드 중복	단일 언어 · 프레임워크	공통 템플릿 · 스킬로 ' 같은 방식 ' 의 중복 관리
데이터 중복	전체 시스템 관점 고려	소유 서비스 명시 , 이벤트 · CDC 로 동기화
분산 복잡성	Actor Model, Reactive	이벤트 기반 + 분산 트레이싱을 에이전트가 분석
테스트의 어려움	(상호 테스트 어려움)	계약 · 통합 테스트를 에이전트가 작성 , 시나리오는 사람이

새로 생긴 문제 — 바이브 코딩의 함정

2022 년에는 없던 위험입니다 . 모두 실제로 자주 일어납니다

분산 모놀리스 양산

서비스는 많은데 동기 호출로 줄줄이 묶임 → 경계부터 재검토

존재하지 않는 API · 패키지

에이전트가 지어낸 이름 → 잠금 파일 · 의존성 검사 필수

비밀키 유출

코드 · 로그에 토큰 하드코딩 → 시크릿 스캔을 CI 에

테스트를 고쳐서 통과

구현 대신 테스트를 수정 → 테스트 파일 변경은 별도 리뷰

서비스마다 다른 규칙

세션마다 스타일이 표류 → 공통 AGENTS.md · 린터로 고정

리뷰 병목

생성 속도 > 리뷰 속도 → 작은 PR, 자동 검사 먼저

06

개발 팀 운영

원본 28~32 쪽 — 팀의 정의가 바뀌었다



팀 구성의 변화

2022 팀



사람 5~9 명이 한 서비스

- 피자 두 판 규모의 팀
- 기획 · 개발 · 테스트 · 운영 분업
- 사람 수가 곧 개발 속도



2026 팀



사람 1 + 에이전트 N

- 사람은 설계 · 리뷰 · 운영 책임
- 에이전트는 구현 · 테스트 · 문서
- 리뷰 역량이 곧 개발 속도

책임은 위임할 수 없습니다 — 에이전트가 쓴 코드도 머지한 사람의 코드입니다

원본 팀 운영 원칙 4 가지 , 2026 버전

지식노동자를 위한 원칙은 에이전트를 다루는 사람에게도 그대로 적용됩니다



최적화된 팀 규모

29 쪽

작은 팀 → 사람 1~2 명이 서비스 하나와 에이전트들을 소유



독립적인 팀 운영

30 쪽

서비스별 AGENTS.md 로 자율 , 공통 가드레일로 일관성



아키텍처 선택 자율

31 쪽

선택은 자유 , 단 ADR 로 이유를 남겨 에이전트도 따르게



상향식 개발

32 쪽

명세 주도 (spec-driven) + 작은 반복 , 큰 계획 일괄 생성 금지

에이전트 가드레일 — 맡기기 전에 정할 것

권한은 작게 시작해서 신뢰가 쌓이면 넓힙니다



샌드박스 · 허용 명령

작업 폴더 밖 접근 금지, 위험 명령은 승인 후 실행



main 직접 push 금지

모든 변경은 브랜치 → PR → CI → 사람 리뷰



비밀은 환경변수로

.env 는 git 제외, 운영 키는 에이전트에게 주지 않음



사람 승인 작업

DB 마이그레이션 · 배포 · 데이터 삭제 · 계약 변경



비용 한도

API 사용량 · 실행 시간 상한, 무한 루프 방지



기록 남기기

에이전트 세션 로그 · PR 설명에 근거를 남김

07

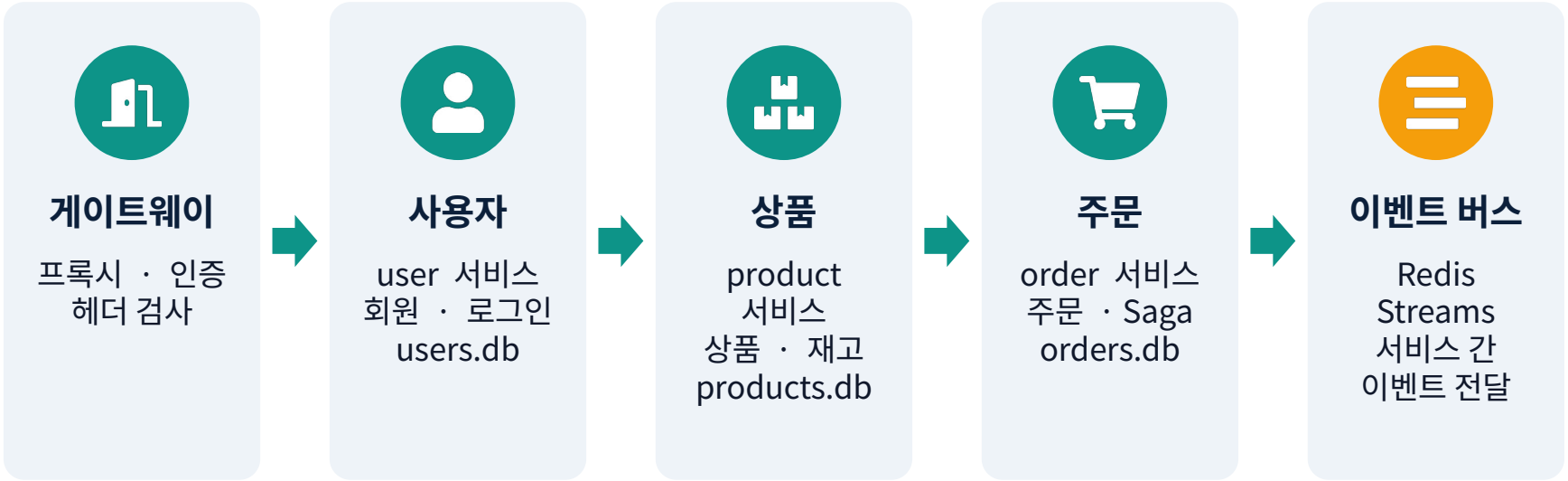
실습 : 주문 시스템 MSA

사용자 · 상품 · 주문을 에이전트 3 개로



실습 아키텍처

docker compose 하나로 로컬에서 전부 실행합니다



서비스마다 자기 DB(SQLite) 를 가지며 , 다른 서비스 DB 에 직접 접근하면 실습 감점

실습 진행 순서

LAB



진행 방식

1 인 팀장이 에이전트 개발자 세 명을 지휘하는 하루

1 **명세 · 계약 확정 (사람 , 40 분)**
요구사항 → 이벤트 스토밍 → contracts/ 와 루트 AGENTS.md 작성

2 **병렬 구현 (에이전트 , 60 분)**
worktree 3 개 + tmux, 서비스별 에이전트에게 계약 기반 구현 지시

3 **통합 · 계약 테스트 (40 분)**
docker compose up, 주문 → 재고 예약 → 완료 흐름 확인

4 **장애 주입 (20 분)**
product 서비스를 멈추고 주문이 보상 처리되는지 확인

서비스 에이전트에게 주는 첫 지시

읽을 파일을 명시
AGENTS.md · 계약 · 요구사항

테스트 먼저
실패하는 테스트 → 구현 순서

범위 제한
자기 서비스 폴더만 수정

완료 조건 명시
pytest 통과 + 실행 방법 보고

```
services/order/AGENTS.md 와 contracts/order.openapi.yaml,
contracts/events.asyncapi.yaml 을 먼저 읽어 .

1. 계약의 모든 엔드포인트에 대한 pytest 를 먼저 작성해 .
   재고 부족 · 결제 실패 · 중복 요청 시나리오를 포함해 .
2. 테스트가 실패하는 것을 확인한 뒤 구현해 .
3. services/order/ 밖의 파일과 contracts/ 는 수정하지 마 .
   계약이 이상하면 수정하지 말고 나에게 보고해 .
4. 끝나면 pytest 결과와 실행 방법을 요약해 줘 .
```

좋은 지시 = 읽을 것 + 순서 + 금지 범위 + 완료 조건

2022 → 2026 한 장 요약

왼쪽은 원본의 개념 , 아래는 2026 년의 실천 방법



모놀리스 vs MSA

모듈러 모놀리스 먼저



서비스 경계

에이전트와 이벤트 스토밍



인터페이스

계약 우선 + 계약 테스트



분산 데이터

Saga + Outbox



DevOps

CI 가 에이전트 PR 검증



장애 대비

재시도 · 멱등성 · 추적



작은 팀

사람 1 + 에이전트 N



팀의 지식

AGENTS.md · ADR

참고 자료

원본 참고자료에 2026 년 기준 자료를 추가했습니다

자료	주소
Martin Fowler — Microservices	martinfowler.com/articles/microservices.html
Martin Fowler — MonolithFirst	martinfowler.com/bliki/MonolithFirst.html
Chris Richardson — Saga · Outbox 패턴	microservices.io
AGENTS.md — 에이전트 규칙 파일	agents.md
Model Context Protocol (MCP)	modelcontextprotocol.io
OpenTelemetry — 분산 추적	opentelemetry.io
(원본 참고) 조대협의 블로그	bcho.tistory.com/948

과제와 Q&A

실습한 주문 시스템을 저장소에 올리고 , 아래 세 가지를 제출하세요



- contracts/ · 루트와 서비스별 AGENTS.md · ADR 1 건 이상
- 장애 주입 결과 : product 중지 시 주문이 CANCELLED 되는 로그 또는 트레이스
- 회고 한 단락 : 에이전트에게 맡긴 일과 내가 직접 결정한 일의 구분