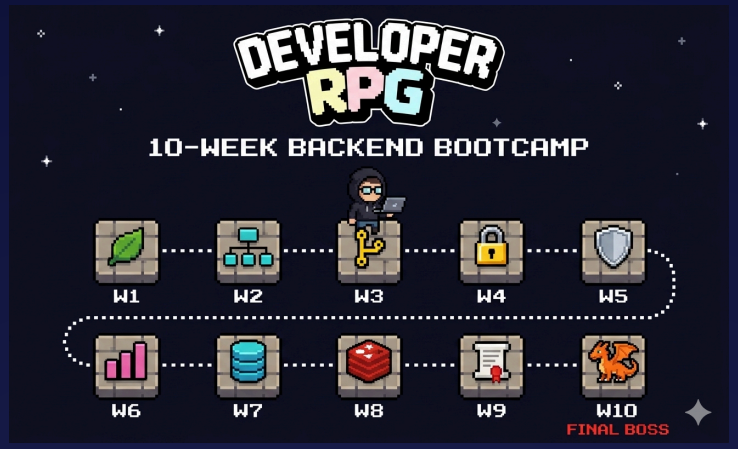


# Week 8 강의

## Event Sourcing — 캐시 너머의 설계

상태 vs 이벤트 · 2026-07-25

# 통장 = 거래의 합 

# ⚔ QUEST 왜 이 주제인가

- W7 Redis = 상태 캐시. 그 너머 패턴을 본다
- 면접 단골 (CQRS / Event Sourcing 들어보셨나?)
- 은행·소셜·고프로 도메인의 핵심 설계
- 오늘은 개념 + 언제 쓰나 + 안 써도 OK 까지

> *"캐시는 현재 상태 의 사본. Event Sourcing 은 과거 모두 의 기록."*

# 오늘 다룰 것 

1. 비유 — 통장 vs 잔액
2. 상태 저장 vs 이벤트 저장
3. CQRS — 읽기와 쓰기 분리
4. 언제 쓰고 언제 안 쓰나
5. 도메인에서 + 이력서에서

# 시작 전 — 용어 카드

| 용어 | 한 줄 정의 |
| --- | --- |
| **Event Sourcing** | 현재 상태 가 아닌 이벤트 흐름 을 저장 |
| **CQRS** | 쓰기(Command) 와 읽기(Query) 책임 분리 |
| **Read Model** | 읽기 전용으로 최적화 된 별도 DB / 캐시 |
| **이벤트 발행** | 도메인 변경을 신호 로 알림 (Spring `@Async` ) |
| **이벤트 버스** | 이벤트를 전달 하는 인프라 (Kafka 등) |

> *모르는 단어 나오면 이 표 다시.*

# ⚔ QUEST Part 1 — 통장 비유

```
[상태] "현재 잔액: 50,000 원" — 과거 X
[이벤트] 거래 _전부_ 기록 — 잔액은 _합산_
```

> *은행 = 거래 기록 이 진실, 잔액은 계산.*

# 같은 좋아요 — 두 모델

```
[상태 모델]
Post(id, likeCount=42)
→ UPDATE Post SET likeCount=43

[이벤트 모델]
LikeEvent(postId, userId, action='LIKE', at=...)
LikeEvent(postId, userId, action='UNLIKE', at=...)
→ INSERT 만 (UPDATE/DELETE 없음)
→ 좋아요 _수_ = 이벤트 합산
```

> *한쪽은 현재 만, 한쪽은 역사 모두.*

# 왜 그래야만 하는가 — 4 동기

| # | 동기 | 핵심 |
|---|---|---|
| 1 | Write 가 무거움 | UPDATE = 인덱스+락+Tx · INSERT 가벼움 |
| 2 | Read >> Write | 보통 100:1 ~ 1000:1 — read model 분리 |
| 3 | 비동기 반영 OK | 좋아요·통계 0.5초 지연 사용자 영향 X |
| 4 | MSA 이벤트 전파 | Kafka 1발행 → N서비스 소비 |

> *4 중 2 개+ 해당하면 ES 도입 진지하게 검토.*

# ⚔ QUEST Part 2 — ES 의 효과

✅ 감사 (Audit) — 누가 언제 무엇을 했는지 _전부_
✅ 시간 여행 — 임의 시점 상태 복원 가능
✅ 디버깅 — _재현_ 항상 가능
✅ 분석 — 행동 패턴 무한 분석
✅ 재처리 — 버그 발견 시 이벤트 재처리

❌ 저장 공간 ↑↑
❌ 조회 _계산_ 필요 (또는 별도 read model)
❌ 학습 곡선 ↑

# CQRS — 함께 쓰이는 패턴

```
Command Query Responsibility Segregation
  명령    쿼리      책임          분리

[Command 측]                [Query 측]
이벤트 저장                   read model
↓                               ↑
Event Store ─── 비동기 ─→ 읽기 전용 DB
                           (캐시·검색·통계)
```

> *쓰기 = 이벤트 1줄, 읽기 = 최적화된 read model.*

# 적용 사례

🏦 은행 거래
- 입출금 = 이벤트
- 잔액 = read model
- 감사 의무 → ES 거의 _필수_

🛒 e-commerce 주문
- 주문/배송/결제 = 이벤트
- 주문 상태 = read model
- 분쟁 시 _시간순_ 재현

# 적용 사례 (계속)

📱 소셜 네트워크
- 좋아요/댓글 = 이벤트
- 알림 / 통계 / 추천 = read model 다수

🚗 IoT / 위치 추적
- GPS pin = 이벤트
- 현재 위치 / 경로 = read model

# 언제 안 쓰나

- ❌ CRUD 가 충분한 단순 도메인 (게시판)
- ❌ 트래픽 작은 서비스 (오버 엔지니어링)
- ❌ 팀 _학습 곡선_ 부담 큰 시점
- ❌ 즉시 일관성 _절대_ 필요한 도메인

- ✅ 감사 의무 있는 도메인
- ✅ "왜 이렇게 됐지?" 자주 물음
- ✅ 분석·추천 read model 다수
- ✅ 팀이 _이벤트 사고_ 익숙

# ⚔ QUEST Part 3 — 부분 도입

전체 ES 부담스러우면 이벤트 발행 만 도입.

```
주문 생성:
  1. Order INSERT (= state)
  2. OrderCreated 이벤트 발행 (= event)

이벤트 구독자:
  - 알림 서비스
  - 통계 서비스
  - 추천 시스템
```

> *부분 도입. 캐시·알림·통계 분리 만으로도 큰 효과.*

# Spring 으로 시작하기

```
record OrderCreated(Long orderId, ...) {}

// 발행
publisher.publishEvent(new OrderCreated(...));

// 구독
@EventListener @Async
public void handle(OrderCreated e) {
  notification.send(...);
}
```

> *외부 큐 (Kafka) 없이도 개념 도입 가능.*

# Kafka 같은 이벤트 버스 까지 가면

```
Producer (서비스 A) → Kafka topic
                          ↓
   ├── Consumer 1 (알림)
   ├── Consumer 2 (통계)
   └── Consumer 3 (감사 로그)
```

✅ 서비스 _독립_
✅ 새 구독자 _쉽게_ 추가
✅ 장애 _격리_ (한 구독자 죽어도 OK)

❌ 운영 복잡도 ↑↑

# 실무에서 — ES 자리

| 도메인 | ES 적합도 | 이유 |
|---|---|---|
| 은행 거래 | ★★★★★ | 감사 + 재현 필수 |
| e-commerce 주문 | ★★★★ | 분쟁 + 분석 |
| 소셜 좋아요 | ★★★ | 분석 + 추천 |
| 게시판 글 | ★★ | 단순 CRUD 충분 |
| 정적 콘텐츠 | ★ | 거의 X |

> 별 5 개 *도메인이 아니면* 이벤트 발행 *만 부분 도입.*

# 이력서엔 — Event Sourcing 카드

[주문 도메인 이벤트 발행으로 알림·통계 분리]
P (문제) 주문 처리 응답 1.2s — 알림·통계가 동기 호출
O (옵션) 비동기 / 큐 / 이벤트 발행
D (결정) Spring 이벤트 + @Async — 운영 복잡도 ↓
A (행동) OrderCreated 이벤트 + 3 구독자 분리
R (결과) 응답 1.2s → 200ms (-83%), 알림·통계 장애 격리

# AI 잘 쓰는 법 

- **잘하는 것**: ES / CQRS 패턴 설명, 도메인 적합도 분석
- **자주 hallucinate**: 운영 복잡도 과소평가, 한국 회사 사례 추측
- **검증 루프**:
    1. AI 답에 도입 비용 까지 묻기
    2. 본인 도메인 감사 의무 확인
    3. POC 작은 범위로
    4. evidence 에 부분 도입 결과

# 그럼 미션은? 🎯

- W8 미션 `09-week8-ai-native` = AI 네이티브 워크플로우 (이 강의 주제 X)
- 단, 6 공통 필수 기능 중 비동기 / 이벤트 영역과 직결
- 팀 프로젝트 (W9) 에서 이벤트 발행 1 건 도입 가능

> *막히면 → `{cohort}-질문` 채널 + 오피스아워 (화·목 `21:00`)*

# 질문 ㄱㄱ ?

이번 주 = "내 도메인은 _상태_ 인가 _이벤트_ 인가"
다음 면접 = "Event Sourcing 들어보셨나?" 답할 수 있게

*격주 강의 끝 — Week 9 부터 팀 프로젝트.*