복구의 엔지니어: 서비스가 무너지는 순간부터 복구까지의 기술

사고 첫 30분을 위한 1인 개발자와 소규모 팀 운영자의 실무 규율

ThakiCloud (Hyojung Han)

복구의 엔지니어: 서비스가 무너지는 순간부터 복구까지의 기술

사고 첫 30분을 위한 1인 개발자와 소규모 팀 운영자의 실무 규율

ThakiCloud (Hyojung Han)

Contents

1 첫 5분, 고치지 않고 분류한다

목요일 밤 11시, 알림 네 개가 동시에 터진다. 결제 API 타임아웃 증가, CPU 사용률 상승, "주문이 안 된다"는 사용자 메일, 대시보드의 34% 에러율. 표면적으로 네 가지다. 하지만 실체는 한 문제다. 이 시점에서 1인 개발자가 흔히 하는 실수는 눈이 가는 알림부터 손을 대는 것이다. CPU가 올라왔으니 인스턴스를 키우고, 결제가 막혔으니 재시작을 누른다. 30분 뒤에도 서비스는 여전히 반쪽이다. 시스템에는 두 가지의 "나"가 남긴 변화가 쌓였다. 원인 찾기는 처음보다 더 어려워졌다.

이 장은 그 30분을 없애기 위한 절차다. 사고 자체를 없애는 일은 아니다. 서버는 죽고 디스크는 차며 의존 서비스는 사라진다. 사고는 온다. 우리가 다룰 것은 사고가 온 뒤의 첫 30분이다. 그 30분의 첫 5분은 방향을 고르는 시간이다.

1.1 왜 분류가 먼저인가

증상을 보고 바로 작업하면 치료와 진단이 섞인다. 이 둘은 반대 방향의 작업이다. 진단은 시스템을 읽는 일이고 치료는 시스템을 바꾸는 일이다. 치료 도중 시스템이 변하면 진단이 오염된다. "재시작하고 나니까 에러가 줄었는데"라고 느끼는 순간, 재시작의 효과인지 단순 시간 경과인지 구분할 수 없게 된다. 이 구분이 나중에 "돌려야 할 변화"를 고르는 기준이 된다.

1인 개발자에게는 이 실수가 더 비싸다. 팀이 있으면 "잠깐, 그게 진짜 원인 맞아요?"라고 물을 사람이 있다. 혼자라면 그 질문은 내 자신에게만 의존한다. 그런데 사고 직후의 자신은 판단력이 가장 낮은 상태다. 긴장한 채로 가장 눈에 보이는 버튼부터 누르기 때문이다. 사고를 감시하고 "그거 말고 다른 것도 보세요"라고 말할 외적 통제자가 없는 것이다. 이 장의 나머지 부분은 그 외적 통제자의 역할을 5분 안에 수행하기 위한 절차다.

분류를 먼저 한다는 것은 사고를 두 단계로 끊는 일이다. 진단 단계와 치료 단계. 여기서 두 가지 손실을 막을 수 있다.

첫째, 잘못된 치료를 멈출 수 있다. 결제 API 타임아웃을 보고 DB 인스턴스를 키우는 순간, 그 인스턴스가 다음에 필요해지는 비용과 리스크가 이미 나갔다. 원인이 외부 결제사 API라면, 그 변화는 아무것도 해결하지 않은 채 시스템에 하나의 차이를 남긴다. 사고 중에 생긴 차이 하나하나가 나중에 원인 찾기를 어렵게 만든다.

둘째, 증거를 보존한다. 처음의 로그, 처음의 대시보드 수치, 처음의 사용자 보고. 이게 제일 값진 증거다. 나중에 "처음에 뭐가 떴지"를 기억으로 복원하면 기억은 편의적으로 수정된다. 당시의 수치를 적어 두고 있어야 후에 이 변화를 돌려도 효과가 있었는지 판단할 수 있다.

그래서 첫 5분의 목표는 명확하다. 작업을 하는 것이 아니라 세 가지 답을 내리는 것이다. 사고인가, 어떤 종류의 사고인가, 어느 정도 심각한가.

1.2 세 질문으로 사고 여부를 판정한다

알림이 울린다고 전부 사고는 아니다. 반대로 알림이 없어도 사고일 수 있다. 매출 그래프가 조용히 내려가는 날, 아무 알림이 없는데 사용자가 "왜 안 돼요"라고 오는 날. 이런 날에 "알림이 없으니까 정상이다"라고 판단하면 안 된다. 판정을 돕는 질문은 세 개다.

사용자 문제가 일어나고 있는가. 에러율만 보고 "내 서비스는 정상"이라 단정하기 쉽다. 헬스체크는 내부 엔드포인트만 돌리기 때문이다. 헬스체크가 성공하는 것은 프로세스가 살아 있다는 뜻이지, 사용자가 원하는 일이 되는다는 뜻이 아니다. 5분 안에 확인해야 할 것은 에러 로그의 빈도와 실제 트래픽에서의 실패 건수다. 가능하면 직접 브라우저를 열어 결제나 로그인 같은 핵심 흐름을 한 번 돌려 본다. 직접 돌려 보는 것이 제일 빠르다. 대시보드에서 3분 동안 그래프를 읽는 것보다, 실제 흐름을

20초 만에 실패시키는 것이 더 많은 것을 알려준다. 실패 화면에서 "오류 코드: TIMEOUT"이 보이면, 이미 원인 범위가 반으로 줄었다.

확산되고 있는가. 한 유저의 문제인가, 일부인가, 전체인가. 대시보드에서 타임라인을 뒤로 30분 넘게 되감아 본다. 점진적으로 나빠지고 있으면 진행 중인 사고다. 갑자기 한 지점에서 끊어졌으면 특정 원인이 있고, 그 지점 주변에 변화를 찾아볼 가치가 있다. 한 유저나 한 지역만 타격받는다면, 그 유저의 세션이나 해당 지역의 네트워크 문제일 가능성이 높다. 부분적인 문제라면 그 경계를 그리는 것이 다음 단계에서 원인을 좁히는 단서가 된다. "결제만 안 되고 나머지는 된다", "한국만 안 되고 해외는 된다", "로그인만 5초 걸린다" 같은 경계는 원인 후보를 크게 좁혀 준다.

새로운 것인가. 오늘 배포한 것이 있는가, 설정을 바꾼 것이 있는가, 의존하는 외부 서비스가 최근 바뀐 것이 있는가. "새로운 것"이 없으면 원인이 외부이거나, 누적된 문제(디스크 부족, 메모리 누수, 인증 토큰 만료)일 가능성이 높다. 이 질문의 답이 "없다"여도 사고는 사고다. 다만 원인을 찾을 방향이 바뀐다. 최근 변경을 뒤집는 데 시간을 쓰지 말고, 외부 의존과 자원을 먼저 살피게 된다.

세 질문에 답할 수 있으면, 당신은 이미 사고의 종류를 절반은 안 것이다.

1.3 4축으로 증상을 분류한다

증상은 크게 네 축으로 나뉜다. 각 축은 다른 도구와 다른 우선순위를 요구한다.

축	증상	먼저 보는 것
가용성	접속 불가, 5xx	프로세스, 네트워크, 로드밸런서
성능	느려짐, 타임아웃	큐 길이, 대기 시간, CPU
정확성	결과가 틀림	최근 코드 변경, 데이터
데이터	손실, 불일치	쓰기 경로, 백업 상태

이 분류의 쓸모는 각 축이 다른 "멈추는 법"을 가진다는 점이다. 가용성 사고는 프로세스를 재시작하거나 트래픽을 돌리면 흔히 잡힌다. 성능 사고는 트래픽을 줄이거나 캐시를 쓰면 잡힌다. 정확성 사고는 거의 항상 특정 코드 변경을 돌리는 것이 답이다. 데이터 사고는 제일 조심해야 한다. 재시작하면 안 될 때가 있다. 재시작이 쓰기를 중단시키는 경우가 있기 때문이다. 프로세스를 죽이는 순간, 처리 중이던 트랜잭션이 반쯤 쓴 상태로 남을 수 있다.

혼자 서비스를 돌리는 사람이라면 이 네 축 중 어디에 속하는지를 5분 안에 못 박아야 한다. "어딘가 이상한데"라고 느끼는 상태로 30분을 보내면 30분 뒤에도 여전히 "어딘가 이상한데"일 것이다. 그리고 네 축이 겹칠 때가 있다. 느려짐(성능)이 심해지면 타임아웃이 되고(가용성), 타임아웃이 반복되면 재시도 폭주로 더 나빠진다. 이때는 겹친 축 중 "처음 시작된 것"을 찾는다. 타임라인을 되감아 보면, 거의 항상 한 축이 먼저 움직여 있다.

1.4 심각도를 세어 결정한다

심각도를 직감으로 결정하면 같은 사고가 사람마다 다른 등급이 된다. 중요한 사고를 "아니, 지금은 괜찮아"라고 놓치는 일도 생긴다. 대신 세 가지 질문으로 세어 본다.

누가 영향을 받는가. 전체 사용자인가, 일부인가, 핵심 고객인가. 전체 사용자의 결제가 막혀 있다면 심각하다. 일부 사용자의 프로필 사진이 안 보이는 것이라면, 다른 이야기다. "누구"는 숫자로 세면 편하다. 활성 사용자가 1,000명이면 10%가 100명이고, 그 100명의 결제가 막혀 있으면 100명이 막혀 있는 것이다.

무엇이 영향을 받는가. 핵심 흐름(구매, 로그인, 데이터 저장)인가, 부수적 기능인가. 핵심 흐름이 멈춘 사고는 시간의 가치가 다르다. 1분당 얼마가 흘러나가는지를 대략이라도 알면 이후 결정이 쉬워진다. 예를 들어 결제가 1분당 평균 100건이고 객단가가 5만 원이라면 10분 멈추면 대략 5,000만

원[추정]의 손실이다. 이런 수치를 사고 전에 한 줄이라도 적어 두면 "30분 동안 이대로 둬도 되나"라는 질문에 답할 수 있다. 사고 중에 손실을 계산하기는 어렵다. 긴장하면 숫자가 잘 안 나온다.

데이터가 위험한가. 데이터 손실이나 변형이 의심되면 무조건 심각도를 한 단계 올린다. 데이터 사고는 되돌리기 어렵기 때문이다. 가용성 사고는 서비스를 되살리면 되지만, 사라진 데이터는 못 되살린다. 여기에 한 가지 추가 질문이 있다. "이대로 두면 나중에 더 나빠지는가." 디스크가 95%면 시간이 지나면 100%가 된다. 메모리 누수면 시간이 지나면 재시작이 필요해진다. 나빠지는 방향의 사고는 심각도보다 "남겨둘 시간"이 짧다.

이 세 질문에 답하면 사고 등급이 사실상 결정된다. 1인 개발자에게는 공식적인 등급 체계보다, "지금 이대로 두면 어떤 손실이 쌓이는가"를 숫자로 쓰는 것이 더 현실적이다.

1.5 첫 5분, 실제 순서로 한 번

목요일 밤 사례를 처음부터 끝까지 순서대로 돌려 본다.

1. 알림 4개를 본다. 한 문제인지 확인한다. 결제 실패, 타임아웃 증가, CPU 상승, 에러율 34%. 공통점은 전부 결제 경로다.
2. 대시보드를 30분 되감는다. 22시 40분부터 점진적 악화. 그 전에 뭘 했나. 22시 15분, 결제 로직에 캐시를 넣는 소규모 배포를 했다. "새로운 것인가" 질문의 답은 예다.
3. 브라우저로 직접 결제 흐름을 돌린다. 실패. 오류 코드 TIMEOUT. 원인이 내 서비스 내부인지, 외부 결제사인지 구분해야 한다.
4. 로그를 본다. 타임아웃이 나는 곳은 외부 결제사 API 호출 구간. 내 서비스의 CPU가 높은 것은 타임아웃으로 쌓인 요청이 재시도로 폭주해서일 것으로 보인다.
5. 분류는 가용성(결제 불가)에 성능(느려짐)이 겹쳤다. 심각도는 전체 사용자 결제, 1분당 수백만 원[추정], 데이터 위험 없음.

6. 결론은 외부 결제사 API 지연이다. 내 배포는 관련 없다. 캐시 구간은 호출 전에 끝난다. 다음 행동은 2장의 "확산 멈추기"로 넘어간다.

여섯 단계가 5분이다. 각 단계는 30초에서 1분. 이 순서가 중요한 이유는 4번에서 "내 배포가 원인이다"라고 착각했다면 22시 15분 배포를 롤백했을 것이다. 롤백 자체는 나쁜 행동이 아니다. 하지만 원인이 외부인데 내 배포를 돌리면 두 가지를 잃는다. 시간을 잃고(롤백 왕복), "내 배포가 문제가 있었다"는 잘못된 기록을 남긴다. 이 기록이 나중에 진짜 원인 찾기를 오염시킨다.

1.6 첫 5분 체크리스트

이 모든 것을 5분 안에 하라는 것이 아니다. 5분 안에 다음을 하면 된다.

- 알림과 사용자 보고를 한데 모은다. 같은 문제인지 확인한다.
- 에러 로그의 첫 줄과 빈도를 적는다.
- 대시보드를 30분 뒤로 되감아 본다.
- 직접 브라우저를 열어 핵심 흐름을 한 번 돌려 본다.
- 최근 24시간의 변화(배포, 설정, 의존)를 기억하거나 기록한다.
- 세 질문에 답한다: 사고인가, 확산 중인가, 새 것인가.
- 네 축 중 어디에 속하는지 못 박는다.
- 심각도를 세어서 적는다.

8가지지만 실제로는 5분이면 충분하다. 각 항목은 30초에서 1분 정도의 일이다. 그리고 이 5분이 이후의 30분을 결정한다. 이 5분을 건너뛰면 사고를 "대응"하는 것이 아니라 "경험"하게 된다. 대응은 방향이 있는 일이고 경험은 방향이 없는 일이다. 사고의 첫 5분은 고칠 방향을 고르기 위한 시간이다.

2 확산을 막는다. 롤백이 첫 대응이다

1장의 결론에서 다음 행동이 결정됐다. 원인은 외부 결제사 API 지연이며, 이 지연으로 쌓인 재시도가 내 서비스를 무너뜨리고 있다. 이제 손을 댄다. 이 장의 첫 문장은 의외일 것이다.

사고 중에 가장 좋은 행동은 멈추는 것이다.

고치기는 원인을 알아야 한다. 멈추기는 원인을 몰라도 된다. 외부 결제사가 왜 느린지는 모른다. 그러나 "결제 요청을 지금 당장 10%만 보내고 나머지는 대기시킨다"는 행동을 원인을 모르고도 수행할 수 있다. 그 결과 재시도 폭주는 사라지고 CPU는 내려오며 내 서비스는 유지된다. 결제는 여전히 느리다. 하지만 느린 서비스와 죽은 서비스는 다른 문제다. 느린 서비스는 사용자가 기다릴 수 있고, 죽은 서비스는 아무것도 할 수 없다.

2.1 멈추기와 고치기의 차이

이 둘을 구분하지 않으면 사고 도중에 가장 비생산적인 일을 하게 된다. 원인을 찾으며 동시에 여러 가지를 바꾸는 일이다. 각 변경은 혼자서는 합리적이고 합치면 혼란이다. 30분 뒤 원인이 밝혀졌을 때, "그때 한 것 중 무엇이 효과가 있었지?"에 답할 수 없다. 답이 없으면 그 30분은 소모가 된다.

멈추기의 목표는 손실을 줄이는 것이다. 완벽하지 않아도 된다. 결제가 계속되면 결제의 비율을 줄인다. 모든 사용자가 느리면 일부 사용자만 통과시킨다. 트래픽이 폭주하면 트래픽을 떨어뜨린다. 이런 행동을 "부분적"이라 비하할 이유가 없다. 사고 도중에 100% 회복은 대부분 불가능하다. 70%를 5분 안에 세우는 것과, 100%를 2시간 만에 세우는

것, 사용자가 후자를 기다려 주지 않는다.

고치기는 원인을 없애는 것이다. 외부 결제사가 복구되어야 한다. 내 캐시 설정이 잘못됐어야 한다. 이런 행동은 멈추기 다음에 온다. 그리고 멈추기로 손실이 충분히 줄었으면, 고치기는 더 서두르지 않아도 된다. 시간이라는 여유가 생긴다. 여유가 생기면 판단력이 돌아온다.

2.2 블래스트 래디어스

멈출 때, 멈출 범위를 정한다. 블래스트 래디어스는 한 행동이 영향을 미치는 범위다. "전체 재시작"의 블래스트 래디어스는 전체 사용자다. "결제 라우트만 큐로 대기"의 블래스트 래디어스는 결제 중인 사용자다. 둘 다 합리적인 행동이지만 범위가 다르다.

1인 서비스에는 블래스트 래디어스를 세분할 구조가 없는 경우가 많다. 모든 게 한 프로세스다. 이런 경우에도 범위를 정할 수 있는 방법은 있다. 트래픽의 양을 줄이는 것이다. 전체를 멈추는 대신, 받는 양을 반으로 줄인다. 로드밸런서나 DNS, 클라우드의 스케일링 노브를 반으로 내리면, 같은 코드에서 절반의 폭발 범위를 만든다. 반경이 절반이면 실패가 나도 되돌릴 여유가 두 배다.

여기서 원칙은 단 하나다. **일단 멈추고 그 다음에 원인을 찾는다. 멈추는 행동을 원인 찾기와 동시에 하지 않는다.** "멈추면서 확인해 볼게"는 언제나 "멈추지도 확인도 못 한 채 20분을 보낸다"로 끝난다.

2.3 최근 변경 역순이 사고를 설명한다

원인을 찾을 때, 시간을 거슬러 올라간다. 사고 지점(T)에서부터 뒤로 본다.

순서	확인 대상	왜 먼저
1	최근 배포	가장 흔한 원인
2	설정, 환경 변수	배포와 비슷하게 흔함
3	데이터 마이그레이션	영향이 늦게 나타남
4	의존 서비스, 외부 API	내 바깥이지만 신호가 같음
5	자원(디스크, 메모리)	누적형, 갑자기 안 걸림

이 순서가 중요한 이유는 1번에서 답이 나오는 경우가 많아서다. "최근 배포가 있다"가 확인되는 순간, 배포를 롤백하는 것이 거의 항상 정답이다. 원인이 배포가 아니더라도, 롤백은 시스템의 상태를 "확실했던 지점"으로 되돌린다. 확실했던 지점에서의 실패는 불확실한 지점에서의 실패보다 이해하기 쉽다.

1장에서 한 배포를 했다고 결론지었다면, 배포를 돌린다. 그런데 1장에서 "원인은 외부"라고 결론지었다면, 이 표는 다른 용도로 쓰인다. 외부 API 지연에 내 서비스가 어떻게 반응하는지 보는 것이다. 재시도 정책이 공격적인가, 타임아웃이 긴가, 큐가 있는가. 이런 "내 서비스의 반응"은 보통 최근 변경에서 왔다.

2.4 한 번에 한 가지만 전부 적는다

사고 도중의 변경 규율은 두 가지다.

한 번에 한 가지만 바꾼다. CPU가 높으니 인스턴스를 키우고, 동시에 타임아웃이 있으니 타임아웃을 줄인다. 이런 "패키지" 변경은 금지다. 각 변경이 독립적으로 효과를 냈는지 알 수 없게 된다. 인스턴스를 키운 다음 5분을 기다린다. 효과가 있으면 그걸로 끝. 없으면 다음 변경. 5분을 기다리는 것이 지루해 보여도, 5분이 사고의 품질을 결정한다.

바꾼 것을 전부 적는다. 시간, 변경 내용, 기대했던 효과, 실제 효과. 이

기록은 4장에서 "회복을 검증"할 때 쓰고, 포스트모템에서 쓴다. 사고 도중에 "내가 뭘 했더라"를 기억으로 복원하면, 복원은 언제나 편하게 된다. 재시작을 세 번 했다고 기억이 있는데, 실제로는 네 번이었다. 이 차이는 나중에 "재시작이 효과를 냈는가"를 판단할 때 치명적이다.

기록은 간단하게 한다. 한 줄이면 된다. "22:53 - 결제 라우트 큐 대기 50% - 에러율 34%에서 21%로 하락." 이 정도면 충분하다. 이 한 줄이 사고의 시간선을 만든다.

2.5 트래픽 제어 도구 네 가지

멈추는 행동을 실현하는 도구로는 네 가지가 대표적이다.

도구	쓰는 경우	강도
기능 플래그	특정 코드 경로만 끄기	세밀
레이트 리미트	요청 수를 떨어뜨리기	중간
서킷 브레이커	실패하는 호출을 차단	세밀
큐, 대기	요청을 쌓아 두고 천천히 처리	강력

기능 플래그는 "최근 배포에서 새로 추가한 경로"를 끄는 데 쓴다. 배포를 통째로 롤백할 수 없는 경우, 예를 들어 배포에 다른 서비스와 공유되는 설정이 섞여 있으면, 플래그로 새 코드만 꺼낸다. 레이트 리미트는 트래픽이 폭주할 때 쓴다. 재시도 폭주는 거의 항상 레이트 리미트로 잡힌다. 서킷 브레이커는 외부 의존이 실패할 때 쓴다. 외부 API에 계속 호출하지 말고 잠시 차단해 둔다. 큐는 "지금 처리할 수 없는 요청을 나중에 처리한다"는 데 쓴다. 결제가 5분 지연되더라도, 10분 뒤 처리되는 것이 지금 실패하는 것보다 낫다.

무엇을 쓸지는 1장의 분류에 따라 다르다. 성능 사고에는 레이트 리미트와 큐. 정확성 사고에는 기능 플래그. 가용성 사고에는 서킷 브레이커와 재시작.

데이터 사고에는 이 네 가지 중 어떤 것도 함부로 쓰지 않는다. 쓰기를 중단하는 행동은 데이터 사고의 증상을 악화시킬 수 있다.

2.6 줄이는 정도를 어떻게 정하나

"50% 차단"이라는 숫자를 어떻게 고르나. 원칙은 간단하다. **반에서 시작하고 효과를 확인하며 조절한다.** 반으로 줄이면 효과가 있으면 25%로 내려오고, 효과가 없으면 75%로 올린다. 처음부터 100%를 막는 것은 막을 이유가 충분한 경우를 제외하면 과하다. 사용자의 일부를 완전히 차단하는 순간, 사용자는 "이 서비스는 죽었다"고 판단한다. 반대로 10%만 줄이면 효과가 거의 없다. 반은 중간값이라 효과 여부가 가장 명확하게 보인다.

여기서 "효과"를 확인하는 기준은 1장에 쓴 심각도 수치다. 에러율이 떨어지는가, CPU가 내려오는가, 매출 손실의 기울기가 완만해지는가. 기준을 사고 전에 정해 두지 못했다면, 사고 도중에 정한다. "에러율 20% 아래로 내려가면 충분"처럼 한 줄이면 된다. 이 한 줄이 없으면 줄이는 행동을 멈출 때를 모른다. 50%를 차단하고 에러율이 30%에서 28%로 내려왔을 때 "이걸로 됐나"를 직감으로 결정하면 사고가 길어진다.

2.7 건드리지 않는 것도 결정이다

반대의 규율도 있다. 사고와 관련 없어 보이는 것은 사고 도중에 건드리지 않는다. "그래도 디스크가 90%인데", "저 설정도 좀 이상한데" 이런 생각이 들면, 그것도 기록한다. 하지만 바꾸지는 않는다. 사고 도중에 한 번에 처리할 수 있는 변경의 수는 아무리 서두르든 둘을 넘지 않는다. 셋째 변경은 사고가 끝난 뒤의 작업이다.

이 규율을 지키면 사고의 시간선은 짧게 남는다. 시간선이 짧으면 나중에 각 변경의 효과를 판단하기 쉽다. 시간선이 길면 판단이 불가능해진다.

판단이 불가능해진 변경은 다음 사고의 원인 후보가 된다. 오늘 건드려 놓고 "효과가 있었는지 모르겠는" 설정은 다음 달에 또 터질 문제의 씨앗이다.

2.8 하지 말아야 할 것들

사고 도중에 하지 말아야 할 행동은 따로 목록으로 남길 가치가 있다.

핫픽스를 쓰지 않는다. "30초면 고쳐" 보이는 수정은 대부분 3시간을 태운다. 그리고 수정은 사고의 시간선에 하나의 더러운 점을 남긴다. 사고 중에 코드를 바꾸는 것은 원인을 확신할 때만 한다. 확신이 없으면 멈추는 행동으로 대신한다.

같은 것을 반복 재시작하지 않는다. 재시작 한 번은 진단이다. 두 번은 패턴이다. 세 번은 의존이다. 재시작을 세 번 했다면, 재시작이 진짜로 효과를 내는지 확인해야 한다. 재시작 직후 2분이 좋았는데 10분 뒤 다시 나빠졌다면, 재시작은 증상을 숨기는 중이다. 이런 신호를 보면 재시작을 멈추고 다른 방향으로 돌린다.

데이터를 "대충" 다시 쓰지 않는다. "백업이 있으니 다시 복구하면 되겠지"는 사고 도중에 가장 위험한 문장이다. 백업이 사고 직전의 상태인지, 그 전의 상태인지, 복구가 성공하는지, 이 세 가지를 확인하지 않고 "되겠지"는 금지다. 데이터 복구 행동을 할 때는 4장의 2차 사고 섹션을 참고해야 한다.

2.9 이 장의 실제 흐름

1장의 사례로 이 장을 끝까지 돌려 본다.

22:50 – 외부 결제사 API 지연 확인. 내 서비스 에러율 34%. 22:52 – 결제 라우트에 레이트 리미트 적용. 요청 50% 차단. 22:55 – 에러율 34%에서 21%로 하락. CPU 90%에서 65%. 22:58 – 나머지 50%도 큐로 대기 전환.

결제는 3분 지연되나 실패는 0. 23:05 – 외부 결제사 상태 페이지 확인: “조사 중” 표시. 내 서비스는 안정.

23:05 시점엔 사용자는 결제가 3분 느리다고 느끼지만 실패는 하지 않는다. 매출 손실은 최소화됐다. 원인은 여전히 외부에 있다. 하지만 내 서비스는 더 이상 무너지지 않는다. 이게 멈추기의 완성형이다. 완벽하지 않다. 손실의 기울기를 바꾼 것이 핵심이다.

3 통신은 시간선을 먼저 쓴다

사고가 터지면 사람은 두 가지를 잃는다. 판단력과 시간. 이 장은 이 둘을 회복하는 가장 비용이 낮은 방법, 즉 "쓰기"를 다룬다.

3.1 말하기 전에 쓴다

사고 도중 말은 빠르다. 메신저에 "지금 에러가 발생했다"는 내용을 입력하는 데 5초가 걸린다. 하지만 이 5초 동안 나온 말은 이후 모든 판단을 오염시킨다. "에러가 발생했다"는 사실이지만 동시에 "원인을 파악했다"는 인상을 줄 수 있다. 상대방은 "원인이 뭐냐"고 묻고, 작성자는 원인을 모른 채 답변해야 하는 위치에 놓인다.

쓰기의 규율은 이렇다. **말하기 전에, 시간선 파일에 한 줄을 기록한다.** 한 줄은 세 요소로 고정된다. 시간, 관찰, 행동.

```
22:50 – 결제 API 타임아웃 증가 관찰, 에러율 34%
22:52 – 결제 라우트 레이트 리미트 50% 적용
22:55 – 에러율 21%로 하락
```

이 세 줄이 있을 때와 없을 때 발언의 질은 다르다. 세 줄이 있으면 "22:50에 시작했고, 22:52에 레이트 리미트를 적용했고, 효과가 있었다"고 명확히 말할 수 있다. 없으면 "요즘 좀 불안정하다" 수준으로 모호하게 말할 수밖에 없다.

시간선 파일은 사고가 끝날 때까지 닫지 않는다. 포스트모템의 자료가 여기서 나온다. 4장에서 언급하겠지만 사고 중에 원인을 완전히 분석할 필요는 없다. 다만 "무엇을 했는지"는 반드시 기록해야 한다. 기록이 없으면 복구가 된 뒤에도 "그때 한 것 중 뭐가 효과가 있었는지"를 알 수 없다.

3.2 쓰는 곳이 중요하다

시간선은 한 파일에만 쓴다. 메신저 스레드, 메모 앱, 대시보드 주석에 흩어지면 안 된다. 흩어진 시간선은 통합이 어렵고 통합 과정에서 정보가 누락된다. 1인이라면 로컬의 텍스트 파일이면 된다. `incident-2026-08-23.md` 같은 이름으로 관리하고 사고가 끝나면 이 파일이 포스트모템의 입력이 된다.

소규모 팀이라면 한 파일에 공동으로 기록하는 것이 좋다. 각자가 자신의 행동을 적고 다른 사람의 행동을 읽는다. 이를 통해 "내가 뭘 했고 네가 뭘 했다"를 실시간으로 공유하게 된다. 공유가 이루어지면 같은 행동을 둘이 동시에 하는 사고가 줄어든다. 2장에서 말한 "한 번에 한 가지만" 규율이 팀에서도 지켜지려면 이 공유가 전제다.

3.3 에스컬레이션은 타이머가 아니라 세 기준

"30분이 지나도 안 고쳐지면 도움 요청"이라는 규칙을 아는 사람이 많다. 하지만 이 규칙의 문제는 30분이라는 숫자가 의미없다는 점이다. 30분이면 충분한 사고도 있고 5분이면 도움을 불러야 할 사고도 있다. 대신 쓰는 기준은 세 개다.

멈출 수 없다면. 2장의 멈추기 도구를 다 써도 손실이 줄지 않는다. 에러율이 그대로이고 매출이 그대로 떨어진다. 멈출 수 없는 상태는 현재 사용할 수 있는 도구가 부족하다는 뜻이다. 더 큰 도구나 다른 사람의 판단이 필요하다.

이해할 수 없다면. 원인이 보이는데 그 원인을 고치는 방법을 모른다. 외부 의존이 죽었는데 그 의존을 우회할 경로가 없다. 데이터가 손상된 것 같은데 손상의 범위를 모르겠다. 이해하지 못하는 상태는 다음 행동을 고를 수 없는 상태다.

권한이 없다면. 고칠 수 있는 방법이 보이는데 그 방법을 실행할 권한이

없다. 클라우드 계정의 결제 한도를 올리려면 다른 사람의 승인이 필요하다. 외부 결제사에 연락하려면 사업자 계정 관리자가 필요하다. 권한이 없으면 작성자가 수행할 수 있는 행동의 범위가 줄어든다.

세 기준 중 하나가 해당되면 도움을 부른다. 이때 중요한 것은 부를 때 무엇을 함께 보내느냐다. "도와주세요"라고만 하면 도와주는 사람은 다시 처음부터 진단을 시작한다. 대신 시간선 파일을 보낸다. 세 줄이면 충분하다. "22:50에 시작했고 22:52에 레이트 리미트를 적용했고, 효과가 있었다. 지금은 멈출 수 있다. 다음으로 원인은 외부 결제사로 보이는데, 해당 API를 우회할 방법이 필요하다." 이 문장이 있으면 도와주는 사람은 22:52 이후의 일만 생각하면 된다.

1인 개발자에게는 이 기준이 더 현실적이다. 도움을 부를 대상은 팀이 아닐 수 있다. 클라우드 제공사의 지원이거나, 또는 그냥 "잠시 쉬었다가 다시"일 수 있다. 다만 세 기준 자체는 동일하다. 멈출 수 없으면 지원 티켓을 쓴다. 이해할 수 없으면 문서를 다시 읽거나 커뮤니티를 검색한다. 권한이 없으면 권한을 가진 사람에게 연락한다. 기준이 있어야 "이제 해야 할 일이 뭐지"를 알 수 있다.

3.4 사용자에게 보내는 메시지는 세 개면 충분하다

사용자에게 보낼 메시지는 많을수록 안 좋아진다. 매 5분마다 "여전히 조사 중"을 보내면 사용자는 그 메시지만을 기다리게 된다. 대신 세 개의 메시지로 고정한다.

첫 메시지: 시작. 이 메시지의 역할은 "모르고 있었는지, 알고 있었는지"를 구분하는 것뿐이다. 내용은 세 가지. 무엇이 일어나고 있는지(증상), 우리가 무엇을 하고 있는지(현재 행동), 다음 업데이트는 언제일지(예상 시간). "지금 결제에 문제가 있고 트래픽을 줄이는 중입니다. 30분 뒤에 다시 업데이트하겠습니다." 이 정도면 된다. 원인을 말하는 것은 이 시점에는 금지다. 원인을 모르면 "원인 파악 중"이라고만 한다.

두 번째 메시지: 진전. 첫 메시지에서 약속한 시간에 보낸다. 이번에는 변화가 있다. "에러율이 줄었습니다" 또는 "여전히 진행 중입니다, 추가로 트래픽을 줄였습니다". 중요한 것은 첫 메시지의 예상 시간을 지키는 것보다 "예상 시간보다 늦어질 것"을 먼저 말하는 것이다. 30분 뒤 업데이트를 약속하고 45분이 지나서 "여전히 조사 중"을 보내면 사용자는 신뢰를 잃는다. 대신 28분에 "조사가 예상보다 길어지고 있습니다, 15분 뒤에 다시 업데이트합니다"를 보내면 같은 지연이라도 다르게 느껴진다.

세 번째 메시지: 종료. "복구되었습니다"라고만 하면 안 된다. 세 가지를 말한다. 무엇이었는지(짧은 원인 설명), 어떻게 고쳤는지(행동), 현재 상태(모든 것이 정상인지, 일부는 여전히 느린지). "외부 결제사 API 지연으로 결제가 실패했습니다. 트래픽을 줄여 임시로 처리했고 결제사 복구 후 정상으로 돌아왔습니다. 현재 모든 결제가 정상 처리되고 있습니다." 이 메시지가 있어야 사용자는 "또 터지면 같은 방식으로 대응하겠다"는 신뢰를 가질 수 있다.

3.5 확신하지 않은 것을 약속하지 않는다

이 장의 핵심 한 줄을 강조한다. **"조금만 기다리면 해결될 겁니다"라는 문장은 해결이 보장된 경우에만 쓴다.** 보장되지 않으면 쓰면 안 된다.

이 유혹은 크다. 사용자의 불안이 작성자의 불안을 만든다. "해결될 겁니다"를 치면 작성자의 마음도 한결 편해진다. 다만 그 편함이 이후의 판단을 흐린다. 해결이 보장되지 않은 상황에서 "보장"을 말하면 그 말이 사실로 만들어져야 한다. 그래서 더 서두르고 더 많은 것을 바꾸고 더 위험한 행동을 한다. 2장의 "한 번에 한 가지만" 규율이 여기서 무너진다.

대신 쓰는 문장은 사실에 기반한다. "레이트 리미트를 적용했고 에러율이 줄었습니다"는 사실이다. "원인은 외부 결제사로 보입니다"는 사실이다(보입니다). "30분 뒤에 업데이트하겠습니다"는 사실이다(작성자가 컨트롤 가능). 이 세 가지 사실만으로도 사용자는 충분히 안심한다. 안심의

핵심은 “누가 보고 있다”는 점이다.

3.6 지원 티켓을 열 때

에스컬레이션 기준 중 “멈출 수 없다면”이 성립했는데, 도움을 부를 사람이 클라우드 제공사의 지원이라면 티켓을 쓰는 방식이 시간을 결정한다. 티켓은 3장의 시간선 구조를 그대로 쓴다. 첫 줄에 증상(에러코드 포함), 그 다음에 시간선, 그 다음에 시도한 것, 그 다음에 요청.

“결제 API에서 504 타임아웃이 34% 발생합니다. 22:50 시작, 22:52 레이트 리미트 적용 후 21%로 하락. 내 네트워크와 프로세스는 정상, 인스턴스 간 통신에서 타임아웃 발생합니다. 인스턴스 A와 B 사이의 네트워크 상태 확인 요청합니다.”

이 티켓에는 세 가지가 들어 있다. 증상의 모양(504, 34%), 시간의 흐름(시작, 완화), 범위의 좁힘(내부 정상, 인스턴스 간 통신). 지원사는 이 셋이 있으면 재진단 없이 확인만 하면 된다. “문제가 발생합니다”만 있는 티켓에는 지원사도 처음부터 진단을 시작한다. 같은 사고가 두 번 일어난다.

3.7 사고 중에 채팅으로 조사하지 않는다

메신저에서 사고를 대응하면서 같은 창에서 로그를 붙이고 코드를 읽고 “아 이거 아닐까요?”를 치고, 다시 로그를 붙인다. 이 흐름은 집중을 조각낸다. 조사는 터미널에서, 기록은 시간선 파일에서, 소통은 메신저에서. 각자 자리가 있다.

메신저에 올리는 것은 결론이다. “22:55에 에러율이 21%로 하락, 레이트 리미트 효과 확인” 같은 한 줄. 과정은 쓰지 않는다. 과정은 시간선 파일에 있다. 메신저의 역할은 “현재 어디쯤인지”를 알리는 것이다. “지금 뭘 하고 있는지”를 보여 주는 것이 아니다.

3.8 사장이나 고객이 전화하면

메시지와 전화는 다른 매체다. 전화에는 "조금만 기다리면 해결될 겁니다"의 유혹이 더 강하다. 상대방이 목소리로 불안하게 물으면 나도 모르게 확신을 주게 된다. 전화에서 지켜야 할 구조는 단순하다. **결론을 먼저, 현재 위치를 다음에, 부탁을 마지막에.**

"지금 결제에 문제가 있고 트래픽을 줄여서 대응하고 있습니다. 에러율은 줄었습니다. 현재 외부 결제사 API가 원인인 것으로 보이고, 그쪽 복구를 기다리는 중입니다." (결론과 현재 위치) "30분 뒤에 다시 연락드리겠습니다. 그전에 추가 상황이 있으면 바로 드리겠습니다." (부탁)

전화에서 절대 하지 말아야 할 것은 원인을 단정하는 것이다. "외부 결제사 탓입니다"라고 말하면 나중에 원인이 다르면 두 번의 사과가 필요하다. "보입니다", "의심된다"는 표현이 이 한 줄을 지킨다.

또 한 가지. 전화 중에 "저기, 사실은 좀 더 심각한데"를 삼간다. 심각도를 전화 중에 올리면 상대방은 그 상승분만을 기억한다. 심각도를 올릴 때는 새 메시지로, 시간선과 함께 올린다.

3.9 1인이라면 상태 페이지가 통신이다

사용자에게 직접 메시지를 보내는 채널이 없는 1인 서비스에는 상태 페이지가 그 역할을 한다. 상태 페이지가 없다면 사고 도중에 만들 필요 없다. 대신 두 가지를 한다. 하나는 서비스 홈페이지나 앱의 첫 화면에 한 줄을 올려 놓는 것. "결제에 문제가 발생하고 있습니다. 확인하고 있습니다." 다른 하나는 사용자가 가장 많이 보는 채널, 예를 들어 공지 게시판이나 SNS 계정에 1장에서 쓰는 세 개의 메시지를 올리는 것.

상태 페이지를 이미 두고 있다면 세 개의 메시지를 그대로 쓴다. 첫 메시지에는 예상 시간을, 두 번째에는 진전을, 세 번째에는 원인과 행동을.

상태 페이지의 역할은 “우리는 보고 있다”를 보여주는 것이다. “우리는 알고 있다”를 보여주는 것이 아니다. 그래서 원인을 쓰는 시점은 세 번째 메시지 이후다.

이 둘의 차이를 지키면 1인이어도 통신이 된다. 상대방에게 “알고 있는지”를 보여주는 행위가 통신의 핵심이다.

3.10 통신은 사고의 일부다

통신은 사고 대응의 부수적 기능이 아니다. 통신은 사고 대응 그 자체다. 사용자가 “서비스가 죽었는지, 죽지 않았는지”를 모르면 그 모른다는 상태가 손실의 일부다. 매출이 그대로 떨어지는 것보다 “이 서비스는 신뢰할 수 없다”고 생각하는 사용자가 생기는 것이 더 오래 남는다.

세 개의 메시지, 한 개의 시간선 파일, 세 개의 에스컬레이션 기준. 이 셋이 있으면 사고 도중에 “무엇을 말해야 하지”를 고민하지 않아도 된다. 고민하는 시간은 판단력보다 먼저 소진된다.

4 회복은 검증이 아니라 희망이 아니다

23:05. 외부 결제사 API가 복구됐다. 에러율은 21%에서 4%로 내려왔다. "되셨네"라고 생각하는 순간, 이 장의 첫 문장이 필요하다.

회복했다고 느끼는 것과, 회복했다는 것은 다른 일이다.

에러율 4%는 사고 전의 0.5%보다 훨씬 높다. "4%면 괜찮지"라고 넘기면 남은 4%는 조용히 계속 발생한다. 사용자는 그 4% 안에 있다. 4%가 괜찮다면 1%는? 1%가 괜찮다면 0%를 기대하는 것은 불합리한가? 이 질문을 사고 도중에 답하면 "회복"의 기준을 숫자로 세울 수 있다.

4.1 회복 기준을 사고 중에 못 박는다

회복 기준은 세 요소로 고정된다. 어떤 지표, 어떤 범위, 얼마나 오래.

어떤 지표. 사고의 축에 따라 다르다. 가용성 사고라면 에러율과 요청 성공률. 성능 사고라면 P95 지연시간. 정확성 사고라면 정답으로 확인되는 비율(예: 결제 성공 후 데이터베이스에 기록된 건수). 데이터 사고라면 불일치 건수. 1장에서 축을 분류했다면 그 축의 지표를 고른다.

어떤 범위. "사고 전"을 기준으로 삼는다. 사고 전의 수치가 기억으로만 남아 있다면 대시보드의 24시간 뷰를 본다. "사고 전"이 정상이지 않을 수도 있다. 24시간 전에 이미 느려지고 있었다면 그 "느려진 상태"가 새 기준이 된다. 기준을 "완벽한 과거"로 잡지 말고 "작동했던 과거"로 잡는다.

얼마나 오래. 한 번의 확인으로 회복을 선언하지 않는다. 지표가 30분간 기준 안에 들어와 있어야 한다. 30분이라는 숫자는 관례다[추정]. 핵심은 지속성이다. 5분간 좋았던 지표는 5분 뒤에 다시 나빠질 수 있다. 특히 2장에서 트래픽을 줄였을 때, 트래픽을 다시 올리는 순간 지표가 다시

흔들릴 수 있다. 이 흔들림을 확인하는 시간이 30분이다.

세 요소를 한 줄로 쓴다. "에러율 1% 이하, 30분간 유지." 이 한 줄이 "되셨네"를 "회복했다"로 바꾸는 문장이다.

4.2 트래픽을 다시 올리는 절차

2장에서 50%를 차단했다면 회복의 마지막 단계는 그 차단을 푸는 일이다. 이 일을 "복구"라고 부르지 않는다. 복구는 새로운 변경이다. 그리고 새로운 변경은 2장의 규율, 즉 "한 번에 한 가지만, 전부 적는다"를 다시 따른다.

50%를 100%로 한 번에 되돌리지 않는다. 50%에서 75%로 올리고 10분을 기다린다. 지표가 기준 안에 있으면 100%로 올린다. 또 10분을 기다린다. 이렇게 두 단계가 한 단계보다 안전하다. 한 단계에서 문제가 나타나면 되돌리는 범위가 반이다.

여기서 "10분"은 회복 기준의 "30분"과 다르다. 30분은 "이 지표가 계속 좋은가"를 확인하는 시간이고, 10분은 "이 변경이 지표를 흔들었는가"를 확인하는 시간이다. 둘은 다른 질문을 한다. 둘 다 필요하다.

4.3 2차 사고는 복구 과정에서 일어난다

가장 위험한 행동은 사고가 거의 끝나갈 때 나온다. 긴장이 풀렸기 때문이다. "거의 됐는데"라는 상태는 판단력을 가장 낮추는 상태다. 2차 사고의 흔한 모양은 네 가지다.

데이터베이스 복구. "백업이 있으니 다시 복구하면 되겠지"라는 생각에서 온다. 이 생각이 위험한 이유는 백업이 사고 직전의 상태인지 확인하지 않기 때문이다. 사고 중에 쓰인 데이터가 백업에 포함되어 있다면 복구는 사고를 되살리는 행위다. 복구를 하기 전에 백업의 시점을 확인한다. 백업 시점이 사고 시작보다 앞다면 복구는 안전하다. 뒷이라면 복구는 금지다.

캐시 전체 삭제. "캐시가 문제겠지"라고 판단해서 캐시를 통째로 비우는 일. 이 행동이 위험한 이유는 캐시 삭제 직후 트래픽이 데이터베이스로 몰리기 때문이다. 데이터베이스가 캐시로 버텨왔다면 캐시가 사라진 순간 데이터베이스는 무너진다. 캐시를 지울 때는 트래픽을 먼저 줄인다. 2장의 순서를 거꾸로 하면 안 된다.

백필 실행. 사고 중에 쌓인 요청을 사고 후 일괄 처리하는 일. 이 요청들이 정상 상태에서는 1시간에 처리되던 것이라면 10분 만에 처리하려고 하면 데이터베이스에 폭주한다. 백필은 원래 속도로 돌린다. "늦어도 되는 일"을 "빨리 해야 하는 일"로 바꾸지 않는다.

설정 "반복" 적용. 사고 중에 두 번 적용했던 설정을 "확실히 하려고" 세 번째로 적용하는 일. 세 번째 적용은 새로운 변경이 아니다. 하지만 시스템에는 새로운 변경으로 보인다. 로그에는 세 줄이 남고 나중에 "어떤 줄이 효과였지"를 알 수 없게 된다.

네 가지의 공통점은 "거의 됐으니"라는 마음에서 온다. 이 마음은 사고의 마지막 10분을 가장 위험하게 만든다. 마지막 10분은 첫 10분과 같은 규율로 다룬다. 한 번에 한 가지만 전부 적는다.

4.4 원인을 몰라도 회복은 선언할 수 있다

이 책이 반복해서 묻는 질문이 있다. "원인을 알지 못했는데, 회복이라 말할 수 있나?" 답은 상황별로 다르다.

원인을 알고 고쳤다면 회복은 "원인이 제거됐다"는 뜻이다. 원인을 모르고 멈췄다면 회복은 "손실이 줄었다"는 뜻이다. 둘 다 회복이지만 강도가 다르다. 2장에서 외부 결제사 API 지연을 확인했지만 그 API가 왜 느린지는 몰랐다. 그 경우 "외부 API가 복구됐다"는 신호를 받을 때까지 내 서비스는 "일시적 완화" 상태다. 이 상태를 회복이라고 부르면 안 된다. 대신 "손실이 줄었다"고 부른다.

이 구분이 중요한 이유는 사용자에게 말하는 문장이 달라지기 때문이다. "복구 완료"라고 말하는 것과 "정상화 진행 중, 현재 지연 최소화"라고 말하는 것은 사용자의 기대를 다르게 만든다. 전자는 "다 됐다"를 기대하게 하고, 후자는 "아직 아니다"를 알게 한다.

1인 서비스에는 이 구분이 더 현실적이다. 대부분의 사고에서 당신은 원인을 "완전히" 알지 못한다. 외부 의존, 클라우드 제공사, DNS, 이 모든 것은 당신의 통제 밖이다. 통제 밖의 사고에서 "회복"의 기준은 서비스 재가동이다. 이 기준을 사고 전에 정해 두면 "회복을 선언해도 되나"를 고민하지 않아도 된다.

4.5 사고가 끝나고 나서 30분

종결 선언을 하고 사용자에게 세 번째 메시지를 보냈다면 사고는 끝난 것이 아니다. "정리"가 남아 있다. 정리는 사고 도중에 임시로 한 것들을 원래대로 되돌리는 일이다.

2장에서 건 레이트 리미트를 없앤다. 3장에서 띄운 임시 상태 페이지나 공지 메시지를 정리한다. 4장에서 트래픽을 단계적으로 올린 것의 최종 확인을 한다. 임시로 키운 로깅 레벨을 원래대로 내린다. 이 정리 중 하나라도 빠지면 다음 사고의 기준선이 오늘보다 높아진다.

정리의 순서는 "위험한 것부터"다. 레이트 리미트는 없애는 순간 트래픽이 원래대로 돌아온다. 이 순간이 2차 사고의 마지막 기회다. 정리를 시작하면 30분간 대시보드를 감시한다. 30분간 아무 일 없이 지나면 정리는 끝났다.

정리된 것을 시간선 파일에 적는다. "23:50 – 레이트 리미트 제거, 30분간 안정 확인." 이 한 줄이 이 사고의 마지막 문장이다. 이 문장이 있어야 이 사고는 "끝났다"는 의미가 생긴다.

4.6 검증은 명령으로 한다

"대시보드가 좋아 보인다"는 검증이 아니다. 보는 것은 관찰이다. 판단은 숫자를 비교하는 일이다. 그래서 종결 선언 전에 숫자를 직접 뽑아 비교한다.

- 에러율: 현재 10분의 값과 사고 전 24시간 뷰의 해당 시간 값을 비교한다.
- 지연시간: P95가 사고 전 범위 안에 들어왔는지 확인한다.
- 트래픽: 차단했던 요청이 다시 들어오고 있는지, 그중 실패하는 비율이 얼마인지 확인한다.
- 데이터: 정확성 축의 사고였다면 샘플 10건을 직접 확인한다. 자동 지표가 "정상"이라고 해도 10건 중 2건이 틀리면 정상이라고 말할 수 없다.

이 네 가지 비교를 시간선 파일에 한 줄로 남긴다. "23:40 - 에러율 0.8% (사고 전 0.5%), P95 420ms (사고 전 380ms), 트래픽 정상, 샘플 10건 중 1건 실패." 이 한 줄이 "회복했다"의 증거다. 증거가 없으면 회복은 희망이 된다.

4.7 종결 선언은 기준 도달과 안정 확인 이후에

"되셨네"를 종결 선언으로 쓰지 않는다. 종결 선언은 회복 기준의 세 요소를 모두 충족한 뒤에 한다. 지표가 기준 안에 들어오고 30분간 유지되고, 트래픽이 정상으로 돌아온 뒤. 이 셋이 다 되어야 "복구 완료"라고 말한다.

종결 선언의 문장은 3장의 세 번째 메시지와 동일하다. 무엇이었는지, 어떻게 고쳤는지, 현재 상태. 여기에 한 가지를 더한다. **남은 부분.** 완전히 복구되지 않은 부분이 있다면 그 부분을 함께 말한다. "결제는 완전히 복구됐고, 주문 알림은 여전히 10분 지연되고 있습니다." 이 한 줄이 사용자의 기대를 정렬한다. "다 해결됐지"라고 생각하고 10분 뒤에 알림이

안 와서 다시 불안해지는 것보다 “알림은 10분 지연된다”를 알고 기다리는 것이 낫다.

종결 선언을 한 뒤에 사고는 끝난다. 끝난 사고는 다음 단계로 넘어간다.

“종결 선언을 한 뒤에 정리를 한다”의 순서를 미리 정해 둔다. 종결 선언은 사용자에게 “다 됐다”를 말하는 시점이다. 정리는 내 시스템이 “다 됐다”를 유지하는 시점이다. 이 둘은 다르다. 종결 선언을 먼저 하면 사용자는 “다 됐다”고 믿는다. 그 믿음 위에 정리가 진행되므로, 정리 중에는 “다 됐다”는 상태가 유지된다. 정리를 먼저 하면 정리 중의 불안정한 순간이 “다 됐다”는 신호를 보내기 전에 온다. 이 순서를 지키면 사용자의 경험과 내 시스템의 상태가 일치한다.

4.8 종결 시에는 시간선만 남긴다

사고가 끝나면 “이제 원인을 분석하자”는 유혹이 온다. 이 유혹을 사고 중에 따르지 않는다. 사고 중에 하는 분석은 판단력이 낮은 상태에서 불완전한 정보로 길게 이어진다. 그 분석의 결론은 나중에 또 뒤집힌다.

대신 종결 시에 남기는 것은 두 가지다. **시간선 파일**과 **변경 목록.** 시간선은 3장에서 만든 파일이다. 변경 목록은 2장에서 “전부 적는다”로 만든 목록이다. 이 둘이 있으면 포스트모템은 이 둘로부터 시작한다. 사고 도중의 분석은 포스트모템의 입력이 된다. 출력은 포스트모템에서 나온다.

이 구분은 사고의 품질을 결정한다. 사고 중에 “원인이 A였다”고 결론짓고 그 결론을 바탕으로 변경을 하면 그 변경은 “A”라는 전제 위에 선다. 나중에 A가 아니면 그 변경은 왜곡된 것이다. 전제 없이 변경을 남기지 않는다. 전제는 포스트모템에서 확인된 뒤에 남긴다.

4.9 첫 30분의 가치

이 책이 다룬 것은 사고의 첫 30분이다. 30분 안에 한 것은 네 가지다.

1장에서 증상을 분류하고, 심각도를 세었다. 2장에서 확산을 멈추고 손실의 기울기를 바꿨다. 3장에서 시간선을 쓰고 사용자에게 세 개의 메시지를 보냈다. 4장에서 회복을 검증하고, 사고를 종료했다.

이 네 가지가 있으면 사고는 대응이 된다. 대응의 가치는 사고를 해결한 데 있지 않다. 해결은 대부분 외부 요인, 즉 "시간"이 한다. 대응의 가치는 그 시간 동안 손실을 얼마나 줄였는가에 있다. 100% 회복을 2시간 만에 하는 것보다 70% 회복을 30분 만에 하는 것이 사용자에게 낫다.

1인 개발자에게는 이 30분이 더 중요하다. 팀이 있으면 30분 뒤에 누군가가 이어간다. 1인이라면 30분이 곧 전체다. 전체를 30분으로 압축하는 것이 이 책이 할 수 있는 전부다. 그 압축의 핵심은 순서다. 분류, 멈춤, 통신, 검증. 이 순서를 지키면 30분이 일어난다. 이 순서를 잃으면 30분은 그냥 30분이다.