오류 예산

실패해도 괜찮은 한계를 정하는 기술

ThakiCloud (Hyojung Han)

오류 예산

실패해도 괜찮은 한계를 정하는 기술

ThakiCloud (Hyojung Han)

Contents

1 안정성의 값

"더 안정적으로 만들어라." 이것은 멈춤 점이 없는 목표. 하나를 고치면 둘이 보이고 둘을 고치면 셋이 보인다. 문제는 노력이 부족한 게 아니라 "얼마나 안정적이면 충분한가"에 답이 없기 때문. 혼자 프로덕션을 돌리는 개발자에게 이 질문은 자기가 제일 많이 묻는다. 오늘 밤에 새 기능을 올릴까, 아니면 모니터링에 밤을 쓸까. 숫자가 없으면 모든 선택은 추측이 되고 추측은 새벽 2시에 비싸진다.

이것을 끝내는 도구는 단순하다. 약속을 숫자로 적는 것. "이 서비스는 한 달의 99.9퍼센트 동안 사용자에게 동작한다"는 한 줄이 SLO, 서비스 레벨 목표. 마케팅 문구가 아니다. 시스템 주위에 긋는 선으로, 다음 시간을 어디에 쓸지와 쓰지 않을지를 정해 준다.

1.1 SLI와 SLO라는 두 단어

숫자를 쓰기 전에 두 단어를 알아야 한다. SLI, 서비스 레벨 지표는 실제로 재는 숫자다. 재는 대상은 사용자가 느끼는 경험, 즉 기계가 아니다. "응답이 200 상태 코드를 반환한 요청의 비율"은 SLI다. "CPU 사용률이 80퍼센트 이하인 서버의 비율"은 아니다. 전자가 사용자에 대한 계약이라면, 후자는 내부의 안도감에 불과하다.

SLO는 그 SLI에 지은 목표선이다. SLI가 재는 숫자라면 SLO가 약속하는 선이다. 한 달 동안 성공 응답 비율을 잴 때 99.95퍼센트였고 SLO가 99.9퍼센트였다면 약속은 지켰다. 아주 작은 여분도 남았다. 그 여분이 이 책의 주인공, 에러 버짓이다.

두 단어는 자주 뒤섞여 쓰이지만 역할은 다르다. SLI가 "무엇을 재는가"를 답하고 SLO가 "얼마를 약속하는가"를 답한다. 사용자가 무엇을 느끼는지

정하고 그것을 재는 숫자를 고르고 그다음에 선을 긋는 순서다. 순서를 바꿀 수 없다. 선을 먼저 긋게 되면 재기 쉬운 SLI를 고르게 되고, 계약은 사용자의 경험에서 인프라의 상태 쪽으로 빠져버린다.

1.2 SLO는 SLA가 아니다

한 번 더 헷갈리는 단어가 있다. SLA, 서비스 레벨 협정은 패널티가 붙은 계약이다. "SLO를 지키지 못하면 크레딧을 드린다"는 문장. SLO 자체는 그냥 선이다. 사용자에게, 그리고 나 자신에게 하는 약속. 1인 개발자에게는 처음부터 SLA가 꼭 필요하지 않다. 대부분 사용자는 보상 청구서까지 만들지 않으니까. 다만 SLO를 먼저 긋는 습관은 나중에 도움이 된다. 파트너 계약에서 99.9퍼센트를 지키라는 문장을 요구받게 되면, 숫자도 측정도 이미 손에 있다. 그때 SLA는 SLO 위에 붙이는 한 문장일 뿐이다.

1.3 데이터가 없는 SLO는 바람이다

종이에 적힌 약속은 데이터로 뒷받침되지 않으면 계약이 되지 못한다. 데이터에 필요한 것은 세 가지뿐이다. 성공과 실패를 기록할 로그나 탐침, 그것을 쌓아 둘 곳, 기간별 성공률을 줄여 내는 요약. 쌓아 둘 곳은 csv 파일 하나여도 된다.

1인 개발자에게 이 작업은 생각보다 작다. 1분마다 서비스를 호출해서 결과를 한 줄씩 파일에 붙여 놓는 스크립트 하나면, 월간 성공률도, 지난 1시간의 연소율도 셈할 수 있다. SLO의 선, 연소율의 선, 정책의 선은 전부 이 한 줄의 로그 위에 세워진다. 그래서 순서는 재는 것부터, 그다음 긋는 것이다. 측정도 없이 99.9퍼센트를 적어 두는 것은, 벽에 욕망을 붙이는 일이다.

1.4 왜 약속을 적어야 하는가

약속은 두 독자를 위해 쓴다. 첫 번째는 혼자 일하고 있다는 걸 모르는 사용자다. 두 번째는 나 자신, 더 정확히 말하면 금요일 밤에 결정을 내려야 하는 나다. 대시보드에 99.9퍼센트가 적혀 있으면 "얼마나 더 테스트해야 하나"에 답이 생긴다. 산술의 문제가 된다. 한 달 남짓한 실패 허용량이 며칠쯤 남았는가.

SLO가 또 하나 바꿔 놓는 것은 나 자신과의 대화다. "더 안정적으로"라고 하면 끝이 없는데, "99.9퍼센트"라고 하면 선이 있다. 선이 있으면, 선을 넘기 위한 이유를 만들게 된다. 그리고 이유를 만드는 일이 곧 판단 그 자체다.

이것이 SLO의 진짜 효용이다. 흐릿한 불안감을 산술로 바꿔 준다. 버짓이 이틀 만에 반씩 쓰였다면, 지금 큰 변경을 올리면 안 된다. 버짓이 일주일치라면 롤백 계획을 가지고 올려도 된다. 두 경우 모두 판단 기준이 "숫자"다. 1인 개발자에게 이 전환이 특히 중요한 이유가 있다. 불안을 받을 다른 사람이 없어서다.

SLO가 어디에 적히느냐도 정해 두자. 코드 저장소의 README 한 줄이면 된다. "SLO: 주문 성공률 99.9퍼센트, 한 달 창. 버짓 잔량은 대시보드에서 본다." 이렇게. 위치가 중요한 이유는, 판단의 순간에 눈이 가는 곳에 있어야 해서다. 대시보드 구석에 붙인 SLO는, 올릴지 말지 고민하는 그 순간에 보이지 않는다. README는 보인다.

1.5 9 하나당 가격은 얼마인가

신뢰성은 9의 개수로 세지만 9 하나는 같은 가격이 아니다. 한 달의 다운타임을 셈해 보자. 한 달을 30일로 잡으면, 99퍼센트는 약 7시간의 실패를 허용한다. 99.9퍼센트는 약 43분이다. 99.99퍼센트는 약 4분이다.

산술 자체는 지루하다. 지루하지 않은 것은 가격이다. 7시간에서 43분으로 가는 데 9가 하나 더 필요하고 43분에서 4분으로 가려면 또 하나 더 필요하다. 두 번째 9는 첫 번째 9보다 훨씬 비싸다. 구조를 바꾸기 때문이다. 중복을 넣고 리전을 나누고 폴오버 시간을 줄이고 폴오버가 실제로 동작하는지 확인하는 훈련까지 만들게 된다. 이것들은 하나같이 돈이 들고, 1인 개발자에게는 돈이 곧 밤이다.

구체적으로 비교해 보자. 99.9퍼센트를 지키려면, 탐침 한 개와 대시보드 한 줄과, 장애가 났을 때 나를 깨울 경로가 필요하다. 99.99퍼센트를 지키려면, 그 위에 두 번째 머신과 자동 전환과, 전환이 실제로 동작한다는 것을 증명하는 정기 훈련이 필요하다. 전자도 하루 이틀의 일이고 후자는 몇 주에 온콜의 부담까지 붙는다. 9는 같아 보이지만 밤의 밀도가 다르다.

그러므로 정직한 질문은 "다음 9은 얼마짜리이고, 내가 그 돈이 있는가"다. 대부분 개인 서비스는 99.5퍼센트에서 99.9퍼센트 사이에 머물면 충분하다 [추정]. 사용자가 하루에 몇 번 여는 계산기라면 99.99퍼센트는 사치다. 사용자가 결제를 거치는 경로라면 99.5퍼센트는 부족하다. 숫자는 사용자의 손실 규모를 따라야지, 자존심을 따라서는 안 된다.

9는 서비스끼리 비교해서도 안 된다. 결제 서비스의 99.9퍼센트와 메모 서비스의 99.9퍼센트는 같은 값이 아니다. 사용자의 손실 규모가 다르니까. 남의 SLO를 비교해서 자랑스러워하거나 부끄러워할 일은 없다. 중요한 비교는 하나다. 내 서비스 안에서, 다음 9이 가격만큼 값어치를 하는가.

흔한 실수는 99.99퍼센트처럼 그럴듯한 숫자를 대시보드에 적는 것이다. 지킬 수 없는 숫자를 적는 것은, 지킨 숫자를 적지 않는 것보다 나쁘다. 미래의 내가 그 숫자를 믿고 결정을 내리기 때문이다. 이 책이 가리키는 규율은 지킬 수 있는 숫자를 정하고, 그 숫자로 결정을 내리는 일이다.

1.6 작은 예시 하나

숫자를 구체적으로 만들기 위해 작은 서비스를 하나 만들자. 소규모 상점의 주문 API고 개발자는 한 명이다. 트래픽은 크지 않다. 하루에 약 5만 건의 요청이 온다고 하자 [추정]. 서비스는 가상 머신 한 대에 있고 데이터베이스는 같은 리전에 있다.

먼저 SLI를 고른다. 사용자에게 손실이 가장 큰 순간은 주문이 통하지 않는 순간이다. 그래서 핵심 SLI는 "주문 요청에 성공한 비율"이 된다. 한 달을 재면 99.92퍼센트다. 내가 약속하고 싶은 선인 99.9퍼센트보다 약간 위다. 그러면 SLO를 한 달 창으로 99.9퍼센트에 둔다.

그다음 에러 버짓이다. 1에서 99.9퍼센트를 빼면 0.1퍼센트. 한 달 요청이 150만 건이라면, 실패해도 되는 요청은 약 1,500건이다. 시간으로 환산하면 약 43분이다. 이 숫자는 작아 보이지만 1인 개발자에게는 오늘 밤의 행동을 정하기에 충분하다. 새 기능은 버짓이 80퍼센트 남은 날에 올리고 20퍼센트 남은 날에는 올리지 않는다. 데이터베이스 백업 훈련은 버짓이 30퍼센트 남은 날에 한다.

99.99퍼센트로 약속을 하나 더 올리면, 버짓은 약 4분으로 줄어든다. 그러면 가상 머신 한 대로는 안 된다. 두 번째 머신과 자동 폴오버, 그리고 그 폴오버가 실제로 동작하는지 확인하는 훈련이 필요해진다. 이건 몇 주일의 일이다. 이 예시가 보여주는 것은, 숫자를 고르는 일과 밤을 몇 주 쓸 일을 고르는 일이 같은 일이라는 것이다.

예시를 하나 더 붙이자. 같은 상점인데, 이번엔 파트너용 API가 있다. 결제대행사가 내 API를 부른다. 최종 사용자가 직접 느끼지 않는 것처럼 보이지만 파트너는 내 API가 느려지거나 실패하면, 상대 서비스의 장애로 기록한다. 그래서 이 API의 SLO는 일반 사용자를 위한 SLO보다 엄격하게, 또는 최소한 동일하게 써야 한다. 파트너에게 하는 약속은 결국 내 신용이고, 신용은 숫자로만 쌓인다.

1.7 데이터가 아직 없을 때

새 서비스라면 아직 재본 데이터가 없을 수 있다. 한 달을 재지 못했고 그러면 숫자를 고를 수가 없다. 이 경우 합성 탐침을 임시 SLI로 쓰자. 1분마다 서비스 밖에서 호출하고 성공과 시간을 기록한다. 그 데이터로 임시 SLO를 정하고 문서에 "이것은 임시다"라고 쓴다. 그리고 실 트래픽이 들어와 한 달치의 실제 데이터가 쌓이면, 그 숫자 기준으로 SLO를 다시 적는다.

여기서 중요한 것은 건너뛰지 않는 일이다. 재도 없이 SLO는 다시 바람이 된다. "99.9퍼센트일 것 같다"는 문장은 SLO가 아니다. SLO는 재어 낸 숫자 위에 그은 선이다.

그리고 탐침은 밖에서 해야 한다. 같은 서버 안에서 탐침을 돌리면 아무것도 증명하지 못한다. 서버가 죽으면 탐침도 함께 죽으니까. 작은 스크립트라도, 다른 머신에서, 인터넷에서 부르는 것이다.

1.8 오늘 밤의 결정

금요일 밤으로 돌아가자. 새 기능이 대기 중이다. 그런데 이번 달 에러 버짓의 40퍼센트가 이미 쓰였다. 선택지는 둘이다. 올리고 알아보거나, 버짓이 다시 차는 다음 주로 미루거나. SLO가 있으면 이것은 계산이 된다. 기능이 가져다줄 것과, 버짓에서 꺼내야 할 실패 허용량의 크기. 그리고 기능이 플래그 뒤에 있다면, 버짓의 일부만 써서 천천히 열어 볼 수도 있다.

이것이 다음 세 장에서 다룰 규율의 전체 그림이다. 2장에서는 재를 SLI를 고르고 SLO 숫자를 정한다. 3장에서는 버짓이 얼마나 빠르게 닳는지, 즉 연소율을 세우고 울릴 선을 긋는다. 4장에서는 그 숫자로 릴리스 판단을 내리는 규칙을 만든다. 모두 한 줄에서 시작한다. 약속을 숫자로 쓰는 것.

첫 단계는 재는 것이다. 사용자가 느끼는 동작 하나를 골라, 그 SLI를 한 달간

재고 숫자를 보자. 그 숫자가 지금 내 서비스가 무엇인지를 알려줄 것이다. 계약은 그곳에서 시작된다.

2 SLO를 고르는 법

1장은 계약이 재는 숫자에서 시작한다. 이 장은 그 숫자를 고르는 방법. SLI를 어떻게 고르느냐가 SLO가 실제 계약이 되는지, 장식에 그치는지를 정한다. 가장 흔한 실수. 재기 쉬운 것을 재는 것이다.

2.1 사용자 여정부터

서버를 잠시 치워 두고 사용자가 내 서비스로 실제로 하는 동작을 적는다. 메모 앱이라면 열고 쓰고 저장한다. 사진 공유 서비스라면 올리고 본다. API라면 파트너가 불러서 올바른 답을 받는다.

이중에서 실패했을 때 손실이 가장 큰 순간을 고른다. 메모 앱에서 저장이 안 되는 순간이 죽는 순간. 여는 것이 약간 느린 것은 짜증나는 것이지 죽는 것은 아니다. 그래서 핵심 SLI는 "저장 요청이 성공한 비율"이 되고, 전체 응답 속도는 되지 않는다. 모두를 똑같이 재게 되면 신호가 잡음 속에 묻힌다.

핵심 SLI는 하나만이어야 할 필요는 없다. 둘이나 셋도 된다. 다만 각각이 "이것은 어떤 손실에 대응하는가"에 답할 수 있어야 한다. 답이 없으면 보조 지표. 보조 지표는 유용하지만 계약은 보조 지표에 지어서는 안 된다.

방법도 단순하다. 동작 하나하나에 "이것이 실패하면 사용자는 무엇을 잃는가"라고 한 줄씩 적는 것이다. 잃는 것이 돈이거나 중요한 업무라면, 그 동작은 핵심 SLI 후보다. 잃는 것이 찝찝함 정도라면 보조다. 다 적어 보면 목록은 보통 짧다. 짧다는 것은 계약의 특징이지 결함이 아니다.

2.2 어떤 유형을 고를 것인가

SLI는 대략 네 가지 유형으로 나뉜다 [추정]. 가용성, 지연, 정확성, 신선도. 가용성은 "동작했는가"이고 지연은 "얼마나 빠르게 동작했는가"이며 정확성은 "답이 맞았는가"이고 신선도는 "데이터가 얼마나 새로운가".

대부분의 서비스는 가용성에서 시작한다. 그러나 서비스의 성격에 따라 유형별 무게가 완전히 달라진다. 검색 서비스라면 요청이 "성공"했는데 답이 틀렸다면, 그건 실패다. 여기서는 정확성이 계약이다. 피드 기반 서비스라면 게시가 1분 늦어져도 사용자가 알아챈다. 여기서는 신선도가 중요하다. 파트너가 부르는 API라면 지연 퍼센타일이 실질적 계약인 경우가 많다.

유형	예	사용자가 느끼는 것
가용성	성공 응답 비율	"깨졌다"
지연	응답 시간 95퍼센타일	"느리다"
정확성	결과 오류 비율	"답이 틀렸다"
신선도	데이터 지연 시간	"정보가 낡다"

유형을 섞어 쓰는 경우도 있다. 가용성 SLO 하나에 지연 SLO 하나. 이 조합은 흔하다. "동작한다"와 "빠르다"는 다른 약속이니까. 다만 정확성이나 신선도를 더 보태기 전에, 이미 지은 두 개를 한 달은 유지해 보자. 두 개를 유지하지 못하게 되면, 셋째는 지어질 자격이 없다.

원칙 하나. 계약의 숫자는 적어야 한다. 1인 서비스가 SLO를 네 개 적으면, 그 어느 것의 유지도 못 하게 된다. 하나부터 시작해서, 그것이 잡지 못 하는 손실을 실제로 겪었을 때 하나를 더 보탠다. 주먹에 담을 만큼이어야 한다.

2.3 한 달을 재는 일

SLO 숫자를 정하기 전에, 최소 한 달은 재야 한다. 이유는 표본 크기다. 요청 1,000건의 성공률은 불안정하다. 하루의 사건 하나가 1퍼센트를 움직인다. 요청 10만 건의 성공률은 기준값으로 쓸 만큼 안정적이다. 트래픽이 적은 서비스가 한 달에 10만 건에 못 미친다면 두 달을 쓴다 [추정].

지연 SLO를 재는 일은 다르다. 퍼센타일이 핵심이다. 평균은 꼬리를 숨긴다. 요청의 99퍼센트가 100밀리초이고 1퍼센트가 10초라면, 평균은 "빠르다"고 하지만 그 1퍼센트는 "깨졌다"고 느낀다. 그래서 지연 SLI는 퍼센타일로 적는다. 예컨대 95퍼센타일. 한 달 동안의 요청을 시간이 짧은 것부터 정렬했을 때, 95퍼센트 지점에 있는 값이 그 달의 95퍼센타일이다 [추정].

측정 기간에 대해 하나 더. 한 달은 요일과 주말의 리듬, 그리고 한 달 결제 사이클을 다 담는다. 일주일으로는 이것이 빠진다. 분기로는 너무 길어 판단의 기준값으로 쓰기 어렵다. 한 달은 리듬을 담고 결정도 할 수 있는 길이여서 기준값의 기본으로 쓰인다.

한 달의 시작점을 어디로 잡느냐도 정해 둔다. 달의 1일이 편하면 1일로, 배포 주기와 맞추고 싶으면 그 날짜로. 정해 놓으면 "이번 달이 거의 다 끝났으니 버짓 계산이 어정쩡하다"는 일이 사라진다. 시작점 없이는 연소율도 충족률도, 매달 어정쩡한 날짜 위에서 흔들리게 된다.

2.4 어디서 재느냐

측정 지점은 사용자에게 가까울수록 좋다. 웹 서비스라면 브라우저나 앱, API라면 파트너의 호출. 그러나 1인 개발자가 그 지점에 서 있기란 어렵다. 실용적인 선택지는 몇 개 있다.

게이트웨이나 로드 밸런서는 밖에서 들어오는 모든 요청을 보므로,

사용자에게 가장 가깝다. 애플리케이션 로그는 게이트웨이가 놓치는 것, 예컨대 200 응답 안에서 빈 데이터를 돌려준 것 같은 것을 잡을 수 있지만 사용자로부터는 한 칸 더 멀다. 그리고 합성 모니터링, 바깥의 탐침이 있다. 작은 프로그램이 1분마다 서비스 밖에서 서비스를 호출하고, 성공과 시간을 기록한다.

탐침은 1인 개발자에게 특히 유용하다. 값이 싸고 실 트래픽이 없는 시간에도 총 파업을 잡아 준다. 밤이나 주말에는 실 트래픽이 제로인 경우가 많다. 실 트래픽만으로 재는 SLI는 그 시간의 장애를 모른다. 탐침은 안다. 다만 탐침은 탐침이 하는 동작만 재지, 전체 사용자 경험을 대표하지는 않는다. 그래서 탐침은 보강이지 대체가 아니다.

여러 지점 중 하나만 고르라면, 유지할 수 있는 가장 가까운 지점을 고르자. 유지하지 못하는 측정은 없는 것보다 나쁘다. 그 위에서 지은 계약은 조용히 죽어버리기 때문이다.

2.5 목표 숫자를 정하기

흔한 질문이 있다. "몇 개의 9를 써야 하나." 답은 "먼저 재고 그다음 정한다"다. SLI를 한 달 동안 돌려 놓고 분포를 본다. 새 서비스라서 재본 지 2주밖에 되지 않았다면, 그 수치를 참고용으로만 쓰고 여유를 두고 정한다 [추정].

시작점은 실측 값보다 약간 위여야지, 먼 미래의 꿈이어서는 안 된다. 실측이 없는 서비스의 경우, 99.5퍼센트에서 99.9퍼센트가 흔히 쓰이는 시작 구간이다 [추정]. 100퍼센트를 쓰는 것은 함정이다. 100퍼센트 SLO는 에러 버짓이 0이라는 뜻이고, 실패 하나면 버짓이 모두 쓴다. 그러면 이 책의 "얼마를 써도 되는가"라는 규율이 동작하지 않는다. 버짓은 반드시 양수여야 한다.

서비스가 결제나 건강 관련 경로라면 99.9퍼센트 이상이 어울릴 수 있다

[추정]. 내부 도구나 트래픽이 적은 서비스라면 99퍼센트도 충분할 수 있다. 차이가 작아 보이지만 시간으로 바꾸면 99퍼센트는 한 달에 약 7시간의 버짓이고, 99.9퍼센트는 약 43분이다. 숫자를 고를 때마다 시간으로 환산해 비교하는 습관을 들이자.

지연 SLO는 퍼센타일로 쓴다. "요청의 95퍼센트가 300밀리초 이내"가 전형적인 형태다 [추정]. 퍼센타일 선택은 누구를 보호할 것인지의 계약이다. 95퍼센타일은 5퍼센트의 사용자에게는 느려도 된다는 뜻이다. 그 5퍼센트가 가장 서비스를 많이 쓰는 사람일 수 있다. 그러므로 "누가 뒤로 밀리는가"를 생각해 고르자.

선을 그은 뒤로는, 그 선을 다시 움직일 준비를 해 두자. SLO는 문신이 아니다. 규칙은 단순하다. 한 달이 지나면 실측 값과 SLO를 비교한다. 실측이 SLO보다 안정적으로 위라면, 선은 안전하다. 실측이 SLO 아래로 내려가는 달이 보이면, 선이 높다. 그리고 실측이 SLO보다 멀리 위에 있다면, SLO를 조금 낮추고 나온 여분을 릴리스 속도에 쓰는 신호다.

움직일 때는 옮긴 이유를 적어 둔다. "실측이 3개월 연속 99.99퍼센트였고 버짓이 한 번도 닳지 않아서 SLO를 99.9퍼센트에서 99.95퍼센트로 낮춘다." 이 한 줄의 기록이 나중에 그 결정을 돌아볼 때 얼굴을 붉히지 않게 해 준다. 이유 없는 숫자는 그럴듯해도 공상이며 공상은 다음 결정에서 다시 사고를 낸다.

트래픽이 아주 적은 서비스라면, 한 줄 더. 하루 요청이 몇십 건에 불과하면, 한 달의 실 트래픽은 몇천 건에 지나지 않고 성공률도 퍼센타일도 크게 흔들린다. 그 경우 합성 탐침을 주된 측정으로 쓰고, 실 트래픽은 보강으로만 쓴다. 그리고 창을 길게 잡는다. 요청이 하루 3건인 서비스에서 5분 창을 보면 아무것도 보이지 않는다. 표본이 적은 달에는 선을 촘촘하게 긋는 것보다, "전부 죽었다"는 순간을 정확히 잡는 것이 우선이다.

한 가지만 더. SLO의 창, 즉 결산을 하는 기간이다. 한 달 창은 전체 건강을 판단하는 데 쓰고 주간이나 하루 창은 이번 주 릴리스를 판단하는 데 쓴다.

3장의 연소율이 바로 이 창 위에서 작동한다. SLI를 한 달간 측정해 그 월간 값으로 SLO 충족 여부를 셈한다.

2.6 흔히 하는 실수

내부 실패를 사용자 실패로 세는 것. 재시도가 두 번째에 성공한 요청은, 사용자에게 실패가 아니었다. SLI는 사용자가 최종적으로 본 결과를 세야 한다. 중간 재시도까지 세게 되면 버짓이 너무 조여서 릴리스를 계속 동결하게 된다.

사용자가 보는 지점이 아닌 것을 재는 것. API 내부 처리가 200밀리초인데, 사용자의 화면에 이르기까지 경험은 800밀리초라면, 200밀리초 SLI는 나 자신에게 하는 거짓말이다. SLI는 사용자가 경험하는 가장 바깥쪽에서 재야 한다. 바깥쪽을 못 재는 경우, 최대한 가까운 지점을 재고 "이것은 대리 지표다"라고 문서에 명시한다.

실측값에 안주하는 것. "지나간 한 달에는 99.95퍼센트였으니까"는 SLO가 아니다. SLO는 앞으로의 달에 대한 선이다. 실측은 시작점이지 근거의 전부가 아니다. 근거의 절반은 "이 서비스에서 실패는 사용자에게 무엇을 주느냐"다. 같은 99.95퍼센트라도, 저장 서비스와 조회 서비스는 다른 선이다.

서버 오류만 세는 것. 5xx는 실패지만 200을 돌려 주면서 빈 데이터나 잘못된 데이터를 준 것도 실패다. 데이터를 돌려 주는 서비스라면 정확성을 SLI에 포함하거나, 최소한 보조 지표로 지켜야 한다. 상태 코드만 보면 버짓이 "건강하다"고 하면서 사용자가 쓰레기를 받는 일이 생긴다.

모든 것에 SLO를 짓는 것. 마이크로서비스마다, 엔드포인트마다, SLO 20개. 그러면 스프레드시트를 만들고 있는 것이다. 1인 개발자의 SLO는 한두 개, 대시보드를 보지 않고도 기억할 수 있는 숫자여야 한다.

2.7 이 장의 산출물

이 장의 작업물은 한 장의 문서다. 사용자 여정, 손실이 가장 큰 순간, 고른 SLI와 그 측정 지점, SLO 목표와 창, 그 숫자를 고른 이유를 쓴다. 이유를 못 쓰면 그 숫자는 아직 정해지지 않은 것이다.

채워진 예시 한 장을 보여주자. 파트너용 이미지 리사이즈 API. 핵심 SLI 하나, 리사이즈 요청 성공 비율. 측정 지점은 게이트웨이. SLO는 한 달 창 99.9퍼센트. 두 번째 핵심 SLI, 응답 시간 95퍼센타일. SLO는 한 달 창 500밀리초 이내. 이유는 파트너의 처리가 시간 제약이 있어서 실패의 손실과 지연의 손실을 따로 계약해야 하기 때문이다.

다음 장에서 이 문서를 알람으로 바꾼다. 버짓이 얼마나 빨리 닳는지, 그리고 언제 눈을 뜨어야 하는가. 그것이 연소율이다.

3 연소율

2장은 계약이 적힌 한 장의 문서로 끝났다. 이 장은 그 문서를 움직이는 장치로 바꾼다. SLO가 주는 것은 한 달 치 실패 허용량. 그러나 1인 개발자가 매일 부딪히는 질문은 "한 달 합계가 괜찮은가"가 아니라 "지금 일어나는 이 실패가 심각한가"다. 그 답은 버짓을 쓰는 속도, 즉 연소율에서 나온다.

3.1 버짓을 숫자로

먼저 에러 버짓을 구체적인 숫자로 만든다. SLO가 99.9퍼센트이고 창이 한 달이라면, 버짓은 한 달의 1퍼센트다. 한 달을 30일로 잡으면 시간으로 환산해 약 43분. SLI가 성공 응답 비율이라면, 1퍼센트 이내의 실패 허용 개수이기도 하다.

SLI가 요청 단위라면, 버짓은 시간 말고도 개수로 환산할 수 있다. 한 달 요청이 10만 건이고 SLO가 99.9퍼센트라면, 허용 실패는 한 달에 100건이다. 그러면 하루는 약 3건, 일주는 약 23건. 이 환산은 대시보드가 시간을 보여주는 대신 개수를 볼 때 유용하다. "오늘 오류 12건"은 "하루에 4일 치 버짓을 쓴" 것과 같은 문장이다. 시간과 개수, 둘 중 눈에 들어오는 단위로 버짓을 적어 두자.

이 숫자는 "얼마나 실패할 것인가"의 예측이 아니라, "얼마까지 실패해도 되는가"의 한계다. 이 차이는 중요하다. 전자는 기대치이고 후자는 한계선이다. 에러 버짓은 목표가 아니라 한계다. "버짓이니까 조금은 깨도 된다"고 쓰는 사람은 없다. 버짓은 한계를 감시하면서 계속 움직일 수 있게 주는 여백이다.

3.2 연소율이란

연소율은 버짓을 소진하는 속도를 배수로 쓴 것이다. 기준은 1배다. 1배로 쓰면 한 달 마지막에 버짓이 정확히 0이 된다. 급한 일이 아니지만 그 달은 붉게 끝난다. 10배로 쓰면 버짓은 약 3일로 다 쓴다. 더 이상 한 달의 문제가 아니라, 지금 이 순간 서비스가 깨지고 있다는 문제다.

연료 계정에 비유할 수 있다. 1배 연소는 느린 새는 것으로, 일주일에 한 번 확인하면 된다. 10배 연소는 3일 만에 탱크가 비는 것으로, 운전을 멈춰야 한다. 문제는 연료 계정에 바늘이 없으면 얼마나 빨리 빠지는지 알 수 없다는 것이다. 연소율이 바로 그 바늘이다.

측정 방법은 단순하다. 최근 창에서 SLI를 보고 실제 실패율을 허용 실패율(1에서 SLO를 뺀 값)로 나눈다. 지난 1시간의 성공률이 99.0퍼센트이고 SLO가 99.9퍼센트라면, 지난 1시간에 허용 실패의 10배를 썼다는 뜻이다. 1퍼센트를 0.1퍼센트로 나눈 것이 10배다.

연소율은 트래픽의 크기에 영향을 받는다. 요청이 적은 시간에는 실패 두 건도 연소율을 크게 움직이지만 사용자가 실제로 잃은 것은 적다. 반대로 요청이 몰린 시간에는 같은 비율도 큰 손실이다. 그래서 연소율만으로 판단을 끝내지 말고 실패의 절대 개수도 함께 보는 것이 안전하다. 연소율은 속도를 말해 주고, 개수는 크기를 말해 준다.

연소율	버짓 소진 시점	성격
1배	한 달 끝	정상
3배	약 10일	만성 문제
10배	약 3일	지금 사건

소진 시점은 30일을 연소율로 나눈 값이다. 신비한 숫자가 아니라 나눗셈이다.

3.3 계산 하나를 끝까지

실 숫자로 계산을 하나 끝내자. 한 달 SLO 99.9퍼센트, 한 달 요청 약 10만 건의 서비스. 허용 실패율은 0.1퍼센트라, 버짓은 한 달에 약 100건의 실패다.

지난 1시간을 보자. 요청 400건에 실패가 10건이었다. 실제 실패율은 2.5퍼센트. 허용 실패율 0.1퍼센트로 나누면, 연소율은 25배. 이 속도로 가면 한 달 버짓은 얼마나 가는가. 30일을 25로 나눈 1.2일이다. 즉, 이 상태가 그대로면 이번 달 버짓은 하루 반이면 소진된다. 이건 이번 달 할 일 목록에 올릴 문제가 아니라, 오늘 밤에 올릴 문제다.

같은 숫자를 2배로 해 보자. 지난 1시간 실패율이 0.2퍼센트라, 연소율 2배. 버짓은 15일까지 간다. 붉은 신호지만 화정은 아니다. 이 속도가 한 달을 채우면 99.9퍼센트 아래로 그 달이 끝나지만 오늘 원인만 고치면 약속은 지킬 수 있다.

이 계산이 보여주는 것은, 두 상황이 대시보드에서는 비슷해 보이지만 성격이 완전히 다르다는 것이다. 차트의 빨간 점은 25배가든 2배가든 같은 점으로 보인다. 연소율은 "화재"와 "붉은 신호"를 점으로부터 분리하는 것이다.

3.4 경고 두 개

연소율로 경고를 두 개 만든다. 이것이 이 장의 핵심이다.

빠른 쪽은 심각한 사건을 위한 짧은 창 경고다. 한 달 버짓을 며칠 안에 다 쓸 만한 속도, 예컨대 5분짜리 짧은 창에서 10배 이상 연소할 때 울린다 [추정]. 이 경고를 울리면 뜻하는 바는 "사용자가 지금, 무시할 수 없는 속도로 잃고 있다"는 것이다. 1인 개발자는 눈을 떠서 보러 가야 한다. 이 경고가 페이지, 즉 지금 당장 일어나야 하는 일이다.

느린 쪽은 만성 문제를 위한 긴 창 경고다. 2배, 3배의 느린 연소라도 며칠만 이어지면 그 달을 갉아먹는다. 예컨대 6시간에서 하루짜리 긴 창에서 3배 이상 연소할 때 울린다 [추정]. 이 경고를 울리면 뜻하는 바는 "뭔가가 천천히 잘못되고 있다. 이번 주에 고쳐라"는 것이다. 이 경고는 티켓이지 페이지가 아니다. 새벽 2시에 일을 멈추지 않지만 백로그 맨 위에 놓아야 한다.

왜 둘인가. 임계값 하나로는 만성 문제를 놓치거나, 짧은 흔들림마다 늑대 소리를 하게 된다. 30분 동안 5배로 뛴 실패는 배포 실수일 가능성이 크고, 순간적이다. 그것에 매번 경고를 울려서 매번 눈을 떴다가, 경고는 의미를 잃는다. 반대로 6시간을 이어가는 3배 연소는 진짜 문제다. 그래서 짧은 창은 높은 임계값을, 긴 창은 낮은 임계값을 쓴다.

두 경고를 모두 세운 뒤, 한 달만 지켜 보자. 어느 쪽이 얼마나 자주 울렸고 울린 뒤 실제로 일어난 것이 무엇이었는지. 그 기록이 다음 조정의 근거가 된다. 경고를 세우는 일은 한 번으로 끝나지 않고 이 한 달의 기록이 첫 조정을 만든다.

두 경고의 이름도 정해 둔다. 페이지는 "사건"이라 부르고 티켓은 "만성"이라 부른다. 이름이 곧 판단이다. 사건 급의 경고를 "확인"이라 부르면 확인처럼 취급하게 되고, 만성 문제를 "사건"이라 부르면 새벽에 혼란을 자초한다. 경고의 이름을 심각도에 맞추는 일도 설계의 일부다.

3.5 바늘이 흔들리지 않게

창 하나짜리 측정값은 불안정하다. 10분짜리 성공률은 배포 하나, 트래픽의 작은 움직임에 크게 뛴다. 흔한 해결법은 창을 두 개 함께 보는 것이다. 짧은 창과 긴 창이 모두 임계값을 넘을 때만 울리게 하면, 순간 흔들림은 걸러지고 지속된 문제는 남는다 [추정]. SRE 실무에서 멀티 윈도우 연소율이라 불리는 표준 기법이다.

예를 들어, 5분 창 10배와 1시간 창 10배를 동시에 만족할 때만 페이지를 울리게 하자. 짧은 흔들림은 전자는 만족하지만 후자는 못 넘기므로 울리지 않는다. 진짜 사건은 둘 다 넘는다. 만성 문제는 30분 창 3배와 6시간 창 3배로 잡는다.

목적은 잡음을 줄이는 것만이 아니다. 경고를 지키게 하기 위함이다. 울리면 아무것도 아니었던 페이지는, 울리지 않는 것보다 나쁘다. 다음 진짜 페이지도 무시하게 되기 때문이다. 1인 개발자에게는 오경고의 비용을 나눌 온콜 로테이션이 없다. 그 비용은 전부 나에게 온다.

3.6 창을 고르는 일

창의 길이는 트래픽에 맞춰야 한다. 5분 창은 분에 몇 천 건의 요청이 오는 서비스에서 동작한다. 그런데 분에 몇 건밖에 요청이 오지 않는 서비스라면, 5분에는 표본이 손에 꼽고 실패 한두 건에 연소율이 크게 뛴다. 그 경우에는 짧은 창을 30분으로, 긴 창을 하루로 늘린다 [추정].

실용적인 규칙은, 창 안에 최소 수백 건의 요청이 들어와야 한다는 것이다 [추정]. 그보다 표본이 적은 창은 서비스를 재는 것이 아니라 잡음을 재는 것이다. 트래픽이 적은 서비스는 창을 길게, 임계값을 낮게. 트래픽이 많은 서비스는 창을 짧게, 임계값을 높게. 창은 한 번 설정하고 잊는 값이 아니라, 바늘이 얼마나 흔들리는지를 보며 조정하는 값이다.

3.7 페이지가 울렸을 때

페이지가 울리면, 10분 안에 대응을 시작한다. 순서는 고정해 두자.

먼저 대시보드의 세 숫자를 본다. 현재 연소율, 남은 버짓, 충족률. 페이지가 말한 내용이 숫자와 일치하는지 확인한다. 가끔은 경고 설정이 어긋나서 스스로를 울리는 경우가 있다. 그럴 때는 경고를 잠그고 어긋난 이유를 적어 둔다. 그냥 넘어가면 다음에 같은 일이 되풀이된다.

다음으로 최근 변경을 본다. 내가 며칠 전이나 몇 시간 전에 무엇을 올렸고, 무엇을 바꿨는가. 1인 개발자에게 대부분의 사건의 원인은 몇 시간 전의 내 손이다. 연소율이 오른 시점이 배포 시점과 맞물리면, 조사하기 전에 먼저 롤백한다.

그다음 판단이다. 롤백으로 풀리면, 사건을 닫고 조사는 오프라인으로 옮긴다. 롤백으로 풀리지 않으면 플래그를 내린다. 그것으로도 풀리지 않으면, 지금 하고 있는 일을 멈추고 살아 있는 채로 조사한다. 조사하면서 한 행동을 한 줄씩 적어 둔다. 다음 아침의 내가 덜 졸린 상태로 이어서 조사할 때, 그 몇 줄이 밤새의 나를 대신해 준다.

1인 개발자의 페이지 수신지는 결국 내 폰이다. 경고를 울리는 채널은 스마트폰 알림이어도 메일이어도 상관없다. 중요한 것은, 울린 뒤 10분 안에 내가 그 알림을 열어 보겠는가다. 열지 않는 경보는 경보가 아니다. 채널의 신뢰도를 점검하는 일도, 월간 점검의 작은 일부다.

이 순서는 큰 서비스를 위한 것이 아니다. 바로 나 혼자인 경우를 위한 것이다. 나 혼자일 때, "먼저 무엇을 하는가"가 가장 비싼 질문이니까.

3.8 대시보드에는 세 숫자

1인 개발자의 대시보드에는 숫자가 세 개면 된다. 지금 연소율, 이번 달에 남은 버짓(시간 또는 퍼센트), 그리고 이번 달 SLO 충족률. 그 이상은 대시보드가 아니라 보고서다.

이 숫자가 답하는 질문은 명확하다. 연소율이 "지금 화재인가"를 답하고 버짓 잔량이 "남은 것이 얼마인가"를 답하며 충족률이 "약속을 지키고 있는가"를 답한다. 경고를 울리면 대시보드에서 이 숫자를 읽고 판단한다. 숫자가 맥락을 주고, 내가 결정을 준다. 맥락 없는 경고는 고함일 뿐이다.

이 대시보드는 고비용의 모니터링 시스템일 필요가 없다. 한 파일에 줄을 쌓고 그 줄을 읽어서 세 값을 보여주는 스크립트면 된다. 1인 개발자의

대시보드는 작을수록 유지된다.

예를 들어 "저장 성공률이 지난 1시간 동안 99.0퍼센트를 유지하고 있다(10배 연소). 이번 달 남은 버짓은 약 30분이다"는 한 줄이 맥락이다. 이 문장에서 판단이 따라온다. 1시간 전 배포를 롤백할 것인가, 플래그를 내릴 것인가, 아니면 이번 주 상위에 올릴 만성 버그인가.

3.9 이 장의 산출물

이 장이 끝나면 손에 경고를 두 개가 있어야 한다. 한 달 버짓이 며칠 안에 위험해지는 순간에 울리는 것과, 무언가 천천히 잘못되고 있음을 알려 주는 것. 그리고 둘 다 남은 버짓을 보여 준다. 그것이 전부다.

다음 장에서 이 숫자를 릴리스 판단에 붙인다. 버짓이 있으면 쏘고 버짓이 없으면 멈추는 것. 에러 버짓 정책의 전부다. 너무 단순해 보이지만 단순한 규칙을 계속 적용하는 것이야말로 1인 개발자의 프로덕션을 추측에서 빼내는 일이다.

4 쏘고 멈추고 고치다

숫자를 고르고 경고를 세웠다. 이제 이 장은 규율 자체다. 언제 올리고 언제 멈추는가. 도구는 한 장의 문서, 에러 버짓 정책.

4.1 정책은 결정 규칙이다

"에러 버짓을 잘 쓰세요"는 정책이 아니다. 바람이다. 정책은 회의를 열지 않아도 결정이 되는 문장이다. "이번 달 에러 버짓 사용이 X퍼센트 미만이면 릴리스를 자유롭게 한다. X퍼센트를 넘으면 버짓이 회복할 때까지 릴리스를 동결한다." X는 내가 고르는 숫자로, 버짓의 절반 근처가 흔히 쓰인다 [추정].

핵심은 "만약"을 숫자로 쓰라는 것이다. 그러면 판단은 더 이상 기분이 아니다. 월요일, 버짓은 80퍼센트 남았다. 올리자. 목요일, 사건이 일어나서 버짓이 20퍼센트까지 줄었다. 동결한다. 금요일, 기능이 다 준비되어도 20퍼센트라면 올리지 않는다. 이것이 산술이다. 버짓이 20퍼센트일 때 계속 올리면, 남은 한 달을 실패 하나에 건 것이니까.

1인 개발자에게는 정책이 더 단순해야 한다. 정책을 강제할 위원회가 없기 때문이다. 10초에 읽을 수 있을 만큼 짧게 쓴다. 내용이 각 한 줄씩인 문서다.

- 올림 임계값: 버짓이 이 이상 남아 있으면 자유롭게 올린다(예, 50퍼센트)
- 동결 임계값: 버짓이 이 아래면 올림을 멈춘다(예, 30퍼센트)
- 동결 시 행동: 동결되면 먼저 하는 일(롤백, 플래그 내림, 아니면 그냥 대기)

이보다 길면 아무도 읽지 않고 아무도는 곧 나다. 정책 문서는 패닉 상태에서 꺼내 보기 위한 것이다.

예외 조항도 같이 써 두자. 예외가 없는 정책은 힘으로 깨지게 된다. 보안 패치나 데이터 훼손 수정은 버짓이 동결되어 있어도 해야 할 일이다. 정책에 적혀 있지 않으면, 나는 정책을 깨는 선택을 하게 되고 한 번 깨진 정책은 다시 권위를 잃는다. 그래서 예외를 정책 안에 편입한다. "보안 패치와 데이터 훼손 수정은 동결에 걸리지 않는다. 대신 적용 뒤 24시간 안에 가능하면 롤백을 검토하고, 이유를 기록한다." 예외 조항은 정책의 일부다.

예외 조항이 제대로 쓰였는지 보는 시험이 있다. 동결 선 아래에서 보안 패치가 필요한 순간에, 정책 문서에서 답을 찾았는가. 한참을 생각해야 한다면, 아직 정책이 완성되지 않은 것이다.

4.2 동결 체크리스트

버짓이 동결 선 아래로 내려가면, 기분 대신 목록을 돌린다.

"돌릴 수 있는 변경이 있는가"가 첫 질문이다. 버짓이 빠지기 시작한 시점이 배포 직후라면, 먼저 롤백하고 그다음에 원인을 찾는다. 원인은 하루를 기다려도 된다. 사용자는 기다려 주지 않는다.

"내릴 수 있는 플래그가 있는가"가 다음 질문이다. 플래그 뒤에 숨은 새 기능은 반쯤 열린 문이다. 버짓이 낮아졌으면 문을 닫는다. 기능이 버짓을 함께 쓰고 있다는 뜻이다. 1인 서비스에는 기능 사이의 벽이 없다. 한 기능이 쓰면 서비스 전체의 버짓이 빠진다.

"모든 배포를 멈춰야 하는가"가 마지막 질문이다. 롤백도 플래그도 소용없으면, 문제는 핵심에 있다. 그때는 전부 동결하고 고친다. 1인 개발자에게 가장 나쁜 상태다. 만드는 시간을 고치는 데 쓰기 때문. 그런데 바로 이 규율이 존재하는 이유이기도 하다. 이 상태에 덜 자주 오게 하려는 것이지, 오지 않게 하려는 것이 아니다.

4.3 채워진 예시

가상의 서비스의 정책을 한 장으로 쓰면 이렇게 된다.

- SLO: 저장 성공 비율, 한 달 창 99.9퍼센트
- 페이지: 5분 창과 1시간 창 모두 10배 이상
- 티켓: 30분 창과 6시간 창 모두 3배 이상
- 올림 임계값: 버짓 잔량 50퍼센트 이상
- 동결 임계값: 버짓 잔량 30퍼센트 미만
- 동결 시 행동: 롤백과 플래그를 먼저 확인, 소용없으면 배포 전부 동결

여섯 줄이다. 릴리스를 결정할 때 볼 것은 이 중에서 두 줄뿐이다. "버짓이 얼마 남았는가"와 "동결 선 위인가"다. 위면 올리고 아래면 동결 체크리스트를 돌린다. 정책의 일은 이 두 줄의 판단을 빠르게 하는 것이다.

위 예시는 SLO가 하나였다. SLO가 둘이나 셋인 서비스에는 버짓이 그만큼 많다. 그러면 어느 버짓이 릴리스에 걸리느냐. 규칙을 단순하게 둔다. 변경이 건드리는 SLI의 버짓을 쓴다. 리사이즈 경로만 바꾸는 변경은 리사이즈 SLI의 버짓을 보고, 인증이나 데이터베이스 같은 공용 지형을 바꾸는 변경은 가장 많이 쓰인 버짓을 본다. 보수적이지만 정직하다.

반대로 "가장 타이트한 버짓이 항상 지배한다"로 규칙을 쓰지 않는 이유는, 늘 조인 하나의 SLO에게 영원히 묶이기 때문이다. 한 달에 수백 건의 요청만 오고 SLO가 99.9퍼센트인 조회 API 하나만 있어도, 그 조회 버짓은 항상 타이트하고 모든 릴리스가 동결된다. 규칙은 버짓을 쓰는 법이지, 릴리스를 막는 법이 아니다.

4.4 일주일 걸어보기

이 숫자로 일주일을 걸어 보자. 월요일, 버짓 잔량 90퍼센트. 새 기능을 플래그 반쯤 열고 올린다. 수요일 오후 3시, 사건이다. 데이터베이스 연결

풀이 꽉 차서 저장이 20분간 실패했다. 페이지가 울리고 플래그를 내리고 연결을 되살린다. 그 사건이 버짓의 30퍼센트를 썼다. 잔량 60퍼센트.

목요일, 버짓 잔량은 여전히 올림 선 위다. 다만 수요일 사건의 원인은 아직 고치지 못했다. 고치는 일은 연결 풀 설정을 바꾸는 일로, 작다. 지금 올리면 버짓이 충분하다. 그래서 플래그 단위로 올려 확인한다. 금요일, 티켓이 울린다. 6시간을 이어가는 3배 연소. 사건은 아니고 만성 문제다. 로그를 보면 특정 파트너의 재시도가 오류를 쌓고 있다. 사용자의 저장에는 영향이 없지만 SLI는 그것을 실패로 센다.

여기서 판단이 하나 필요하다. 파트너의 재시도를 고칠 것인가, SLI의 셈법을 고칠 것인가. 재시도가 끝내 성공한다면, 그것은 사용자 실패가 아니었다. SLI를 최종 결과 기준으로 고치고 일주일간 검증한다. 이게 맞으면 만성 연소는 사라지고, 버짓은 회복된다. 이 예시가 보여주는 것은, 정책이 올림의 스위치일 뿐만 아니라, 계약 자체를 고치는 장치이기도 하다는 것이다.

일주일이 끝났다. 월요일 90퍼센트에서 시작해, 수요일 사건의 30퍼센트, 금요일 만성 문제의 셈법 고침으로 끝났다. 이번 주는 정책을 한 번도 깨지 않았다. 깨지 않았을 때보다, 깨지 않았다는 사실을 알아차린 순간이 더 중요하다. 알아차리지 못하면, 다음 달에 언제 깨졌는지도 모르게 되기 때문이다.

4.5 매달의 점검

규율의 마지막 조각은 월간 점검이다. 1인 개발자의 점검은 아무도 없는 점검, 다음 달의 나와 하는 10분이다. 결정을 적는다.

"이번 달 SLO를 지켰는가." 답이 맞다면, 남은 버짓이 얼마였는지 본다. 남는 버짓이 매달 크다면, SLO가 너무 엄격했거나 릴리스가 너무 느렸을 수 있다. 답이 아니라면, 버짓이 어디로 갔는지 본다. 어떤 사건, 어떤 만성 문제, 어떤

배포가 썼는가.

"경고가 올바르게 울렸는가." 울리고서 아무것도 아니었던 페이지가 있었는가. 페이지 없이 지나간 화재가 있었는가. 둘 다 임계값이나 창이 어긋난다는 신호다. 숫자를 조정한다.

"정책을 지켰는가." 버짓이 동결 선 아래인데 올렸다면, 왜 그랬는가. 이유가 정당한 것이라면, 정책에 그 특수 사례를 적어 두자. 이유가 기분이라면, 실패한 것은 규율이다. 그것도 적는다. 적지 않는 점검은 느낌이다.

점검은 한 줄로 끝난다. "다음 달에는 ~한다." "다음 달에는 버짓이 매달 다 쓰이니까 올림 선을 50퍼센트에서 40퍼센트로 낮춘다." 혹은 "다음 달에는 버짓이 절대 안 닳으니까 SLO를 99.9퍼센트에서 99.95퍼센트로 느슨하게 한다." 한 줄의 결정. 그것이 점검의 전부다.

점검은 월말 10분이면 된다. 세 가지 질문에 답하고 한 줄을 적고 끝낸다. 10분을 지키지 못하는 점검은 없느니만 못하다. 길어진 점검은 다음 달로 미뤄지고 미뤄진 점검은 결국 사라진다. 사라진 점검이 쌓인 달에는, 버짓이 어디로 갔는지도, 정책을 언제 깨었는지도 모르게 된다.

4.6 버짓이 절대 안 닳을 때

정책은 나쁠 때를 위한 것처럼 보인다. 그런데 너무 잘되고 있을 때도 신호가 있다. 한 달이 끝나고 버짓이 매번 90퍼센트 이상 남는다면, 어딘가 어색하다.

가능성 하나. SLO가 너무 엄격하다. 99.9퍼센트를 약속했는데 실측이 매달 99.99퍼센트를 보이면, 그 약속은 장식이었다. SLO를 99.95퍼센트로 낮추고 나온 여분을 릴리스 속도에 쓴다. 이것이 이 책이 말하는 트레이드오프의 실체다. 안정성과 속도는 같은 버짓의 양면이다.

가능성 둘. 릴리스가 너무 느리다. 한 달에 한 번만 올리면 버짓은 당연히

남는다. 왜 올리기를 이렇게 두려워하는가. 작은 것을 자주 올리는 것은 큰 것을 드물게 올리는 것보다 버짓을 적게 쓴다. 깨질 때 터지는 범위가 작고 롤백이 쉽기 때문이다.

두 경우 모두 같은 행동을 요한다. 버짓을 보고 한 가지 결정을 바꾸는 것. 건강한 시스템은 버짓이 중간쯤에서 오르내리는 시스템이다. 항상 다 쓰는 버짓은 지나치게 약한 시스템의 버짓이고, 절대 안 닳는 버짓은 지나치게 겁난 시스템의 버짓이다.

4.7 버짓이 매달 다 쓸 때

반대의 신호는 매 달 버짓이 0으로 끝나는 경우다. 이것은 "계약을 다시 쓰자"의 상황이다.

먼저 SLI를 점검한다. 사용자가 실제로 느끼는 것을 재고 있는가. 재시도가 숨겨 주는 순간 실패까지 세고 있다면, 버짓이 너무 조여서다. 사용자가 보지 못하는 대리 지표를 재고 있다면, 안 아픈 실패에 버짓을 내고 있는 것이다.

SLI가 맞다면 SLO를 본다. 99.9퍼센트를 약속했는데, 서비스 구조가 그것을 지킬 수 없다면, 그 약속은 내 손을 넘어선 것이다. 단일 머신에 폴오버도 없는 구조라면, 구조를 올리는 것보다 약속을 낮추는 편이 정직할 수 있다. 1인 개발자에게는 그것이 흔히 정직한 선택이다. 99.5퍼센트를 지킬 때가, 99.9퍼센트를 매달 깨는 것보다 값지다.

마지막으로 범위를 본다. 버짓이 허락하는 것보다 많은 기능을 만들고 있는가. 에러 버짓은 곧 기능 예산이다. 새 기능 하나마다 실패할 방법이 하나씩 추가된다. 기능을 버짓을 쓰는 속도로 계속 붙이면, 서비스는 커지면서 안정성은 줄어든다. 그때는 붙이는 것을 멈추고 한 달을 안정성에 쓰는 달이 된다.

4.8 전체 그림

뒤를 돌아보자. 이 책이 만든 것은 세 가지뿐이다.

- 사용자가 느끼는 동작에 대한 SLI 한두 개, 그 위의 SLO 한 장
- 짧은 창과 긴 창으로 된 연소율 경고 두 개
- 임계값 두 개와 동결 시 행동을 적은 정책 한 장

작다. 1인 개발자가 다 소유할 수 있을 만큼 작다. 이 책의 가치는 도구가 붙여 준 판단이다. "오늘 밤에 올릴까"에 답이 생겼고, 그 답은 내 기분이 아니다. "이 경고가 중요한가"에도 답이 생겼는데, 이번엔 추측이 아니다. "SLO를 낮출 것인가"에까지 답이 생겼다. 그 답은 자존심이 아니다.

이 책의 마지막 과제는 나한테 있다. 2장의 문서와 3장의 경고를 들고 내 정책 한 장을 쓰는 것이다. 릴리스를 결정할 때 보이는 곳에 붙여 두기. 그리고 숫자가 멈추라고 할 때, 멈추는 것이다. 그것이 규율의 전부다. 스스로 그은 선에서 실패하고 그 선에서 다시 움직이는 것. 1인 개발자가 프로덕션을 돌리는 방식이다.