TECH BRIEF

MSAP.ai

GSLB 기반 Active-Active 구성

하드웨어·클라우드·K8GB 3계층 심층

하드웨어·클라우드·K8GB 3계층 심층 비교와 클라우드 네이티브 Active-Active DR 설계 가이드

2026-07-13 · 작성자: OPENMARU · msap.ai

목차

GSLB 기반 Active-Active 구성 백서

1장: 개요 — Active-Active와 GSLB의 정의

GSLB와 Active-Active는 이후 모든 장이 전제로 삼는 두 개념입니다. 먼저 두 개념을 정의하고 각각의 역할 범위를 구분합니다. GSLB는 트래픽 계층에서 사이트를 선택하는 도구이며, Active-Active의 성패는 그 위 계층인 데이터 정합성과 페일오버 지연에서 갈립니다. 두 개념을 처음부터 구분해 두면, 제품 선택으로 Active-Active가 완성된다는 흔한 오해를 초반에 걷어 낼 수 있습니다.

1.1 GSLB의 정의와 역할 범위

이 절은 GSLB의 정의를 이전「GSLB 이해하기」백서에서 그대로 계승하고, GSLB가 수행하는 일과 수행하지 않는 일의 경계를 분명히 합니다. 정의를 앞선 백서와 일치시켜야 이후 7~9장의 제품 논의와 용어 충돌이 생기지 않습니다.

1.1.1 GSLB란 무엇인가 (이전 백서 계승)

GSLB(Global Server Load Balancing, 광역 서버 부하 분산)는 지리적으로 분산된 복수의 데이터센터 또는 클라우드 리전 간에 네트워크 트래픽을 지능적으로 분배하는 기술입니다 [S1]. 단일 데이터센터 내부에서 서버 간 트래픽을 나누는 일반 로드밸런서(SLB, Server Load Balancer)와 달리, GSLB는 데이터센터 단위, 즉 사이트(Site) 간의 글로벌 분배를 담당합니다. 분배의 단위가 서버가 아니라 사이트라는 점이 두 기술을 가르는 첫 번째 기준입니다.

GSLB의 핵심 동작 원리는 DNS(Domain Name System, 도메인 이름 시스템) 기반 라우팅입니다 [S1]. 클라이언트가 특정 도메인에 접속하려고 DNS 질의를 발생시키면, GSLB는 각 사이트의 헬스 상태와 가용성, 현재 부하, 클라이언트의 지리적 위치, 관리자가 설정한 라우팅 정책을 종합 평가하여 최적 사이트의 IP 주소를 DNS 응답으로 반환합니다. 이후 클라이언트는 반환된 IP로 직접 연결하며, GSLB 자체는 실제 데이터 전송 경로(Data Path, 데이터 경로)에 관여하지 않습니다. SLB가 모든 요청을 자신을 경유시켜 실시간으로 분배하는 인라인 방식인 반면, GSLB는 DNS 질의 시점에만 개입하는 아웃오브밴드 방식입니다.

구분	SLB (사이트 내부)	GSLB (사이트 간)
분배 범위	데이터센터 내부 (로컬)	사이트 간 (글로벌)
동작 방식	트래픽이 LB를 경유 (인라인)	DNS 응답으로 IP 반환 (아웃오브밴드)
개입 시점	모든 요청에 실시간 분배	DNS 질의 시점에만 개입
동작 계층	L4(TCP/UDP) / L7(HTTP)	DNS 계층
대표 기능	서버 부하 분산, 세션 유지, SSL 종료	사이트 선택, DR 페일오버, 지리 기반 라우팅

GSLB는 헬스체크, 지리적 근접성, 레이턴시, 가중치 같은 라우팅 정책으로 고가용성과 재해복구(DR, Disaster Recovery)를 실현하며, 현실에서는 하드웨어 어플라이언스, 클라우드 매니지드 서비스, 쿠버네티스 네이티브(K8GB) 세 갈래의 제품군으로 구현됩니다 [S5]. 어떤 제품군을 쓰더라도 DNS 응답을 동적으로 생성해 사이트를 선택한다는 원리는 동일하며, 기존 DNS 인프라를 활용하므로 별도의 네트워크 장비 변경 없이 도입할 수 있다는 공통 이점을 가집니다.

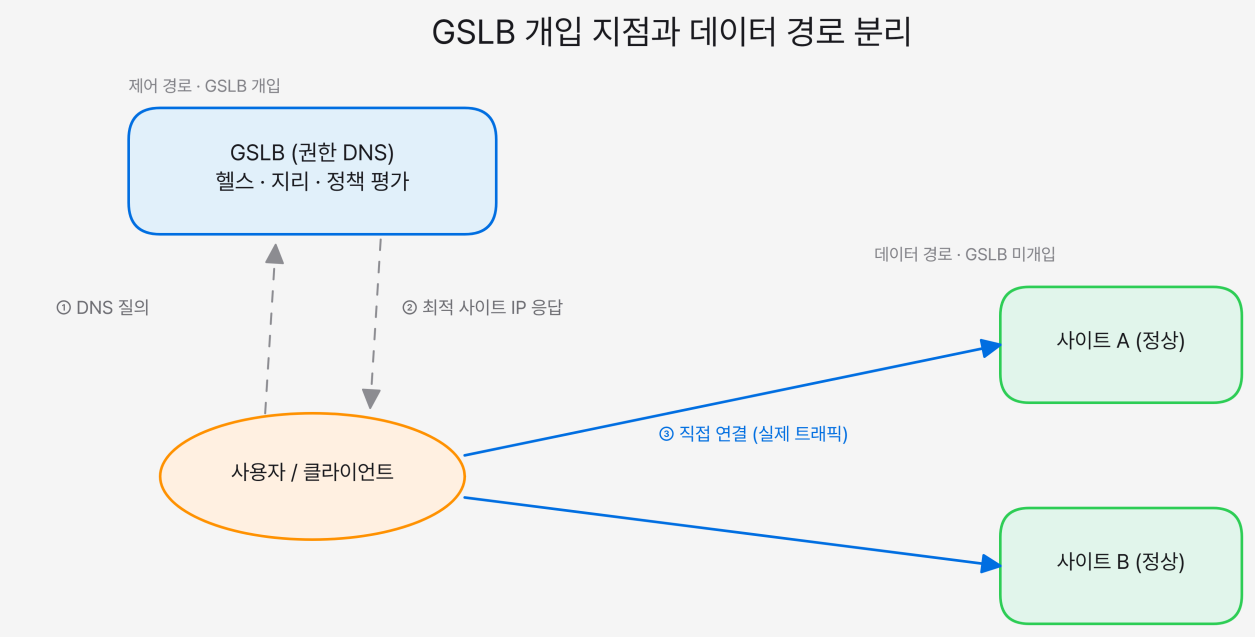

그림. DNS 질의부터 사이트 IP 응답까지 GSLB가 개입하는 지점과 데이터 경로가 분리되는 구조.

1.1.2 GSLB의 역할과 한계 — 트래픽만 분산한다

GSLB가 실제로 수행하는 일은 세 가지로 압축됩니다. 첫째, DNS 응답을 조작해 트래픽을 사이트 단위로 분산합니다. 둘째, ICMP·TCP·HTTP·애플리케이션 레벨의 다층 헬스체크로 각 사이트의 가용성을 주기적으로 판정합니다 [S1]. 셋째, 장애가 감지된 사이트를 DNS 응답에서 제외해 정상 사이트로 트래픽을 자동 전환합니다. 이 세 가지가 GSLB의 책임 범위 전부이며, 모두 트래픽 계층에서 완결됩니다.

문제는 GSLB가 수행하지 않는 일에 있습니다. GSLB는 사이트 간 데이터의 정합성을 보장하지 않으며, 사용자 세션의 일관성도 책임지지 않습니다. Active-Active 구성에서 양쪽 사이트가 동시에 쓰기를 받으면 발생하는 쓰기 충돌(Write Conflict)과, 네트워크 분단 시 양쪽이 각자 주도권을 갖는 Split-Brain(분할 뇌) 문제는 GSLB의 관할 밖입니다 [S4]. GSLB는 어느 사이트로 요청을 보낼지만 결정할 뿐, 그 사이트에 도착한 쓰기가 다른 사이트의 데이터와 어떻게 화해하는지는 데이터 복제 계층이 별도로 설계해야 합니다.

구분	GSLB가 수행하는 것	GSLB가 수행하지 않는 것
트래픽	사이트 단위 DNS 분산	세션 상태의 사이트 간 이전
감지	다층 헬스체크·장애 판정	데이터 복제 지연 감지

구분	GSLB가 수행하는 것	GSLB가 수행하지 않는 것
전환	장애 사이트 자동 제외	쓰기 충돌·Split-Brain 해소
정합성	해당 없음	데이터 일관성 보장

이 경계가 중요한 이유는, GSLB 제품을 아무리 잘 골라도 데이터 계층 설계가 없으면 Active-Active가 성립하지 않기 때문입니다. GSLB는 DNS 질의 시점에만 개입하는 Stateless(무상태) 기술이라, 연결이 맺어진 이후의 세션 상태나 데이터 상태를 알지 못합니다 [S1]. 따라서 GSLB로 트래픽을 반씩 나눈 순간부터, 그 트래픽이 만들어 내는 데이터를 어떻게 일관되게 유지할 것인가 하는 진짜 난제가 시작됩니다. 이 난제는 5장(데이터 복제와 정합성)에서 본격적으로 다루며, GSLB는 그 난제를 푸는 도구가 아니라 난제를 드러내는 출발점이라는 점을 이 항에서 분명히 해 둡니다.

1.2 Active-Active 구성의 개념과 배경

이 절은 Active-Active의 정의와 정량적 가치를 제시하고, 이 구성이 지금 폭넓게 검토되는 배경을 연결합니다. 정의 직후에 실제 RTO 수치를 붙여 추상적 설명을 피합니다.

1.2.1 Active-Active의 정의와 가치

Active-Active(액티브-액티브)는 두 개 이상의 사이트가 동시에 트래픽을 처리하는 구성입니다 [S4]. 모든 사이트가 실시간으로 서비스에 참여하고, GSLB가 사이트 간 트래픽을 가중치나 근접성 기준으로 분산합니다. 반대편의 Active-Passive(액티브-패시브)는 평상시 하나의 Primary 사이트만 트래픽을 처리하고 나머지 Standby 사이트는 장애가 나야 활성화되는 구성으로, 대기 사이트는 데이터 동기화만 수행하며 유휴 상태로 남습니다.

Active-Active의 가치는 정량 지표로 드러납니다. 한 사이트가 멈춰도 나머지 사이트가 이미 트래픽을 받고 있으므로 별도의 기동 시간 없이 흡수하며, 이때 RTO(Recovery Time Objective, 복구 목표 시간)는 수초에서 수십초 수준입니다 [S4]. 동기 복제를 병행하면 RPO(Recovery Point Objective, 복구 목표 시점)는 0에 가깝게 유지됩니다. 반면 Active-Passive는 Standby를 기동하고 워밍업하는 시간이 필요해 RTO가 수분에서 수십분에 이르고, 비동기 복제 구간만큼 데이터 손실이 발생해 RPO가 0보다 커집니다. 이 RTO·RPO 격차가 두 구성을 가르는 가장 실질적인 판단 근거이며, 상세 비교는 2장에서 다룹니다.

가치는 세 방향으로 요약됩니다. 우선 상시 가용성입니다. 재해가 나도 서비스가 멈추지 않는다는 무중단 관점의 이점이 가장 큽니다. 다음은 리소스 활용률입니다. 모든 사이트가 상시 트래픽을 받으므로, Standby 인프라를 놀리는 Active-Passive보다 투자 대비 효율이 높습니다. 마지막은 상시 검증된 DR입니다. Active-Active 사이트는 이미 운영 중이므로 별도의 DR 테스트로 정상 기동 여부를 확인할 부담이 줄어듭니다 [S4]. 다만 이 이점들은 양방향 쓰기 충돌과 Split-Brain 위험, 그리고 CAP 정리(Consistency·Availability·Partition tolerance) 상의 상충을 떠안는 대가로 얻어지며, 이 균형은 이 백서 전체를 관통하는 주제입니다 [S10].

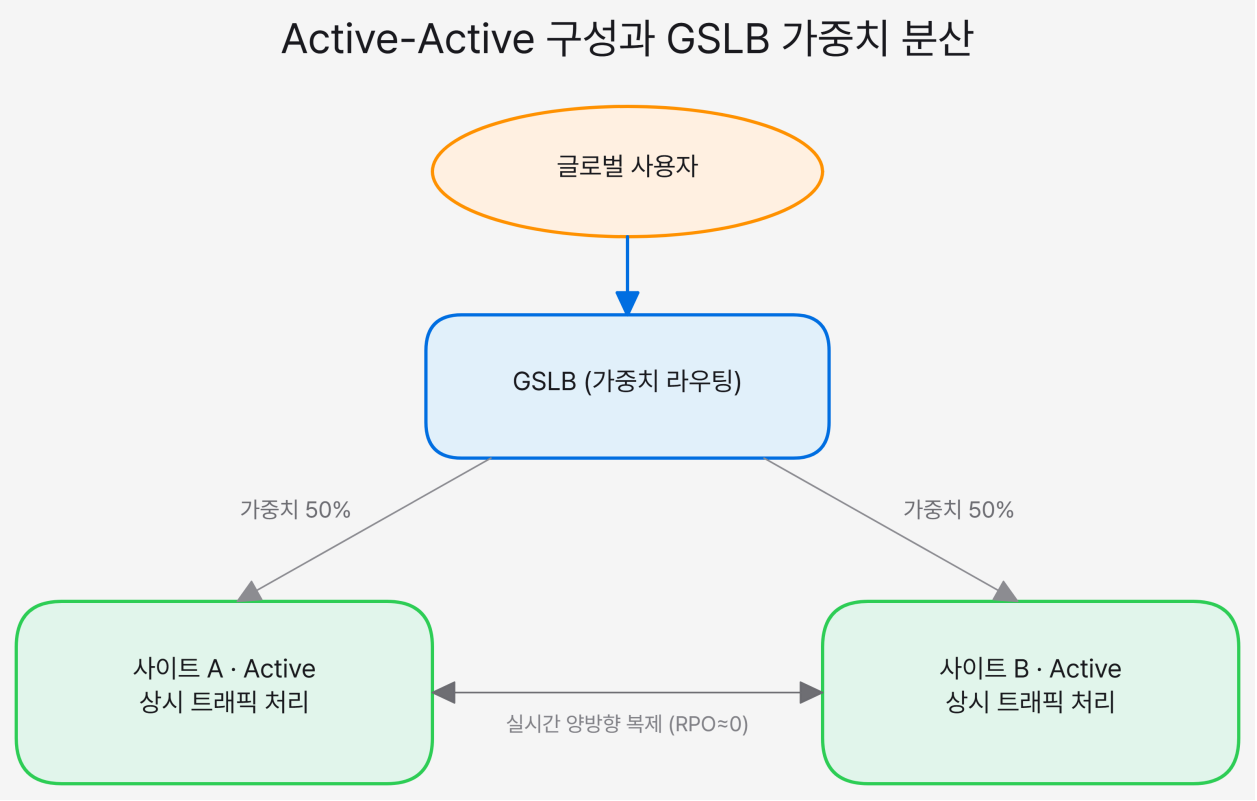

그림. 두 사이트가 동시에 트래픽을 처리하는 Active-Active 구성과 GSLB의 가중치 기반 분산.

1.2.2 상시 가용성 요구의 부상 (Why now)

Active-Active 전환을 지금 검토하게 만드는 동인은 네 가지로 정리됩니다. 첫째는 멀티리전·멀티클라우드 기반 상시 가용성 요구입니다. 서비스가 국내를 넘어 글로벌로 확대되면서, 전 세계 사용자에게 낮은 지연으로 무중단 서비스를 제공해야 한다는 기대치가 높아졌습니다 [S1]. 사용자를 근접 사이트로 유도하면서 동시에 어느 사이트가 죽어도 서비스가 이어지게 하려면, 모든 사이트가 상시 살아 있는 Active-Active가 자연스러운 귀결이 됩니다.

둘째는 대형 클라우드의 리전·AZ(Availability Zone, 가용 영역) 장애가 반복적으로 현실화된다는 점입니다. 단일 리전에 인프라를 몰아 둔 구성은 그 리전이 흔들리면 함께 멈춥니다. 특정 사고를 단정하기보다, 대규모 클라우드에서 리전 단위 장애가 주기적으로 발생해 왔다는 사실 자체가 멀티사이트 상시 가용성 설계를 밀어 올리는 압력으로 작용합니다 [S10]. Active-Passive로도 이에 대응할 수 있으나, 전환 지연과 미검증 Standby의 위험 때문에 무중단이 필수인 서비스는 Active-Active를 택하게 됩니다.

셋째는 규제입니다. 금융권은 전자금융감독규정에 따라 RTO 2시간 이내의 재해복구 체계를 의무적으로 갖추어야 하며, 이커머스 역시 99.99% SLA(Service Level Agreement, 서비스 수준 협약)를 보장하기 위해 멀티사이트 DR을 구축합니다 [S1]. RTO 목표가 공격적일수록 수동 전환에 의존하는 구성으로는 목표를 맞추기 어려워, 자동 페일오버가 전제된 Active-Active의 근거가 강해집니다.

넷째는 쿠버네티스(Kubernetes) 표준화입니다. 컨테이너 오케스트레이션의 사실상 표준인 쿠버네티스를 여러 리전과 클라우드에 걸쳐 운영하는 기업이 늘면서, 클러스터 간 트래픽을 네이티

브하게 분배하는 GSLB 수요가 커졌습니다 [S6]. CNCF(Cloud Native Computing Foundation) 프로젝트인 K8GB가 주목받는 배경이 여기에 있으며, 이는 9장에서 상세히 다룹니다. 이 네 동인은 독자가 "왜 지금 검토해야 하는가"에 답하고 내부 설득 근거를 확보하는 출발점이 됩니다.

1.3 클라우드 네이티브 관점에서 본 Active-Active DR과 GSLB

앞의 두 절이 GSLB와 Active-Active를 일반론으로 정의했다면, 이 절은 이 백서가 두 개념을 다루는 관점을 분명히 합니다. 오늘날 멀티사이트 Active-Active DR과 GSLB는 대부분 클라우드 네이티브(Cloud Native) 환경 위에서 설계됩니다. 워크로드가 컨테이너와 쿠버네티스(Kubernetes)로 표준화되고, 인프라가 코드로 관리되며, 서비스가 잘게 나뉘어 배포되는 흐름이 Active-Active를 구현하는 방식 자체를 바꾸어 놓았기 때문입니다 [S6]. 따라서 GSLB를 단독 장비가 아니라 클라우드 네이티브 스택의 트래픽 진입 계층으로 이해할 때, 이후 장들의 제품·설계 논의가 하나의 그림으로 꿰어집니다.

1.3.1 클라우드 네이티브란 무엇이며 DR에 어떤 함의를 갖는가

클라우드 네이티브는 CNCF(Cloud Native Computing Foundation, 클라우드 네이티브 컴퓨팅 재단)가 정의한 시스템 구축 방식으로, 컨테이너로 패키징한 마이크로서비스를 쿠버네티스 같은 오케스트레이터가 동적으로 배치하고, 인프라를 불변(Immutable) 자원으로 다루며, 선언적(Declarative) API로 원하는 상태를 기술하는 접근입니다 [S6]. 핵심 특징은 탄력성, 회복탄력성, 관측 가능성, 그리고 자동화입니다. 애플리케이션이 특정 서버나 리전에 고정되지 않고 어디서든 동일한 정의로 재구성되므로, 한 사이트가 사라져도 같은 워크로드를 다른 사이트에서 즉시 세울 수 있습니다.

이 특성은 재해복구의 전제를 바꿉니다. 전통적 DR은 물리 서버·수작업 구성·정적 IP에 묶여 Standby 사이트를 유지하고 검증하는 비용이 컸고, 그래서 평상시 대기 사이트를 놀리는 Active-Passive가 현실적 타협이었습니다. 반면 클라우드 네이티브 환경에서는 사이트가 곧 선언적으로 정의된 클러스터이므로, 동일한 정의를 복수 리전에 펼쳐 상시 가동하는 Active-Active의 진입 장벽이 크게 낮아집니다 [S6]. 다만 이 이점은 상태(State)를 애플리케이션 밖으로 밀어냈다는 전제 위에서만 성립하며, 상태 계층의 정합성 문제는 여전히 남아 5장에서 별도로 다룹니다.

다음 표는 전통 인프라와 클라우드 네이티브 환경에서 DR 특성이 어떻게 달라지는지를 대비한 것입니다.

관점	전통 인프라 DR	클라우드 네이티브 DR
사이트 정의	물리 장비·수작업 구성	선언적 매니페스트(코드)
재구축 속도	느림 (수작업 프로비저닝)	빠름 (오케스트레이터가 재배치)
상태 처리	서버에 국소화되기 쉬움	외부 저장소로 분리 (Stateless 지향)
기본 구성 경향	Active-Passive 타협	Active-Active 진입 장벽 낮음
페일오버 신호	수동 점검·별도 모니터링	Readiness Probe·관측성 파이프라인

1.3.2 클라우드 네이티브가 Active-Active를 자연스럽게 만드는 이유

클라우드 네이티브 스택의 여러 구성 요소가 Active-Active의 난제를 나눠 맡습니다. 첫째는 Stateless 마이크로서비스입니다. 상태를 외부 저장소로 밀어낸 서비스는 어느 사이트에서 실행되든 동등하므로, 트래픽을 양쪽에서 받는 구성이 자연스럽습니다. 세션과 상태를 실제로 어떻게 다루는지는 6장에서 상세히 다룹니다. 둘째는 쿠버네티스 멀티클러스터입니다. 사이트가 곧 클러스터가 되면서, GSLB는 여러 클러스터 앞단에서 어느 클러스터로 사용자를 보낼지 결정하는 North-South(외부와 클러스터 사이) 트래픽 진입 계층을 담당합니다. 이 역할을 쿠버네티스 네이티브로 구현한 것이 CNCF 프로젝트 K8GB이며 9장에서 다룹니다 [S6][E29].

셋째는 선언적 구성과 GitOps입니다. GSLB 정책과 DR 구성을 매니페스트로 기술해 Git에 두면, ArgoCD·Flux가 이를 여러 리전에 동일하게 적용하고 구성 드리프트를 자동 교정합니다. DR 구성이 코드가 되므로 재현성과 감사 추적이 확보되고, 사이트 재구축이 반복 가능한 절차가 됩니다 [S6]. 넷째는 서비스 메시입니다. GSLB가 North-South 진입을 맡는 동안, Istio·Linkerd 같은 서비스 메시는 클러스터 내부와 클러스터 간의 East-West(서비스와 서비스 사이) 통신을 담당해 로컬리티 인지 라우팅, mTLS(상호 TLS 인증), 재시도와 서킷 브레이킹을 제공합니다 [E35][E36]. 두 계층은 대체 관계가 아니라 트래픽의 방향을 나눠 맡는 보완 관계입니다. 다섯째는 관측성입니다. Prometheus·OpenTelemetry 기반 지표가 헬스체크와 페일오버 판단의 신호원이 되며, 이 주제는 11장의 운영 검토로 이어집니다.

이 구도를 한 줄로 요약하면, 클라우드 네이티브 환경에서 Active-Active는 세 계층의 협업으로 성립합니다. GSLB가 사이트 간 트래픽 진입을 맡고, 서비스 메시가 서비스 간 통신을 맡으며, 외부화된 상태 저장소와 복제 계층이 데이터 정합성을 맡습니다. GSLB는 이 스택에서 트래픽 진입 계층에 놓인 한 부분이며, 그 위아래 계층이 함께 설계되어야 Active-Active가 완성됩니다. 이 백서가 GSLB 제품 비교(7~9장)에 앞서 트래픽 분산(3장), 헬스체크(4장), 데이터 정합성(5장), 세션(6장)을 먼저 다루는 이유가 여기에 있으며, 그중 쿠버네티스 네이티브 방식이 클라우드 네이티브 원칙에 가장 밀착한 구현이라는 점은 9장에서 구체화합니다 [S6][S10].

2장: Active-Active vs Active-Passive 의사결정

Active-Active와 Active-Passive는 재해복구(DR, Disaster Recovery, 재해로 중단된 서비스를 복구하는 체계) 구성의 두 갈래이며, 둘 중 무엇을 고를지는 취향이 아니라 목표 지표가 정합니다. 이 장은 두 구성을 RTO(Recovery Time Objective, 목표 복구 시간)·RPO(Recovery Point Objective, 목표 복구 시점)·리소스 활용률·운영 복잡도라는 축으로 정량 비교하고, 서비스 등급별로 어느 쪽이 정당화되는지 판단 기준을 제시합니다. GSLB(Global Server Load Balancing, 지리적으로 분산된 사이트 간 트래픽을 분배하는 기술)는 두 구성 모두에서 트래픽 전환을 담당하지만, 구성 선택 자체는 데이터 계층의 정합성 요구가 좌우한다는 점을 이 장에서 분명히 합니다.

2.1 두 구성의 근본 차이

두 구성의 차이는 "모든 사이트가 동시에 일하는가, 한 사이트만 일하고 나머지는 대기하는가"라는 한 줄로 요약되지만, 그 한 줄에서 RTO·RPO·비용·복잡도가 연쇄적으로 갈라집니다. 이 절은 그 연쇄를 수치로 대비하고, Active-Active가 얻는 가용성 이득의 반대편에 어떤 설계 부채가 쌓이는지를 함께 제시합니다.

2.1.1 RTO/RPO·리소스 활용률 비교

RTO는 장애 발생 후 서비스를 다시 정상화하기까지 허용되는 시간이고, RPO는 장애 시점 기준으로 되돌아가 허용되는 데이터 손실의 크기입니다. 이 두 지표가 구성 선택을 직접 규정합니다. Active-Active는 모든 사이트가 실시간으로 트래픽을 처리하므로 한 사이트가 빠져도 나머지가 즉시 흡수하고, 그 결과 RTO가 수초에서 수십초 수준으로 짧습니다. 동기 복제를 병행하면 RPO는 0에 수렴합니다. 반면 Active-Passive는 평상시 Primary 사이트만 트래픽을 받고 Standby 사이트는 데이터 동기화만 수행하며 대기하므로, 장애 시 Standby 기동·DNS 전파·데이터 정합성 확인에 시간이 쌓여 RTO가 수분에서 수십분에 이릅니다 [S4]. 단방향 비동기 복제를 쓰면 마지막 복제 이후의 데이터가 유실될 수 있어 RPO가 0보다 커집니다.

리소스 활용률의 차이도 큽니다. Active-Active는 이중으로 구축한 인프라를 상시 가동하므로 투자 대비 활용률이 높고, 사용자와 지리적으로 가까운 사이트로 트래픽을 분산해 지연시간까지 줄입니다. Active-Passive는 Standby 인프라가 평상시 유휴 상태로 남아 투자 효율이 낮은 대신, 초기·운영 비용 자체를 억제할 수 있습니다 [S10]. 또한 Active-Active는 실제 트래픽을 처리하는 사이트가 곧 검증된 복구 경로이므로 별도의 DR 훈련 없이도 복구 능력이 상시 확인되는 반면, Active-Passive는 Standby가 정상 기동하는지 정기적으로 검증하지 않으면 실제 장애 때 기동 실패 위험을 안고 갑니다.

다음 표는 두 구성의 핵심 특성을 종합 비교한 것입니다 [S4][S10].

항목	Active-Active	Active-Passive
트래픽 처리	모든 사이트가 동시에 실시간 처리	Primary만 처리, Standby는 대기
리소스 활용률	높음 (전 인프라 상시 활용)	낮음 (Standby 유휴)

항목	Active-Active	Active-Passive
RTO	수초~수십초	수분~수십분
RPO	0에 근접 (동기 복제 시)	비동기 복제 간격만큼 손실 가능
데이터 동기화	실시간 양방향 복제 필수	단방향 비동기 복제
Failover 방식	자동 (GSLB가 장애 사이트 즉시 제외)	수동/반자동 (전환 트리거 필요)
데이터 일관성	CAP 정리상 Trade-off 존재	단일 쓰기 지점으로 유지 용이
비용	높음 (이중 인프라 상시 운영)	상대적으로 낮음
아키텍처 복잡도	높음 (충돌 해소·분산 합의 필요)	상대적으로 낮음
장애 복구 신뢰성	상시 검증됨 (운영 중)	정기 검증 절차 필요

이 표에서 읽어야 할 핵심은 RTO·RPO 열입니다. 목표 RTO가 수초 단위이고 RPO가 0이어야 하는 서비스라면 Active-Active 외에 선택지가 없고, RTO가 수분·수십분까지 허용되는 서비스라면 Active-Passive로 목표를 충족하면서 비용과 복잡도를 절약할 수 있습니다. 즉 표의 나머지 항목은 이 두 지표를 정한 뒤에 따라오는 결과입니다.

2.1.2 데이터 충돌 위험과 운영 복잡도

Active-Active가 짧은 RTO와 높은 활용률을 얻는 대가로 떠안는 것은 데이터 계층의 복잡도입니다. 두 사이트가 동시에 쓰기를 받으므로 동일 레코드에 대한 양방향 동시 쓰기, 즉 Write Conflict(쓰기 충돌)가 구조적으로 발생할 수 있습니다. 이를 방치하면 사이트마다 다른 값이 확정되어 데이터가 어긋나므로, 충돌을 해소하는 별도 전략이 반드시 설계에 들어가야 합니다. 대표적으로 타임스탬프 기준으로 마지막 쓰기를 채택하는 LWW(Last-Write-Wins), 수학적으로 병합 가능한 CRDT(Conflict-free Replicated Data Types, 충돌 없는 복제 데이터 구조), 그리고 Paxos·Raft 같은 분산 합의 알고리즘이 있으며, 각각 단순함과 데이터 유실 위험 사이에서 트레이드오프를 가집니다 [S4]. 이 선택은 5장의 데이터 복제 설계로 이어집니다.

더 무거운 위험은 Split-Brain(분단 뇌)입니다. 사이트 간 네트워크가 단절되면 양쪽이 서로를 장애로 오인하고 각자 독립적으로 쓰기를 계속 수락해, 재연결 시 회복 불가능한 데이터 불일치가 남습니다. 이를 막으려면 과반 노드가 합의할 때만 쓰기를 허용하는 Quorum, 제3의 위치에 중재자를 두는 Witness/Arbiter 노드, 소수파 노드의 쓰기를 강제 차단하는 Fencing 같은 방어선을 미리 갖춰야 합니다 [S4]. 이 방어선은 홀수 노드 배치와 전용 저지연 네트워크를 전제하므로 그 자체가 추가 비용입니다.

운영 복잡도는 정성적 수사가 아니라 "무엇을 추가로 설계해야 하는가"의 목록으로 드러납니다. Active-Active 도입 시 최소한 다음 항목이 새로 설계 대상이 됩니다.

- 양방향 복제와 Write Conflict 해소 로직 (LWW·CRDT·분산 합의 중 데이터 성격에 맞는 선택)

- Split-Brain 방지 구성 (Quorum·Witness·Fencing)
- 사이트 간 실시간 동기화를 위한 고대역폭·저지연 전용 네트워크
- 사이트 전환 시에도 유지되는 세션·캐시 무효화 전략
- 모든 사이트에서 동일한 서비스 수준을 유지하기 위한 배포·모니터링 일원화

Active-Passive는 이 목록의 대부분을 지불하지 않습니다. 단일 쓰기 지점을 유지하므로 Write Conflict와 Split-Brain이 원천적으로 발생하지 않고, 단방향 복제라 정합성 관리가 단순하며, 전통적 DR 패턴이라 운영 노하우가 축적되어 있습니다 [S10]. 대신 잃는 것은 앞 항에서 본 짧은 RTO와 상시 검증이므로, 복잡도 절감과 가용성 이득은 정확히 반대 방향의 저울입니다.

2.2 언제 어떤 구성을 선택하는가

같은 조직 안에서도 서비스마다 정답이 다릅니다. 이 절은 서비스 등급을 축으로, 어떤 조건이 Active-Active를 정당화하고 어떤 조건에서 Active-Passive가 더 합리적인지를 가릅니다. 판단의 출발점은 언제나 그 서비스가 감당해야 할 RTO/RPO 목표와 데이터 정합성 요구입니다.

2.2.1 Active-Active가 필요한 서비스 등급

Active-Active가 복잡도 비용을 지불할 가치가 있는 서비스는 세 가지 조건 중 하나 이상을 강하게 요구합니다. 첫째, 무중단이 사실상 필수인 등급입니다. 결제·인증처럼 수초의 중단도 매출 손실과 직결되고 재시도로 메울 수 없는 서비스는 RTO를 수초 단위로 압축해야 하므로, Standby 기동 시간이 개입하는 Active-Passive로는 목표를 맞출 수 없습니다 [S10]. 둘째, 글로벌 사용자를 상대하는 서비스입니다. 여러 리전에 사용자가 흩어져 있으면 GSLB의 GeoIP·RTT(Round Trip Time, 네트워크 왕복 시간) 기반 라우팅으로 각 사용자를 가까운 사이트에 붙여 지연시간을 줄이는 이득이 크고, 이 이득은 모든 사이트가 상시 가동될 때만 실현됩니다. 셋째, DR 능력을 상시 검증해야 하는 등급입니다. 운영 중인 사이트가 곧 복구 경로이므로, 별도 훈련 없이 복구 능력이 지속 확인되는 특성은 규제·감사 대응에서 특히 유효합니다 [S4].

규제 요건도 이 등급 판단에 직접 작용합니다. 금융권은 전자금융감독규정에 따라 RTO 2시간 이내의 재해복구 체계를 의무적으로 갖춰야 하며, 이커머스 기업 역시 99.99% 수준의 SLA(Service Level Agreement, 서비스 수준 협약)를 보장하기 위해 멀티사이트 DR을 구축합니다 [S10]. 규정이 요구하는 RTO 상한이 넉넉하더라도, 실제 서비스 특성상 수초 내 전환과 데이터 무손실이 필요하다면 Active-Active가 그 요구를 충족하는 유일한 구성이 됩니다.

이러한 조건을 서비스 유형에 대입하면 판단이 구체화됩니다. 온라인 결제 게이트웨이, 통합 인증(SSO) 서비스, 글로벌 커머스의 주문 처리, 실시간 트레이딩처럼 중단이 즉시 손실로 이어지고 사용자가 지역적으로 넓게 분포한 서비스는 Active-Active가 정당화되는 대표 등급입니다. 반대로 같은 회사의 사내 배치 작업, 야간 정산, 리포트 생성처럼 수분의 지연이 문제되지 않는 서비스까지 Active-Active로 끌어올릴 이유는 없습니다. 등급을 나누는 기준은 "이 서비스가 수초의 중단과 최소한의 데이터 손실도 허용하지 못하는가"이며, 답이 '그렇다'인 서비스에만 Active-Active의 복잡도 비용을 배정하는 것이 합리적입니다.

2.2.2 Active-Passive가 더 적합한 경우

Active-Active가 언제나 우월한 구성은 아닙니다. 오히려 요구 조건을 넘어서는 과잉 설계는 비용과 장애 표면을 함께 키웁니다. Active-Passive가 더 합리적인 첫 번째 경우는 비용 효율이 우선인 서비스입니다. Standby 사이트를 최소 사양으로 축소 운영하면 이중 인프라를 상시 가동하는 Active-Active보다 총비용을 크게 낮출 수 있고, 수분·수십분의 RTO가 허용되는 서비스라면 이 절감이 그대로 이득으로 남습니다 [S10]. 두 번째는 단일 지역 서비스입니다. 사용자가 한 지역에 몰려 있으면 지리적 분산으로 지연시간을 줄이는 Active-Active의 강점이 발휘될 여지가 적어, 굳이 양방향 복제의 복잡도를 감수할 근거가 약합니다.

세 번째이자 가장 결정적인 경우는 강한 일관성이 요구되는 데이터입니다. 단일 마스터에서만 쓰기를 받아야 정합성이 보장되는 업무는 애초에 양방향 동시 쓰기를 허용할 수 없으므로, 단일 쓰기 지점을 유지하는 Active-Passive가 구조적으로 부합합니다. 이 구성에서는 Write Conflict와 Split-Brain이 발생하지 않아 앞 절에서 열거한 충돌 해소·분단 방지 설계를 통째로 생략할 수 있고, 그만큼 검증할 실패 경로가 줄어 운영 신뢰도가 오히려 높아집니다 [S4].

무리한 Active-Active 도입의 역효과도 함께 봐야 합니다. 조직이 양방향 복제·분산 합의·크로스 리전 세션 공유를 감당할 역량을 갖추지 못한 상태에서 Active-Active를 강행하면, 얻으려던 가용성 대신 오작동하는 충돌 해소 로직과 잘못 구성된 Quorum이 새로운 장애 원인이 됩니다. 이 경우 잘 검증된 Active-Passive가 실제 가용성 면에서 더 나은 결과를 냅니다. 다만 Active-Passive를 택하더라도 Standby의 정상 기동을 정기적으로 확인하고 DNS TTL과 헬스체크 주기를 튜닝해 페일오버 지연을 관리하는 것은 여전히 필수 과제입니다 [S4]. 요컨대 두 구성 사이의 선택은 우열의 문제가 아니라 서비스가 감당해야 할 RTO/RPO와 정합성 요구, 그리고 조직 역량을 맞추는 문제입니다.

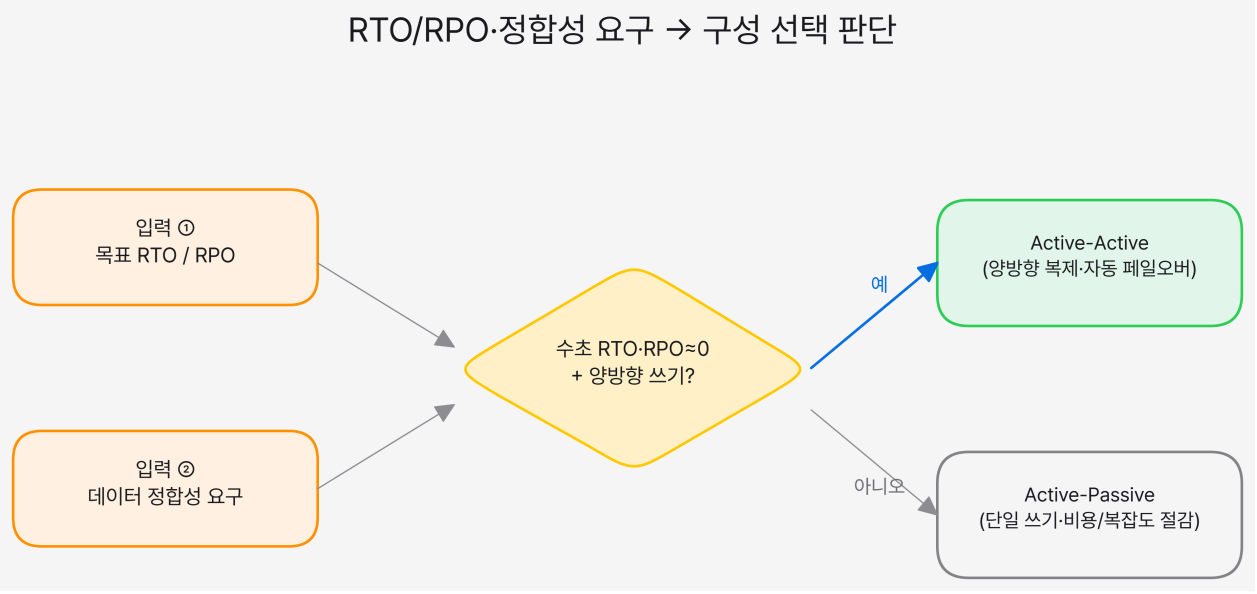

그림. 목표 RTO/RPO와 데이터 정합성 요구를 두 입력으로 놓고 Active-Active와 Active-Passive 중 하나로 수렴하는 판단 흐름.

3장: 글로벌 트래픽 분산 계층 — DNS와 Anycast

Active-Active 구성에서 사용자 요청을 어느 사이트로 보낼지 결정하는 첫 관문이 글로벌 트래픽 분산 계층입니다. 이 계층은 두 가지 기술로 구현됩니다. 하나는 DNS(Domain Name System, 도메인 이름 체계) 응답을 동적으로 제어해 정책 기반으로 사이트를 고르는 방식이고, 다른 하나는 동일한 IP 주소를 여러 지역에서 광고해 네트워크 경로 자체로 최근접 노드에 도달시키는 Anycast 방식입니다. DNS는 지리·부하·가중치 같은 다양한 기준을 반영하는 지능적 분산에 강하지만 캐시와 TTL(Time To Live, 캐시 유효 시간)에 묶여 전환이 느립니다. Anycast는 TTL과 무관하게 네트워크 레벨에서 빠르게 재수렴하지만 정책 표현력이 제한적입니다. 두 기술은 배타적 선택지가 아니라 서로의 약점을 메우는 보완 관계이며, 실무 설계는 대개 둘을 계층적으로 결합합니다[S4][S8]. 본 장은 DNS 기반 GSLB의 질의 해석 흐름과 분산 알고리즘을 구현 레벨에서 정리한 뒤, Anycast가 DNS의 TTL 한계를 어떻게 보완하고 두 기술이 어떻게 역할을 나누는지를 다룹니다.

3.1 DNS 기반 GSLB 동작 원리

DNS 기반 GSLB는 정적인 레코드를 반환하는 일반 DNS와 달리, 질의를 받을 때마다 각 사이트의 상태와 라우팅 정책을 평가해 최적 사이트의 IP를 동적으로 응답합니다. 이 절은 클라이언트 질의가 권한 서버까지 도달하는 흐름과 그 흐름 속에서 캐시가 개입하는 지점을 먼저 짚고, 이어 분산 결정을 실제로 내리는 알고리즘군을 서비스 특성별 선택 기준과 함께 정리합니다.

3.1.1 질의 해석 흐름 (Resolver → Authoritative GSLB)

사용자가 `app.example.com`에 접속하려 할 때 트래픽은 곧바로 사이트로 가지 않습니다. 먼저 이름을 IP로 바꾸는 DNS 해석이 일어나고, GSLB는 바로 이 해석 과정에 개입합니다. 흐름은 다음 순서로 진행됩니다. 클라이언트가 로컬 DNS 리졸버(Local DNS Resolver, 재귀 질의를 대행하는 캐시 서버)에 도메인을 질의하면, 리졸버는 해당 도메인의 권한 DNS(Authoritative DNS)로 지정된 GSLB에 재귀 질의를 보냅니다. GSLB는 이 시점에 세 가지를 평가합니다. 첫째 각 사이트의 헬스체크 상태, 둘째 지리·RTT·가중치 등 분산 알고리즘, 셋째 사이트별 부하와 메트릭입니다. 평가 결과 최적 사이트의 IP를 TTL과 함께 리졸버에 응답하고, 리졸버는 이를 캐시에 저장한 뒤 클라이언트에 전달합니다. 그제야 클라이언트가 그 IP로 직접 HTTP/HTTPS 연결을 맺습니다[S4][S1]. 여기서 핵심은 GSLB가 이 IP 선택 단계에만 관여하고 이후 실제 애플리케이션 트래픽에는 관여하지 않는다는 점입니다. GSLB는 데이터 경로(Data Path) 밖에 위치하는 아웃오브밴드(out-of-band) 방식이며, 이것이 확장성의 근간인 동시에 페일오버 지연이라는 고유한 제약의 원인이기도 합니다.

위치 판단의 정확도는 GSLB가 클라이언트를 어떻게 인식하느냐에 달려 있습니다. 기본적으로 GSLB는 질의를 보낸 리졸버의 IP를 기준으로 위치를 추정하는데, 서울 사용자가 미국에 분산된 퍼블릭 DNS(예: 8.8.8.8)를 쓰면 GSLB는 리졸버 위치인 미국을 기준으로 판단해 엉뚱한 사이트를 반환할 수 있습니다. 이 오판을 줄이는 표준이 EDNS(Extension Mechanisms for DNS, DNS 확장 메커니즘) Client Subnet(RFC 7871)입니다. 재귀 리졸버가 클라이언트 IP의 서브넷 일부(IPv4는 통상 /24, IPv6는 /56)를 마스킹해 질의에 실어 보내면, 권한 서버인 GSLB는 리졸버가 아

닌 실제 클라이언트 서브넷을 기준으로 최적 사이트를 고를 수 있습니다[S8][E1][E2]. 다만 모든 리졸버가 이를 지원하지는 않으며, 프라이버시를 우선하는 일부 리졸버는 EDNS Client Subnet을 기본 비활성화합니다.

캐시가 끼어드는 지점을 이해해야 뒤이은 TTL 튜닝(6장)을 설계할 수 있습니다. 응답 경로에는 권한 DNS 서버, ISP 재귀 리졸버, 퍼블릭 DNS, 운영체제 DNS 캐시, 웹 브라우저 캐시가 층층이 존재합니다. 이 가운데 권한 서버는 관리자가 설정한 TTL을 그대로 적용해 직접 제어할 수 있고, ISP·퍼블릭 리졸버와 OS 캐시는 대체로 TTL을 존중합니다. 문제는 웹 브라우저 캐시로, 상당수 브라우저가 TTL을 무시하고 15분에서 6시간까지 자체 캐싱합니다[S8]. 따라서 GSLB가 장애 사이트의 IP를 응답에서 제거하더라도, 이미 캐시된 구 IP를 쥔 클라이언트는 그 캐시가 만료될 때까지 장애 사이트로 계속 접속을 시도합니다. 이 캐시 사슬이 DNS 기반 페일오버의 실질 지연을 만들며, 순수 DNS 방식의 전환이 최소 수십 초 수준에 머무는 근본 이유입니다.

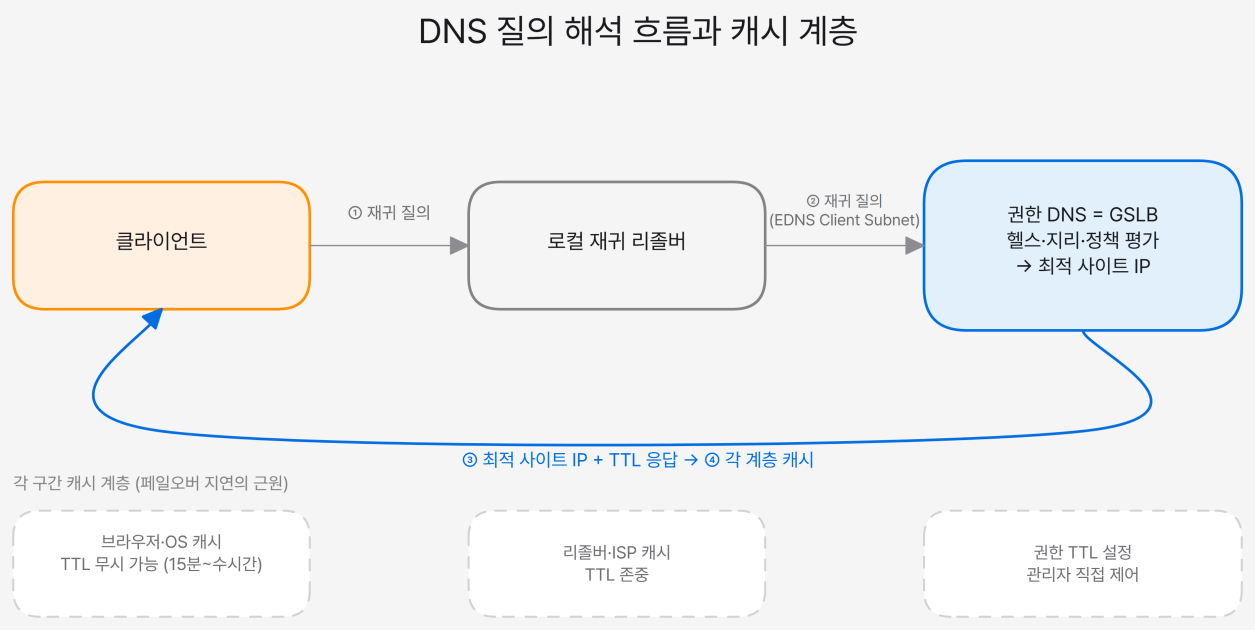

그림. 클라이언트 질의가 로컬 리졸버를 거쳐 권한 GSLB에 도달하고 최적 사이트 IP를 응답받기까지의 흐름과, 각 구간에 개입하는 캐시 계층(권한 서버·재귀 리졸버·OS·브라우저)의 위치.

3.1.2 분산 알고리즘 (Geo·RTT·Weighted 등)

GSLB가 어느 사이트를 고를지는 적용한 분산 알고리즘이 정합니다. 알고리즘은 크게 용량 기준, 위치 기준, 세션 지속 기준으로 나뉘며, 서비스 특성에 맞춰 선택하거나 조합합니다[S1][S4]. **Round Robin**은 각 사이트 IP를 순차 반환하는 가장 단순한 방식으로, 사이트 간 처리 용량이 동일하고 특별한 요구가 없을 때 적합합니다. 구현이 쉽고 트래픽이 고르게 퍼지지만 용량 차이나 클라이언트 위치를 고려하지 못합니다. **Weighted Round Robin**은 여기에 가중치를 더해, 예컨대 사이트 A에 70, 사이트 B에 30을 부여하면 트래픽이 약 7 대 3으로 배분됩니다. 사이트별 용량이 다를 때 쓰며, 카나리 배포나 점진적 마이그레이션처럼 트래픽 비율을 의도적으로 기울일 때도 유용합니다. **Least Connection**은 각 사이트의 현재 활성 연결 수를 받아 가장 적은 곳으로 보냅니다. 파일 다운로드나 스트리밍처럼 세션 길이가 불균일한 서비스에서 효과적입니다.

위치 기준 알고리즘은 지연 최소화가 목적입니다. **Geolocation/Proximity**는 GeoIP 데이터베이스로 클라이언트의 지리적 위치를 파악해 가장 가까운 사이트로 보냅니다. 글로벌 서비스의 응답 시간을 줄이거나, 데이터 주권 규정에 따라 특정 지역 트래픽을 해당 국가 데이터센터로 묶을 때 씁니다. **RTT(Round-Trip Time, 왕복 지연)** 기반 라우팅은 Geolocation보다 한 단계 정밀한 방식으로, 실제 네트워크 왕복 시간을 측정해 응답이 가장 빠른 사이트를 고릅니다. 물리적 거리뿐 아니라 네트워크 혼잡도까지 반영하므로 성능 최적화 정확도가 높지만, 지속적인 RTT 측정 오버헤드가 따릅니다. 세션 지속이 필요한 서비스에는 **Source-IP Hash**를 씁니다. 클라이언트 소스 IP에 해시를 적용해 동일 사용자를 항상 같은 사이트로 고정하는 방식으로, 세션 퍼시스턴스를 DNS 계층에서 확보합니다. 다만 NAT(Network Address Translation) 환경에서는 다수 클라이언트가 하나의 공인 IP를 공유해 부하가 한쪽으로 쏠릴 수 있습니다. 마지막으로 **Priority**(우선순위) 방식은 사이트에 순위를 매겨 최우선 사이트로만 보내다가 그 사이트가 장애가 나면 다음 순위로 넘기는 구성으로, Active-Passive 성격의 페일오버 정책에 대응합니다.

다음 표는 각 알고리즘의 분배 기준과 적합 상황을 정리한 것입니다.

알고리즘	분배 기준	적합 상황
Round Robin	순차 배분	사이트 용량이 동일하고 요구가 단순할 때
Weighted Round Robin	가중치 비율	사이트별 용량 차이, 카나리·점진 마이그레이션
Least Connection	현재 연결 수	세션 길이가 불균일한 서비스(다운로드·스트리밍)
Geolocation/Proximity	지리적 위치	글로벌 지연 최소화, 데이터 주권 준수
RTT	실측 왕복 지연	네트워크 품질 기반 성능 최적화
Source-IP Hash	소스 IP 해시	세션 지속성이 필요한 서비스
Priority	사이트 우선순위	최우선 사이트 우선, 장애 시 순차 페일오버

실무에서는 한 알고리즘만 쓰기보다 계층적으로 조합하는 경우가 많습니다. 예를 들어 1차로 Geolocation으로 리전을 좁힌 뒤 그 안에서 Weighted Round Robin으로 용량을 반영하고, 마지막으로 헬스체크로 장애 사이트를 걸러내는 식입니다. 선택의 축은 결국 서비스가 무엇에 민감한가입니다. 글로벌 지연이 관건이면 Geolocation과 RTT를, 사이트 간 용량 편중이 관건이면 Weighted Round Robin을, 세션 고정이 관건이면 Source-IP Hash를 앞세우는 것이 기본 방향입니다.

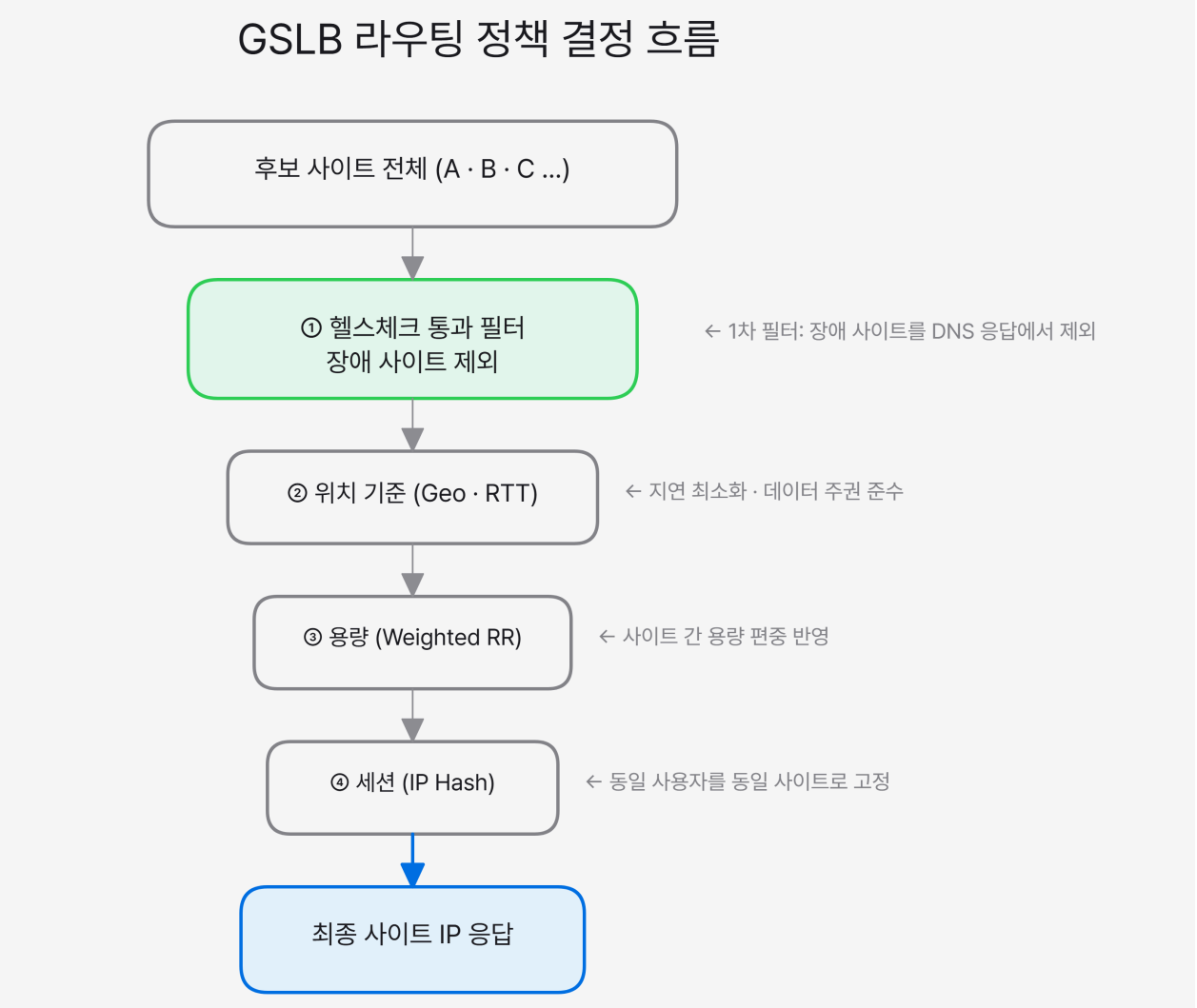

그림. 헬스체크 통과 여부를 1차 필터로 두고, 위치·용량·세션 지속 기준의 분산 알고리즘이 순차 적용되어 최종 사이트 IP가 선택되는 정책 결정 흐름.

3.2 Anycast 병행 전략

DNS 기반 분산은 정책 표현력이 뛰어나지만, 3.1에서 본 캐시 사슬 때문에 전환 속도에는 태생적 한계가 있습니다. Anycast는 이 한계를 네트워크 계층에서 우회하는 보완 기술입니다. 이 절은 BGP(Border Gateway Protocol, 경계 경로 프로토콜) Anycast가 어떻게 TTL과 무관하게 전환하는지, 그리고 그 강점과 한계를 감안해 DNS와 Anycast가 실제로 어떻게 역할을 나누는지를 다룹니다.

3.2.1 BGP Anycast의 네트워크 레벨 전환

BGP Anycast는 네트워크 계층(L3)에서 동작하는 라우팅 기술입니다. 지리적으로 분산된 여러 노드가 동일한 IP 주소를 공유하고, 각자 BGP로 그 IP를 광고합니다. 사용자가 Anycast IP로 요청을 보내면 인터넷 라우터들이 BGP 경로 정보에 따라 네트워크 토폴로지상 가장 가까운 노드로 패킷을 전달합니다. 여기서 "가까움"은 물리적 거리가 아니라 AS-path(자율 시스템 경로) 길이와 라우팅 정책 등 네트워크 홉 기준입니다[S8]. 핵심 이점은 전환 방식에 있습니다. 노드에 장애가 나면 해당 노드가 BGP 광고를 철회하고, 그러면 트래픽은 자동으로 차순위 노드로 재라우팅

됩니다. IP 주소 자체는 바뀌지 않으므로 DNS TTL 캐싱에 의한 지연이 원천적으로 없습니다. DNS 방식이 캐시 만료를 기다려야 하는 데 반해, Anycast는 BGP 경로 재수렴만으로 수초 안에 전환이 일어납니다. 새 노드를 추가할 때도 BGP 광고만 시작하면 자동으로 트래픽 분산에 편입되어 확장이 단순합니다.

그러나 Anycast는 만능이 아니며 분명한 대가와 한계가 있습니다. 첫째, 운영 비용과 복잡도가 높습니다. 글로벌 Anycast 네트워크를 직접 운영하려면 수많은 물리적 PoP(Point of Presence, 접속 거점), BGP 피어링 관계, 상시 트래픽 엔지니어링이 필요합니다[S8]. 둘째, 정책 표현력이 제한적입니다. 라우팅이 BGP 경로 기준으로만 결정되므로 가중치·부하·세션 같은 세밀한 정책을 DNS만큼 유연하게 걸 수 없습니다. 셋째, 세션 유지 관점의 유의점이 있습니다. BGP 경로는 인터넷 상황에 따라 재수렴할 수 있는데, 장시간 유지되는 연결이 경로 변경으로 다른 노드에 재분배되면 세션이 끊길 수 있습니다. 넷째, BGP route leak(경로 유출)이나 경로 이상이 발생하면 트래픽이 예기치 않은 노드로 흘러가는 라우팅 불안정성이 존재합니다. 이런 특성 때문에 Anycast는 상태를 오래 쥐지 않는 DNS·CDN 같은 Stateless 서비스에 특히 잘 맞고, 장시간 Stateful 세션을 다루는 서비스에는 세션 계층 보완이 함께 필요합니다.

다음 표는 DNS 기반 전환과 Anycast 기반 전환을 대비한 것입니다.

비교 항목	DNS 기반 GSLB	BGP Anycast
동작 계층	DNS(L7) 계층	네트워크(L3) 계층
라우팅 기준	지리·부하·RTT·가중치 등 정책	BGP 경로(AS-path, 네트워크 홉)
IP 관리	사이트별 고유 IP, DNS가 선택	모든 노드가 동일 IP 공유
전환 속도	TTL 만료에 의존(수십 초~수분)	BGP 재수렴(수초~수분), TTL 무관
정책 세밀도	높음(다양한 정책 조합)	제한적(경로 기반)
운영 복잡도	상대적으로 낮음	높음(PoP·피어링·트래픽 엔지니어링)

3.2.2 DNS와 Anycast의 역할 분담

두 기술의 특성을 나란히 놓고 보면 상호 보완 구도가 분명해집니다. DNS는 지리·부하·가중치를 반영하는 지능적 정책 기반 분산을 담당하고, Anycast는 TTL에 묶이지 않는 빠른 네트워크 레벨 전환을 담당합니다. 어느 한쪽이 다른 쪽을 대체하는 관계가 아니라, 계층을 나눠 각자의 강점을 발휘시키는 것이 실무의 정석입니다[S8][S6]. 대표적인 조합은 GSLB의 DNS 서버 자체를 Anycast로 배포하는 방식입니다. DNS 질의가 네트워크상 가장 가까운 GSLB 노드에서 처리되므로 DNS 응답 지연이 줄고, GSLB 인프라 자체의 고가용성과 DDoS 방어력도 함께 확보됩니다. 그다음 그 GSLB가 실제 애플리케이션 트래픽은 정책 기반으로 최적 사이트에 배분합니다. 즉 Anycast로 가장 가까운 거점에 도달한 뒤, 그 거점의 DNS 기반 GSLB가 최적 백엔드를 고르는 계층적 구조입니다. 또한 CDN 엣지 라우팅에서는 Anycast로 트래픽을 수신하되 EDNS Client Subnet 정보로 더 정확한 캐시 노드를 선택하는 조합도 널리 쓰입니다.

이 역할 분담은 실제 클라우드 매니지드 제품에서 두 갈래로 갈립니다. AWS Route 53처럼 순수 DNS 정책 라우팅에 무게를 둔 서비스는 정책 표현력이 강한 대신 전환이 TTL에 종속됩니다. 반면 GCP(Google Cloud Platform)의 Cloud DNS와 Cloud Load Balancing 조합은 단일 Anycast IP로 전 세계 트래픽을 받아 네트워크 레벨에서 로드밸런싱하므로, DNS TTL에 의존하지 않는 빠른 장애조치가 강점입니다[S5]. Cloudflare 역시 글로벌 Anycast 네트워크(330여 개 PoP)를 기반으로 빠른 응답과 CDN·DDoS·WAF 통합을 함께 제공합니다. 어느 방식이 유리한지는 서비스가 정책 세밀도를 원하는지 전환 속도를 원하는지에 달려 있으며, 두 요구가 동시에 크다면 앞서 설명한 Anycast 배포 + DNS 정책 라우팅의 계층적 결합이 답이 됩니다. 개별 제품이 실제로 어느 방식을 택하는지와 벤더별 라우팅 정책 수 비교는 8장에서 상세히 다룹니다. 핵심은 트래픽 분산 계층을 DNS냐 Anycast냐의 양자택일로 보지 않고, 지능적 분산과 빠른 전환이라는 두 요구를 계층으로 나눠 함께 충족시키는 설계입니다.

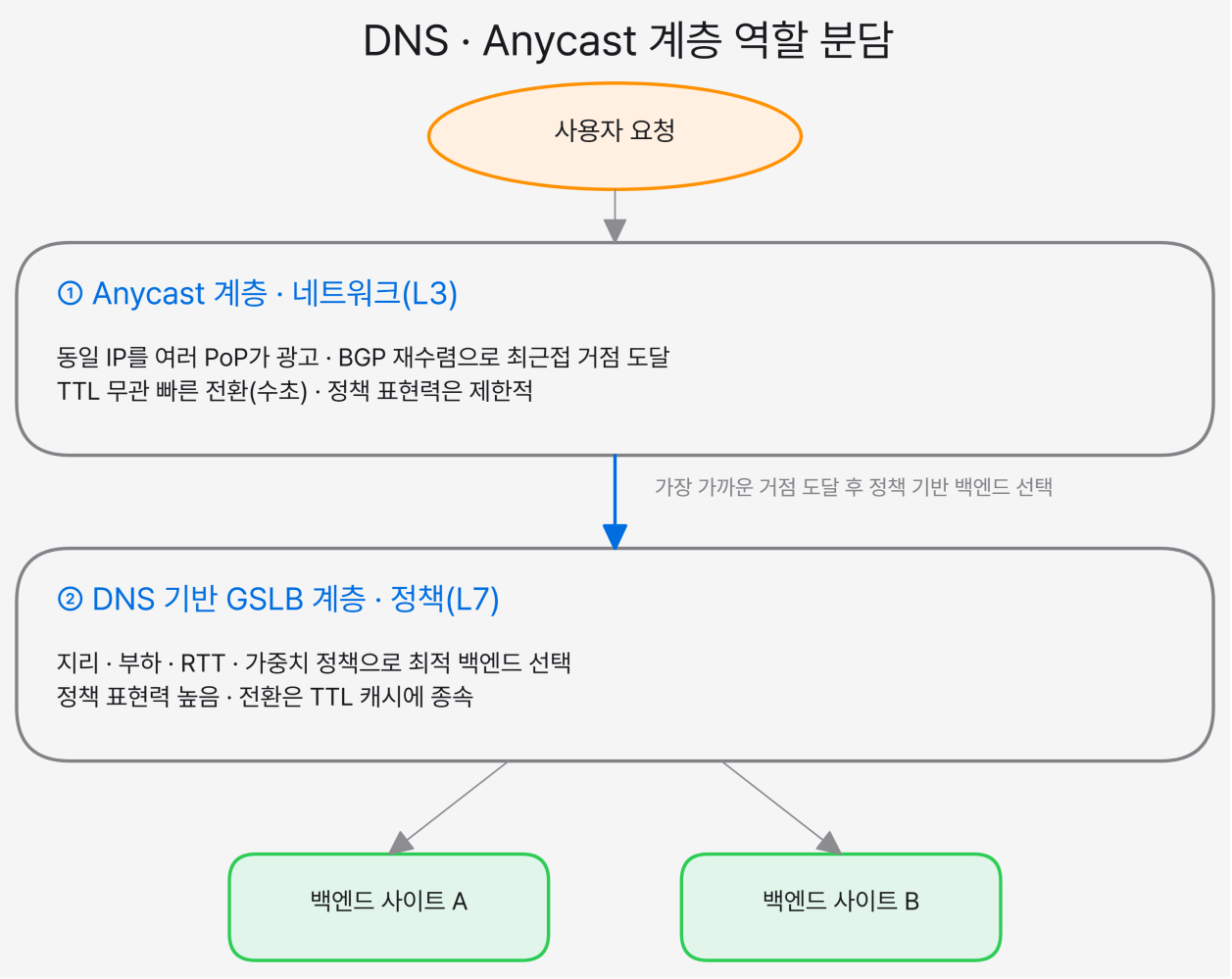

그림. Anycast가 담당하는 네트워크 레벨 거점 도달과 DNS 기반 GSLB가 담당하는 정책 기반 사이트 선택을 상하 계층으로 나눈 역할 분담 개념도.

4장: 헬스체크와 자동 페일오버

GSLB(Global Server Load Balancing)가 트래픽을 정상 사이트로만 흘려보내려면, 먼저 어느 사이트가 정상인지 끊임없이 판정해야 합니다. 이 판정을 담당하는 것이 헬스체크(Health Check)이며, 판정 결과에 따라 장애 사이트를 응답에서 제외하고 정상 사이트로 트래픽을 옮기는 절차가 자동 페일오버(Automatic Failover)입니다. Active-Active 구성에서 두 사이트가 상시 트래픽을 나눠 처리한다는 사실 자체가, 헬스체크가 오판하는 순간 정상 사이트를 죽이거나 죽은 사이트로 트래픽을 계속 보내는 사고로 직결됨을 뜻합니다. 이 장에서 다루는 핵심은 하나로 요약됩니다. 감지 속도를 높이려고 판정을 민감하게 하면 오탐이 늘고, 오탐을 줄이려고 둔감하게 하면 감지가 늦어집니다. 따라서 계층별 점검 조합과 파라미터 수치, 그리고 페일오버·재편입 절차를 서비스 특성에 맞춰 명시적으로 정의하는 것이 페일오버 품질을 좌우합니다 [S4]. 4.1절에서 다층 헬스체크의 계층 구성과 파라미터 튜닝을, 4.2절에서 감지부터 복구 재편입까지의 시간 축 절차를 다룹니다.

4.1 다층 헬스체크 설계

헬스체크는 점검 깊이에 따라 잡아내는 장애의 종류가 달라집니다. 네트워크 도달성만 보는 얕은 점검은 가볍지만 애플리케이션 내부 고장을 놓치고, 비즈니스 로직까지 확인하는 깊은 점검은 실제 가용성을 정확히 반영하지만 구현과 부하가 늘어납니다. 이 절에서는 계층별 점검이 각각 무엇을 검증하고 무엇을 놓치는지, 그리고 감지 민감도를 정하는 파라미터를 어떤 수치로 설정하는지를 구현 관점에서 정리합니다.

4.1.1 L3/L4/L7·애플리케이션 레벨 점검

헬스체크는 OSI 계층에 따라 네 가지 수준으로 나뉩니다. 가장 낮은 수준인 ICMP(Internet Control Message Protocol, 네트워크 제어·오류 통보용 프로토콜) 점검은 대상 서버에 Echo Request(핑)를 보내고 응답이 오는지만 확인합니다. 이는 서버가 네트워크에 살아 있는지(L3 도달성)만 알려 줄 뿐, 그 위에서 도는 서비스가 정상인지는 판단하지 못합니다. 서버는 켜져 있으나 애플리케이션 프로세스가 죽은 상태를 ICMP는 정상으로 오인합니다 [S4]. 다음 수준인 TCP(Transmission Control Protocol, 연결 지향 전송 프로토콜) 점검은 특정 포트로 3-Way Handshake(연결 수립 절차)를 시도해 포트가 열려 있는지(L4 가용성)를 봅니다. 서비스 포트가 리스닝 상태인지는 확인하지만, 그 포트 뒤의 애플리케이션 로직이 정상인지는 여전히 모릅니다. 그 위의 HTTP(HyperText Transfer Protocol, 웹 전송 프로토콜) 또는 암호화 버전인 HTTPS 점검은 GET 요청을 보내 상태 코드(예: 200 OK)와 응답 본문을 검사합니다. 웹 서비스가 응답을 만들어 낸다는 사실(L7 응답)까지 확인하므로 얕은 점검보다 실질적입니다. 다만 웹 계층은 응답하지만 그 뒤의 데이터베이스(DB) 연결이 끊긴 부분 장애는 HTTP 200을 그대로 돌려주어 놓칠 수 있습니다 [S3].

가장 깊은 애플리케이션 레벨 점검은 전용 엔드포인트(예: `/health`)를 두고 DB 연결, 캐시, 외부 API 응답, 디스크 여유 등 의존 요소의 상태를 종합해 반환합니다. GSLB는 이 응답의 상태 코드와 본문 문자열을 함께 검사해 사이트의 실질 가용성을 판정합니다 [S4]. 애플리케이션 레벨 점검이 실제 장애를 잡는 이유가 여기 있습니다. 얕은 점검은 "서버가 살아 있는가"만 묻지만, 깊

은 점검은 "이 사이트가 사용자 요청을 끝까지 처리할 수 있는가"를 묻기 때문입니다. 예컨대 금융 서비스에서는 로그인 페이지 응답 확인에 더해 코어뱅킹 시스템의 잔액 조회 테스트 트랜잭션까지 수행해 실질 가용성을 검증한 사례가 있습니다 [S3]. 실무에서는 이 네 수준을 배타적으로 쓰지 않고 계층적으로 조합합니다. 낮은 계층으로 빠르게 걸러 내고 높은 계층으로 정밀하게 확정하는 방식이 오탐과 미탐을 함께 줄이는 데 유리합니다.

점검 유형	프로토콜/동작	검증 대상	장점	놓치는 장애
ICMP (L3)	Echo Request/Reply	네트워크 도달성	오버헤드 최소, 구현 간단	프로세스 다운 등 애플리케이션 장애
TCP (L4)	포트 3-Way Handshake	서비스 포트 가용성	포트 리스닝 상태 확인	포트 뒤 로직 오류
HTTP/HTTPS (L7)	GET + 상태코드·본문	웹 응답 정상 여부	웹 서비스 동작 확인	DB·캐시 등 의존성 부분 장애
Application-Level	종합 점검 스크립트	비즈니스 로직 수준	실질 서비스 가용성 판단	커스텀 개발 필요, 오버헤드 증가

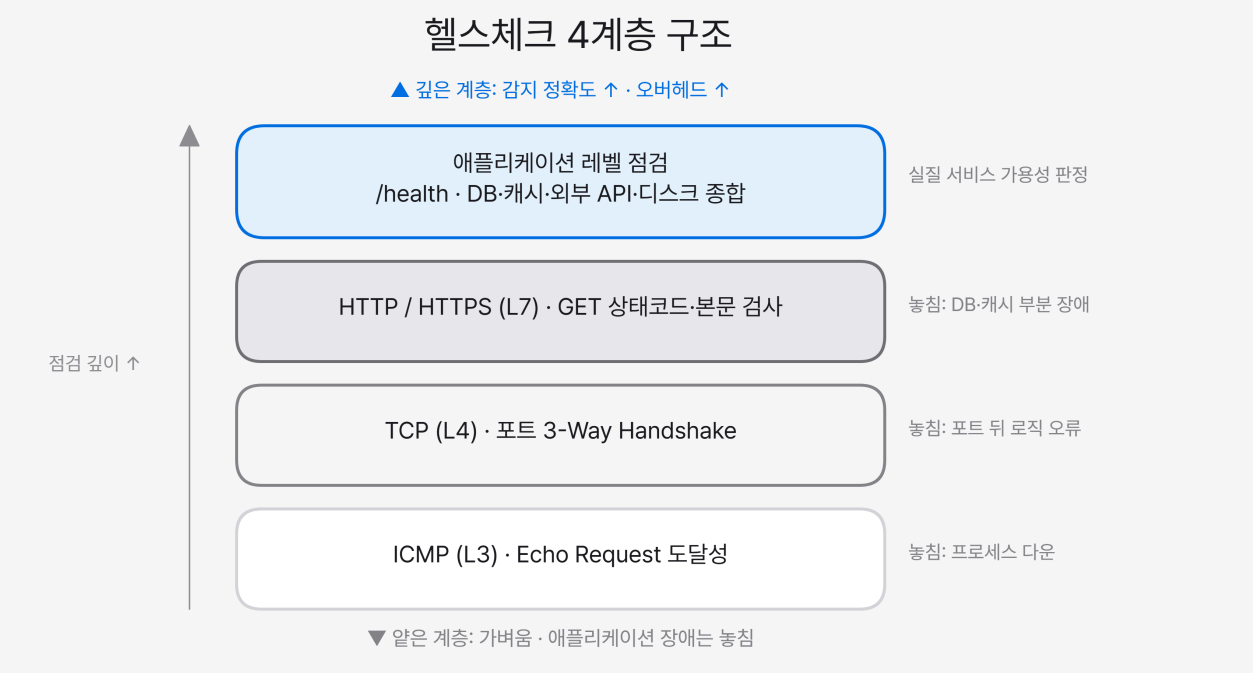

그림. 얕은 계층에서 깊은 계층으로 갈수록 감지 정확도는 높아지고 오버헤드는 증가하는 헬스체크 4계층 구조.

4.1.2 파라미터 튜닝 (Interval·Timeout·Threshold)

계층 조합을 정한 뒤에는 헬스체크의 민감도를 결정하는 네 파라미터를 수치로 확정해야 합니다. Interval(점검 주기)은 헬스체크를 얼마나 자주 수행할지를 정합니다. 짧으면 장애를 빨리 감지하지만 점검 트래픽과 부하가 늘고, 길면 부하는 줄지만 감지가 늦어집니다. Timeout(응답 대기 시간)은 단일 점검에서 응답을 얼마나 기다릴지를 정하며, 네트워크 지연을 감안해 Interval보

다 짧게 잡습니다 [S4]. Threshold(장애 판정 임계, Failure Threshold)는 연속 몇 회 실패해야 사이트를 DOWN으로 마킹할지를 정하는 값입니다. 이 값이 낮으면 일시적 지연에도 과잉 반응해 오탐이 늘고, 높으면 오탐은 줄지만 감지가 지연됩니다. Recovery Threshold(복구 판정 임계)는 반대로 연속 몇 회 성공해야 다시 UP으로 복원할지를 정하며, 플래핑(Flapping, 상태가 오락가락하는 현상) 방지를 위해 장애 판정 임계보다 크게 둡니다 [S4].

권장 실무값은 다음과 같습니다. 애플리케이션 레벨 점검을 10초 간격으로 수행하고 3회 연속 실패로 장애를 판정하면, 장애 발생 시점부터 약 30초 이내에 감지가 완료됩니다(10초 x 3회 = 30초) [S4]. Timeout은 5초 안팎으로 두어 네트워크 왕복 지연을 흡수하되 응답 없는 사이트를 오래 기다리지 않게 합니다. Recovery Threshold는 3~5회로 설정해 장애 판정보다 확실히 크게 둡니다. 이 조합의 설계 의도는 명확합니다. 감지 속도(30초 이내)를 확보하면서도, 단발성 실패 한 번으로 정상 사이트를 죽이지 않도록 3회라는 완충을 두는 것입니다. 다만 이 수치는 서비스 등급에 따라 조정 대상입니다. 즉각 전환이 필수인 미션 크리티컬 서비스는 Interval을 5초로 줄이고, 부하가 민감한 서비스는 Interval을 30초까지 늘립니다 [S4]. Interval을 무작정 줄이면 점검 트래픽이 사이트별로 누적되어 오히려 부하 요인이 되므로, 감지 목표 시간과 점검 부하의 균형점을 서비스별로 정해야 합니다. 헬스체크 임계치를 1~2회로 과도하게 민감하게 잡으면 일시적 부하에도 불필요한 페일오버가 발생한다는 점은 현장 실패 패턴으로 반복 확인된 사항입니다 [S3].

파라미터	의미	권장 실무값	튜닝 방향
Interval	점검 수행 주기	10초 (미션 크리티컬 5초, 부하 민감 30초)	짧으면 빠른 감지·부하 증가
Timeout	단일 점검 응답 대기	5초 (Interval보다 짧게)	네트워크 지연 감안
Failure Threshold	장애 판정 연속 실패 횟수	3회	낮으면 오탐, 높으면 감지 지연
Recovery Threshold	복구 판정 연속 성공 횟수	3~5회	Failure Threshold보다 크게 (플래핑 방지)

이 표의 기본 조합(10초·3회)은 장애 감지 30초를 보장하는 출발점이며, 이후 6장에서 다루는 TTL(Time To Live, DNS 캐시 유지 시간) 대응과 결합해야 사용자 체감 전환 시간이 결정됩니다.

4.2 페일오버 시퀀스

헬스체크가 사이트를 DOWN으로 판정한 뒤, 실제로 그 사이트가 트래픽에서 빠지고 정상 사이트로 전환이 완료되기까지는 여러 단계가 시간 축을 따라 이어집니다. 이 절에서는 감지부터 트래픽 제외까지의 순서와 각 구간에서 지연이 쌓이는 지점을 살펴보고, 복구된 사이트를 다시 편입할 때 플래핑을 막는 방법을 정리합니다.

4.2.1 장애 감지부터 트래픽 제외까지

자동 페일오버는 정상 운영 상태에서 시작해 네 국면을 거칩니다. 정상 운영 국면에서는 두 사이트가 각각 트래픽을 나눠 처리합니다. 장애 감지 국면에서 GSLB가 한 사이트의 헬스체크 실패

를 감지하고, 연속 실패 횟수가 임계(예: 3회)에 도달하면 해당 사이트를 DOWN으로 마킹합니다. 트래픽 전환 국면에서 GSLB는 DNS 응답에서 장애 사이트의 IP를 제거하고, 이후 모든 질의에는 정상 사이트의 IP만 응답합니다. 그러나 여기서 즉시 전환이 끝나지 않습니다. 이미 클라이언트와 중간 리졸버에 캐시된 구 IP는 기존 TTL이 만료될 때까지 유효하므로, 트래픽은 TTL 만료를 따라 점진적으로 전환됩니다. 완전 전환 국면에 이르러서야 장애 사이트의 트래픽이 0이 되고 정상 사이트가 전량을 흡수합니다 [S4].

총 페일오버 시간이 어디서 결정되는지 파악하는 것이 중요합니다. 헬스체크 파라미터만으로 총 시간이 정해지지 않기 때문입니다. 아래 표처럼 감지 시간(Interval x Threshold), GSLB 내부 상태 전파, DNS 전파 지연(TTL 만료 대기), 클라이언트 재연결 시간이 순차로 누적됩니다. 예컨대 감지에 30초, TTL 60초 만료 대기가 더해지면 최선의 조건에서도 수십 초, 캐시 직후 장애라면 1분 이상이 소요됩니다 [S4]. 더 큰 변수는 DNS 캐시 계층입니다. TTL을 아무리 낮춰도 브라우저 자체 캐시(대략 15분~수 시간)나 TTL을 무시하고 자체 최소값을 강제하는 일부 ISP(Internet Service Provider, 인터넷 서비스 제공자) 리졸버 때문에 실제 전환은 더 늦어질 수 있습니다 [S1]. 실제 금융 사례에서 TTL을 60초로 두어도 클라이언트 캐싱을 고려해 실전 전환 시간을 2~5분으로 산정한 것이 이 때문입니다 [S3]. 따라서 이 구간의 지연을 줄이려면 6장에서 다루는 낮은 TTL·Migration TTL 패턴과 다중 A레코드·클라이언트 재시도를 함께 설계해야 합니다.

구성 요소	소요 시간	지연 발생 지점
장애 감지	Interval x Threshold (예: 10초 x 3회 = 30초)	헬스체크 파라미터
GSLB 내부 전파	사이트 간 상태 동기화 (수초)	메트릭 교환 주기
DNS 전파 지연	기존 TTL 만료 대기 (TTL 60초면 최대 60초)	리졸버·클라이언트 캐시
클라이언트 재연결	애플리케이션 의존	기존 연결 종료·재질의

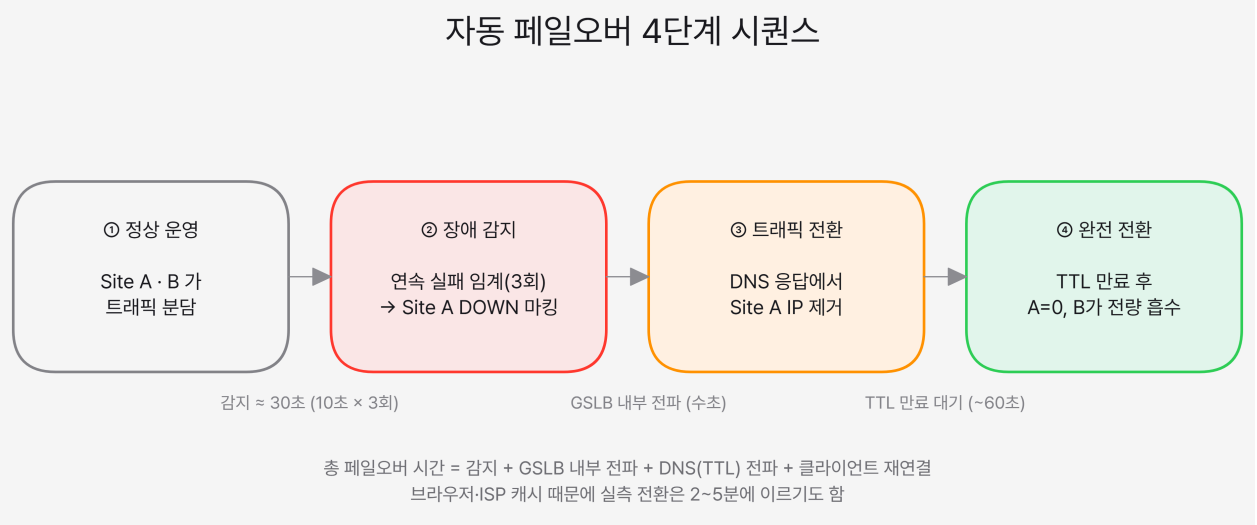

그림. 정상 운영 → 장애 감지 → DNS 응답에서 IP 제거 → TTL 만료 후 점진 전환에 이르는 자동 페일오버 4단계 시퀀스.

4.2.2 복구 시 재편입과 플래핑 방지

장애 사이트가 복구되면 다시 트래픽에 편입해야 하지만, 이 재편입은 감지보다 더 신중해야 합니다. 복구 국면의 절차는 다음과 같습니다. 첫째, 헬스체크가 재성공하기 시작하면 GSLB는 곧바로 사이트를 UP으로 되돌리지 않고 Recovery Threshold(예: 3~5회 연속 성공)를 충족하는지 지켜봅니다. 둘째, 임계를 충족하면 사이트를 UP으로 복원하고 DNS 응답에 해당 IP를 다시 포함합니다. 셋째, 트래픽이 TTL을 따라 점진적으로 재분배됩니다 [S4]. 성급한 재편입이 위험한 이유는 복구 직후 사이트가 아직 안정화되지 않았을 수 있기 때문입니다. 캐시가 비어 있거나 연결 풀이 준비되지 않은 사이트에 전량 트래픽이 몰리면 다시 과부하로 쓰러지고, 이 과정이 반복되면 상태가 UP과 DOWN을 오가는 플래핑에 빠집니다. 플래핑은 페일오버를 반복 유발해 오히려 2차 장애를 부릅니다.

플래핑을 막는 핵심은 복구 판정을 장애 판정보다 엄격하게 두는 비대칭 설계입니다. 장애는 3회 실패로 빠르게 감지해 피해를 줄이되, 복구는 3~5회 연속 성공을 요구해 "확실히 안정된 뒤에만" 편입합니다 [S4]. 감지는 민감하게, 복구는 둔감하게 두는 이 비대칭이 감지 속도와 안정성을 동시에 확보하는 방법입니다. 여기에 안정화 대기(복구 후 일정 시간 트래픽을 지연 편입)와 가중치 점진 상향을 더하면, 복구 사이트가 처음부터 전량을 받지 않고 낮은 비율부터 점차 흡수하게 되어 재부하 위험을 줄일 수 있습니다. 헬스체크 임계를 지나치게 민감하게 잡아 정상 서버를 반복적으로 DOWN 판정하는 것은 현장에서 반복되는 실패 패턴이므로, 복구 임계와 안정화 대기는 오탐 억제 장치로도 기능합니다 [S3]. 재편입 절차를 정리하면 다음과 같습니다.

- 헬스체크 재성공 시작 후 Recovery Threshold(3~5회 연속 성공) 충족 여부를 확인합니다.
- 임계 충족 전까지는 사이트를 DNS 응답에 포함하지 않습니다.
- 복구 확정 후 안정화 대기 시간을 두어 캐시·연결 풀이 준비될 여유를 확보합니다.
- 가중치를 낮은 값에서 시작해 점진적으로 상향하며 트래픽을 서서히 흡수시킵니다.
- 복구 임계는 항상 장애 임계보다 크게 유지해 플래핑을 억제합니다.

이렇게 감지·전환·재편입의 각 임계와 대기를 수치로 못박아 두면, 11장에서 종합하는 실무 권장값 체크리스트와 6장의 TTL·클라이언트 재시도 설계가 서로 맞물려 사용자 체감 페일오버 품질을 일관되게 관리할 수 있습니다.

5장: 데이터 복제와 정합성 — Active-Active의 핵심 난제

GSLB는 트래픽을 여러 사이트로 나눌 뿐, 각 사이트가 다루는 데이터를 대신 맞춰 주지 않습니다. 3장과 4장에서 다룬 DNS 분산과 자동 페일오버가 완벽하게 동작해도, 두 사이트가 같은 레코드를 동시에 갱신하는 순간 데이터 계층에서 정합성 문제가 발생합니다. Active-Passive 구성이 단일 쓰기 지점 덕분에 일관성을 비교적 쉽게 지키는 반면, 모든 사이트가 동시에 쓰기를 받는 Active-Active는 양방향 복제와 충돌 해소를 필수로 떠안습니다 [S4]. 이 장은 복제 방식(동기·비동기·반동기)의 트레이드오프, 쓰기 충돌 해소(LWW·CRDT·분산 합의), 그리고 네트워크 분단 시의 최후 방어선인 Split-Brain 방지를 다룹니다. 세 주제를 관통하는 물리 법칙은 하나입니다. 사이트 사이의 거리가 곧 지연이며, 이 지연이 일관성과 성능을 동시에 만족시키지 못하게 만듭니다 [S4].

5.1 복제 방식 선택

5.1.1 동기·비동기·반동기 복제

Active-Active DR에서 가장 큰 도전 과제는 다중 사이트 간 데이터 일관성 유지입니다 [S4]. 데이터베이스 복제 방식은 크게 세 가지로 나뉘며, 각각 일관성과 성능, 손실 위험에서 서로 다른 지점에 위치합니다.

동기 복제(synchronous replication)는 한 사이트에 쓰기 요청이 들어오면 원격 사이트가 그 쓰기를 반영해 확인 응답을 보낼 때까지 원본 쓰기를 완료로 처리하지 않습니다. 모든 확인이 끝난 뒤에야 클라이언트에게 성공을 반환하므로 강한 일관성(strong consistency)을 보장하고, 어느 사이트로 읽든 같은 값을 돌려줍니다. 대가는 쓰기 지연입니다. 확인 응답이 원격 사이트를 한 번 왕복해야 하므로 쓰기 성능이 사이트 간 왕복 지연(RTT, Round Trip Time, 왕복 지연)에 직접 묶입니다. 손실은 없어 RPO(Recovery Point Objective, 목표 복구 시점)가 0에 수렴하며, 금융·결제처럼 데이터 무결성이 필수인 서비스에 적합합니다 [S4].

비동기 복제(asynchronous replication)는 원본 사이트가 로컬 쓰기를 마치는 즉시 클라이언트에 성공을 반환하고, 원격 사이트로의 전파는 뒤이어 별도로 진행합니다. 쓰기 성능이 높고 원거리 사이트의 지연에 영향을 받지 않지만, 전파가 끝나기 전 원본 사이트가 무너지면 아직 복제되지 않은 분량이 사라집니다. 즉 복제 지연분만큼 데이터 손실이 발생할 수 있으며, 일관성은 최종 일관성(eventual consistency)에 머뭅니다. 일반 웹 서비스나 로그성 데이터처럼 짧은 손실을 감내할 수 있는 경우에 알맞습니다 [S4].

반동기 복제(semi-synchronous replication)는 둘 사이의 절충입니다. 원격 사이트가 쓰기를 완전히 반영할 때까지 기다리지는 않되, 최소한 원격 노드가 로그를 수신했다는 확인은 받은 뒤 쓰기를 완료 처리합니다. 완전 동기만큼의 지연 없이 손실 위험을 최소화하므로, 성능과 안전성의 균형이 필요할 때 선택합니다 [S4]. 아래 표는 세 방식을 일관성·성능·손실 축으로 정리한 것입니다.

복제 방식	일관성	성능(쓰기)	데이터 손실	적합 시나리오
동기 복제	강한 일관성	쓰기 지연 증가(네트워크 RTT 의존)	없음(RPO=0)	금융·결제 등 데이터 무결성 필수
비동기 복제	최종 일관성	높은 쓰기 성능	복제 지연분 손실 가능	일반 웹 서비스, 로그 데이터
반동기 복제	절충안	중간	최소화	성능과 안전성 균형 필요 시

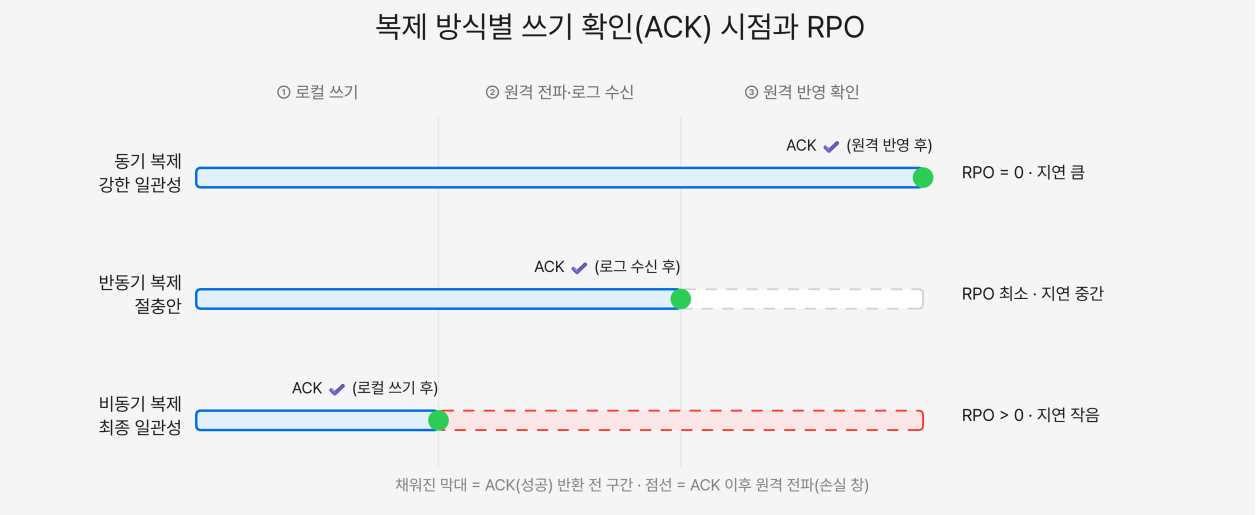

그림. 동기·비동기·반동기 복제의 쓰기 확인 흐름과 RPO·지연 위치 비교.

방식 선택의 핵심은 서비스가 감내할 수 있는 손실 크기입니다. 손실을 전혀 허용하지 않는다면 동기 복제로 가되 뒤이어 다룰 지연 문제를 감수해야 하고, 성능이 우선이면 비동기로 가되 장애 시 손실을 설계 전제로 받아들여야 합니다.

5.1.2 RPO 목표와 쓰기 지연의 균형

복제 방식 결정은 결국 목표 RPO와 쓰기 성능 사이의 균형점을 찾는 작업입니다. 이 균형을 좌우하는 것이 지리적 거리에서 오는 물리적 지연입니다. 동기 복제의 쓰기 지연은 네트워크 RTT에 직접 의존하므로 [S4], 사이트가 멀수록 매 쓰기가 더 오래 걸립니다. 빛이 광섬유를 지나는 속도는 유한하고 라우팅·큐잉 지연이 여기에 더해지므로, 수백 킬로미터 떨어진 리전 사이에서는 왕복 지연이 수 밀리초에서 수십 밀리초에 이릅니다. 동기 복제에서는 이 왕복이 모든 쓰기 트랜잭션마다 반복되어 누적됩니다. 거리는 지연이라는 물리 한계는 설계로 없앨 수 없고, 관리할 수 있을 뿐입니다.

목표 RPO가 0이면 사실상 동기 복제가 강제되며, 그만큼의 쓰기 지연을 서비스가 받아들일 수 있어야 합니다. 이 지연이 처리량 요구를 넘어선다면 선택지는 몇 가지로 좁혀집니다. 첫째, 사이트 간 거리를 줄여 RTT 자체를 낮춥니다. 다만 두 사이트가 너무 가까우면 광역 재해를 함께 겪을 위험이 커지므로, DR 관점의 지리적 분리와 지연 사이에서 다시 절충해야 합니다. 둘째, 동기 복제 대상을 정말로 손실이 허용되지 않는 핵심 데이터로 한정하고, 나머지는 비동기로 돌려

전체 쓰기 지연을 낮춥니다. 셋째, 반동기 복제로 옮겨 원격 반영까지 기다리지 않고 로그 수신 확인만으로 완료 처리해, 손실을 최소화하면서 지연을 줄입니다 [S4].

RPO 목표별 권고는 다음과 같이 정리할 수 있습니다.

목표 RPO	권고 복제 방식	설계 유의점
0(무손실)	동기 복제	쓰기 지연 = RTT 누적, 사이트 간 거리·처리량 사전 검증
초 단위	반동기 복제	로그 수신 확인 기반, 지연과 손실의 절충
분 단위 이상	비동기 복제	최종 일관성 전제, 장애 시 복제 지연분 손실 수용

여기서 유의할 점은 RPO 목표를 데이터 도메인 단위로 나눌 수 있다는 것입니다. 계정 잔액과 결제 원장처럼 손실이 곧 사고인 데이터는 동기로, 사용자 활동 로그나 추천 캐시처럼 재생성이 가능한 데이터는 비동기로 두는 혼합 전략이 현실적입니다. 단일 RPO를 서비스 전체에 일괄 적용하면, 손실 민감도가 낮은 데이터까지 동기 복제의 지연 비용을 치르게 되어 전체 성능이 불필요하게 희생됩니다. 복제 방식 선택은 서비스 전체가 아니라 데이터 도메인마다 내리는 결정이라는 점이 실무의 출발점입니다.

5.2 쓰기 충돌과 Split-Brain 방지

5.2.1 쓰기 충돌 해소 (LWW·CRDT·분산 합의)

Active-Active에서 두 사이트가 같은 레코드를 거의 동시에 갱신하면 쓰기 충돌(write conflict)이 발생합니다. 양방향 쓰기를 받는 구성이라면 충돌 해소 로직이 선택이 아니라 필수입니다 [S4]. S4는 네 가지 해소 전략을 제시하며, 각각 구현 복잡도와 데이터 안전성에서 뚜렷한 차이를 보입니다.

LWW(Last-Write-Wins, 마지막 쓰기 우선)는 타임스탬프를 기준으로 가장 나중의 쓰기를 채택합니다. 구현이 단순하고 판정이 빠르지만, 늦게 도착한 쪽이 무조건 이기므로 먼저 쓴 유효한 갱신이 조용히 사라질 수 있습니다 [S4]. 예를 들어 두 사이트에서 같은 사용자 프로필을 각기 다른 필드로 수정했는데 타임스탬프만으로 한쪽 레코드 전체를 채택하면, 다른 쪽이 수정한 필드까지 함께 유실됩니다. 여기에 사이트 간 시계 오차가 겹치면 실제로는 먼저 일어난 쓰기가 더 늦은 것으로 판정되는 역전도 생깁니다. LWW는 충돌이 드물고 손실을 감내할 수 있는 데이터에만 안전합니다.

애플리케이션 레벨 충돌 해소는 애플리케이션이 충돌을 감지하고 비즈니스 로직에 따라 병합합니다 [S4]. 필드 단위로 병합하거나 도메인 규칙(예: 재고는 더 작은 값 채택, 장바구니는 합집합)으로 판정하므로 LWW의 무단 유실을 피할 수 있으나, 충돌 유형마다 병합 규칙을 구현해야 해 로직이 늘어납니다.

CRDT(Conflict-free Replicated Data Type, 충돌 없는 복제 자료형)는 수학적으로 충돌이 발생하지 않도록 설계된 자료 구조를 사용합니다 [S4]. 카운터·집합·순서 같은 특정 자료형에 한해, 어

떤 순서로 갱신이 도착해도 모든 사이트가 같은 최종 상태로 수렴함을 보장합니다. 병합 규칙을 자료형이 내장하므로 애플리케이션이 충돌을 직접 판정할 필요가 없다는 점이 강점입니다. 다만 표현할 수 있는 자료형이 제한되어 모든 도메인을 CRDT로 모델링하기는 어렵습니다.

분산 합의 알고리즘(Paxos·Raft)은 리더 선출과 과반 합의를 통해 일관성을 보장합니다 [S4]. 복잡하지만 강력하며, 순서와 단일성이 중요한 데이터에 적합합니다. 아래 표는 데이터 성격에 따른 선택 기준을 요약합니다.

충돌 해소 방식	특성	데이터 유형 적합성	유의점
LWW	타임스탬프 기반 단순 채택	충돌 드문 단일 값	무단 유실·시계 오차 역전 위험
애플리케이션 레벨 병합	비즈니스 로직으로 병합	필드별 병합 가능한 레코드	충돌 유형별 규칙 구현 부담
CRDT	수학적 수렴 보장	카운터·집합·순서	표현 가능한 자료형 제한
분산 합의(Paxos·Raft)	과반 합의·리더 선출	순서·단일성 필수 데이터	구현·운영 복잡도 높음

선택의 기준은 데이터의 성격입니다. 증가·감소만 하는 카운터는 CRDT가, 유실이 곧 사고인 원장은 분산 합의가, 충돌이 드문 설정 값은 LWW가 각각 합리적입니다.

5.2.2 Split-Brain 방지 (Quorum·Witness·Fencing)

쓰기 충돌 해소가 정상 통신 상황의 문제라면, Split-Brain은 사이트 사이 네트워크가 끊긴 분단(network partition) 상황의 문제입니다. 네트워크 파티션이 발생했을 때 양쪽 사이트가 서로를 장애로 오인하고 각자 독립적으로 쓰기를 계속 수락하면, 두 갈래의 데이터가 서로 모르게 갈라져 나중에 병합이 불가능한 불일치가 남습니다 [S4]. Active-Active가 안고 있는 대표적 위험이 바로 이 Split-Brain이며 [S4], 통신이 회복된 뒤 어느 쪽이 옳은지 판정할 수 없는 상태가 가장 치명적입니다. 앞 절의 충돌 해소는 두 쓰기가 서로에게 전파된다는 전제 위에서 동작하지만, 분단 중에는 전파 자체가 끊기므로 별도의 방어선이 필요합니다.

첫째 방어선은 Quorum 기반 의사결정입니다. 과반수 노드가 합의할 때만 쓰기를 허용하므로, 분단으로 노드가 둘로 갈리면 과반을 확보한 쪽만 쓰기를 계속하고 소수파는 쓰기를 멈춥니다 [S4]. 과반이라는 조건 덕분에 양쪽이 동시에 주도권을 갖는 상황이 원천적으로 배제됩니다. 여기서 노드 수를 홀수로 두는 이유가 나옵니다. 짝수 노드는 정확히 반씩 갈리면 어느 쪽도 과반이 되지 못해 전체가 멈추거나, 반대로 양쪽 모두 과반을 주장하는 교착에 빠질 수 있습니다. 홀수 구성은 어떤 분단에서도 한쪽만 과반이 되도록 보장합니다.

둘째, Witness/Arbiter 노드입니다. 두 사이트만으로는 분단 시 1대 1로 갈려 과반이 성립하지 않으므로, 제3의 위치에 데이터를 갖지 않는 가벼운 중재자 노드를 배치합니다 [S4]. 이 Witness가 어느 쪽과 통신되는지에 따라 과반이 결정되어, 사이트가 둘뿐인 구성에서도 홀수 투표 정족수를 만들 수 있습니다. Witness는 실제 데이터 부담이 없어 저사양으로 운영할 수 있고, 두 주 사

이트와 장애 영역이 겹치지 않는 제3 지역에 두는 것이 중요합니다. Witness가 한쪽 사이트와 같은 장애를 겪으면 중재자 역할을 하지 못합니다.

셋째, Fencing 메커니즘입니다. Quorum에서 밀려난 소수파 노드가 착오로라도 쓰기를 이어 가지 못하도록, 소수파의 쓰기를 강제로 차단합니다 [S4]. 과반을 잃은 노드는 스스로 쓰기를 중단해야 하지만, 소프트웨어 오류나 지연으로 그러지 못할 수 있으므로 상위 계층이 해당 노드를 격리해 데이터 경로에서 떼어 냅니다. Quorum이 "누가 쓸 자격이 있는가"를 정한다면, Fencing은 "자격 없는 쪽이 실제로 쓰지 못하게" 만드는 마지막 장치입니다.

세 기법은 대체재가 아니라 계층입니다. Quorum으로 정족수를 정의하고, Witness로 짝수·2사이트 한계를 보완하며, Fencing으로 소수파의 이탈 쓰기를 봉쇄합니다. 이 방어선이 없으면 앞서 설계한 복제와 충돌 해소가 아무리 정교해도 분단 한 번에 데이터가 갈라집니다. Active-Active의 성패가 데이터 계층 설계에서 갈린다는 이 장의 결론은, 결국 정상 상황의 복제·충돌 해소와 비정상 상황의 Split-Brain 방지를 서비스 특성에 맞춰 사전에 함께 결정하라는 요구로 정리됩니다 [S4].

6장: 세션·상태 관리와 TTL 대응

GSLB는 트래픽을 어느 사이트로 보낼지 결정할 뿐, 그 사이트로 넘어간 사용자가 이전과 같은 상태로 서비스를 이어 가는지는 보장하지 않습니다. Active-Active 구성에서 사용자 체감 전환 속도를 실제로 정하는 것은 두 가지입니다. 하나는 상태(세션)를 어디에 두느냐이고, 다른 하나는 DNS(Domain Name System, 도메인 이름 체계) 응답의 TTL(Time To Live, 캐시 유효 시간)을 어떻게 다루느냐입니다. 앞의 4장이 장애 감지 속도를, 5장이 데이터 정합성을 다뤘다면, 이 장은 그 사이에서 사용자 경험을 좌우하는 세션 계층과 이름 해석 계층을 다룹니다. 핵심 메시지는 하나로 모입니다. DNS는 페일오버를 보조해야 하며, 페일오버 자체가 되어서는 안 됩니다[S4]. 상태 설계와 TTL 설계가 어긋나면 GSLB가 아무리 빠르게 장애 사이트를 응답에서 제외해도 사용자는 재로그인을 강요당하거나 캐시된 옛 IP로 실패한 연결을 반복합니다.

6.1 세션·상태 전략

6.1.1 Sticky·외부 저장소·Stateless 비교

세션을 어디에 두느냐에 따라 사이트 전환 시 사용자가 겪는 경험이 갈립니다. 대표적인 네 가지 방식의 특성은 아래와 같습니다[S4].

방식	저장 위치	장점	단점	사이트 전환 시 세션
Sticky Session	GSLB/LB가 사용자를 동일 사이트에 고정	구현 간단, 세션 공유 불필요	장애 시 세션 유실, 부하 불균형	유실됨(재로그인)
External Session Store	Redis·Memcached 등 외부 저장소에 중앙 집중	사이트 무관 접근, 페일오버에 유리	외부 저장소가 SPOF 가능	유지(저장소 가용 시)
Session Replication	사이트 간 세션을 실시간 복제	어느 사이트에서든 세션 유효	네트워크 트래픽·동기화 지연 증가	유지(복제 완료분)
Stateless(JWT)	클라이언트가 토큰에 정보 보관	서버 간 동기화 불필요, 확장성 최고	토큰 크기 제한, 즉시 무효화 어려움	유지(서버 상태 무관)

Sticky Session은 GSLB나 로드밸런서가 동일 사용자를 항상 같은 사이트로 보내는 방식입니다. 구현이 단순하고 사이트 간 세션 공유가 필요 없다는 장점이 있으나, 고정된 사이트가 장애로 빠지는 순간 그 사이트에만 있던 세션이 통째로 사라집니다[S4]. 결국 페일오버가 발생한 사용자는 재로그인을 겪게 되어 무중단이라는 Active-Active의 취지가 사용자 관점에서 무너집니다. External Session Store는 Redis 같은 외부 저장소에 세션을 중앙 집중해 어느 사이트에서도 같은 세션을 읽습니다. 페일오버에 유리한 대신 그 저장소 자체가 SPOF(Single Point of Failure, 단일 장애점)가 될 수 있어, 저장소를 다시 다중화해야 하는 재귀적 과제를 낳습니다. Session Replication은 사이트 간 세션을 실시간 복제하지만 네트워크 트래픽과 동기화 지연이라는 비용

을 수반합니다. Stateless 방식은 세션 정보를 서버가 아니라 클라이언트가 지닌 JWT(JSON Web Token, JSON 기반 인증 토큰) 등에 담아, 어느 사이트가 요청을 받아도 토큰만 검증하면 되므로 서버 간 동기화가 아예 불필요합니다. 대신 토큰 크기 제한과 발급된 토큰을 만료 전에 즉시 무효화하기 어렵다는 한계가 있습니다[S4]. 이 비교의 결론은 분명합니다. Active-Active 환경에서는 Stateless 아키텍처(JWT) 또는 크로스 리전 복제를 갖춘 External Session Store(Redis Cluster)가 가장 안정적이며, Sticky Session은 페일오버 시 세션 유실이 불가피하므로 미션 크리티컬 서비스에서는 권장되지 않습니다[S4].

6.1.2 크로스 리전 세션 공유

세션도 결국 데이터의 한 종류입니다. 따라서 리전 간에 세션을 공유하려는 순간, 5장에서 다룬 복제의 트레이드오프가 그대로 세션 계층에 옮겨 붙습니다. External Session Store를 여러 리전에 걸쳐 운영하려면 리전 간 세션 저장소를 복제해야 하는데, 여기서 동기 복제와 비동기 복제의 선택 문제가 다시 등장합니다. 동기 복제로 모든 리전에 세션 쓰기를 확정하면 사용자는 어느 리전으로 전환돼도 즉시 같은 세션을 보지만, 리전 간 왕복 지연(RTT, Round Trip Time, 왕복 소요 시간)이 매 로그인·매 세션 갱신마다 쓰기 지연으로 더해집니다. 비동기 복제로 지연을 줄이면 이번에는 복제가 도착하기 전 리전으로 전환된 사용자가 세션을 찾지 못하는 창이 생깁니다. 즉 세션 저장소의 복제 지연은 곧 페일오버 직후의 재로그인 확률로 나타납니다.

이 지점에서 Stateless 토큰이 근본적 완화책이 되는 이유가 드러납니다. JWT처럼 상태를 클라이언트가 들고 다니면 리전 간에 복제할 서버 측 세션 자체가 존재하지 않으므로, 복제 지연도 SPOF도 원천적으로 사라집니다[S4]. 요청을 받은 리전은 토큰 서명만 검증하면 되고, 이 검증에 필요한 키는 정적이어서 세션 데이터처럼 실시간 복제할 필요가 없습니다. 다만 Stateless가 만능은 아닙니다. 토큰을 즉시 무효화하기 어렵다는 한계 때문에 로그아웃이나 권한 회수를 실시간으로 반영해야 하는 서비스에서는 별도의 무효화 목록(블록리스트)을 두게 되는데, 이 목록이 다시 리전 간 공유 대상이 되어 앞서의 복제 문제를 축소된 형태로 되살립니다. 그럼에도 무효화 목록은 전체 세션에 비해 크기가 훨씬 작아 복제 부담이 낮습니다. 실무에서는 인증 상태의 대부분을 Stateless 토큰으로 처리하고, 즉시 무효화가 필요한 최소한의 항목만 크로스 리전 저장소로 공유하는 절충이 자리 잡습니다. 세션도 데이터라는 관점을 유지하면, 세션 계층의 설계는 5장의 복제 방식 선택과 일관되게 다뤄야 한다는 원칙이 분명해집니다.

6.2 TTL·캐시로 인한 지연 완화

6.2.1 낮은 TTL과 Migration TTL 패턴

GSLB가 장애 사이트의 IP를 응답에서 즉시 빼도, 클라이언트 측 DNS 캐시에 이전 IP가 남아 있으면 전환이 그만큼 지연됩니다[S4]. 이 지연을 결정하는 값이 TTL입니다. TTL은 DNS 응답이 캐시에 유지되는 시간을 초 단위로 지정하며, 장애가 나도 기존 TTL이 만료되기 전까지 클라이언트는 캐시된 옛 IP로 계속 접속합니다[S8]. 따라서 TTL을 낮추면 캐시 갱신 주기가 짧아져 페일오버 수렴이 빨라집니다. GSLB 환경의 일반 권장값은 60초입니다. TTL을 60초로 두면 최악의 경우(캐시 직후 장애)에도 최대 1분, 평균적으로는 약 30초 안에 DNS가 갱신됩니다[S4][S8].

TTL을 무작정 낮추는 데에는 대가가 따릅니다. TTL이 낮을수록 캐시가 자주 만료되어 DNS 질의가 그만큼 늘고, 권한 DNS 서버 부하가 비례해 증가합니다[S8]. 게다가 일부 ISP 리졸버는 300

초 미만의 TTL을 무시하고 자체 최소값을 강제하며, 브라우저는 TTL을 무시하고 15분에서 최대 몇 시간까지 자체 캐싱하기도 합니다[S8]. 즉 TTL을 0으로 두어도 즉각적인 페일오버는 보장되지 않습니다. 이 때문에 상황별로 TTL을 조정하는 Migration TTL 패턴이 쓰입니다. 평상시에는 TTL을 300초에서 3600초로 길게 유지해 DNS 부하를 낮추고, 계획된 사이트 전환이 예정되면 변경 48시간 전에 TTL을 60초로 미리 하향한 뒤, 기존의 긴 TTL이 완전히 만료된 다음에 실제 변경을 수행하고, 안정화 후 TTL을 원래 값으로 복원합니다[S4]. 이 패턴은 평상시 부하와 전환 시 민첩성이라는 두 요구를 시간축에서 분리해 양립시킵니다. 계획된 변경에는 매우 효과적이지만, 예고 없는 돌발 장애에는 사전 하향 시간이 없어 평상시 TTL이 그대로 수렴 병목이 된다는 한계가 있습니다. 그래서 상시 낮은 TTL과 Migration TTL 패턴 중 무엇을 택할지는 서비스 등급과 DNS 인프라의 질의 처리 용량을 함께 보고 정해야 합니다.

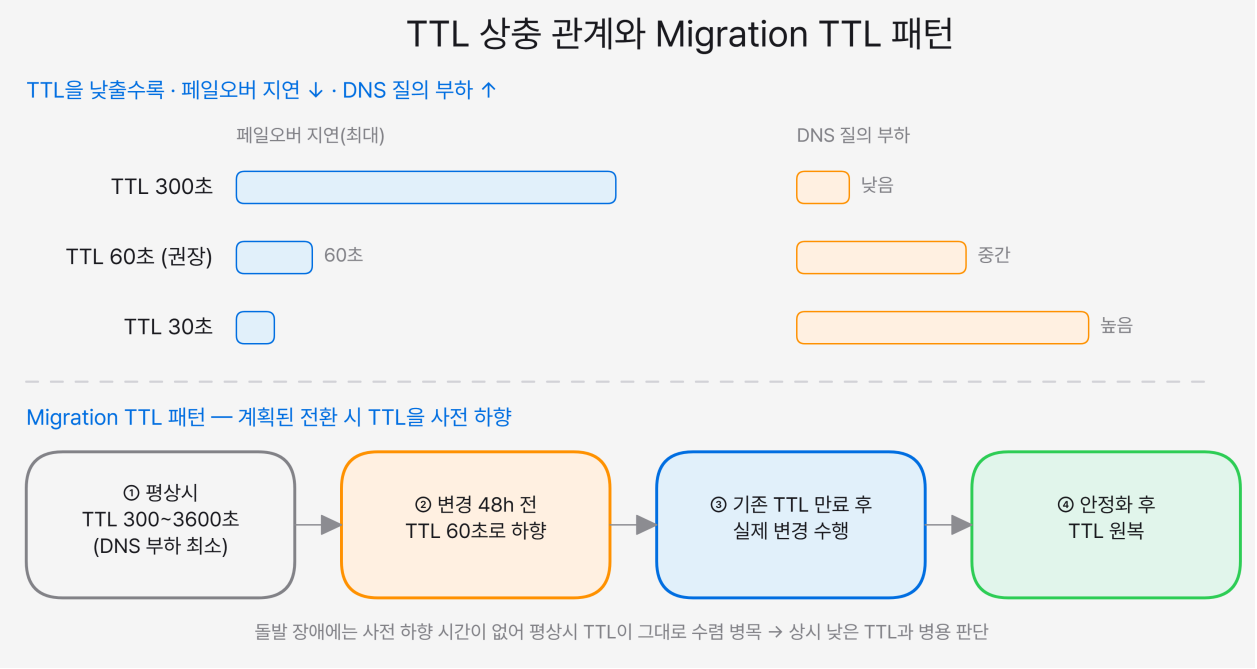

그림. TTL 값(300초·60초·30초)에 따른 페일오버 최대·평균 지연과 DNS 질의 부하의 상충 관계, 그리고 Migration TTL 패턴의 사전 하향 시간축.

6.2.2 다중 A레코드와 클라이언트 재시도

TTL을 아무리 조정해도 브라우저와 OS 캐시까지 통제할 수는 없으므로, DNS 계층만으로 완전한 페일오버를 보장하기는 어렵습니다[S4]. 여기서 부족분을 메우는 것이 클라이언트 계층의 보완입니다. 첫 번째 수단은 다중 A레코드입니다. GSLB가 정상 사이트의 IP를 하나만 응답하는 대신 여러 개를 동시에 반환하면, 클라이언트는 첫 IP 연결이 실패했을 때 TTL 만료를 기다리지 않고 곧바로 다음 IP로 시도할 수 있습니다[S4]. 예를 들어 `app.example.com`에 대해 Site-A(10.1.1.1)와 Site-B(10.2.2.2)를 함께 응답하면, 클라이언트는 10.1.1.1에 연결 타임아웃(권장 3초)을 두고 시도한 뒤 실패 시 즉시 10.2.2.2로 넘어가 정상 서비스에 접속합니다[S4].

이 재시도 동작을 표준화한 것이 Happy Eyeballs 알고리즘(RFC 8305)입니다[S4][S8][E3]. 다중 A레코드로 받은 여러 목적지를 클라이언트가 순차로 또는 병렬로 시도해 먼저 연결되는 경로를 채택하는 방식으로, 장애 IP에 매달리는 대기 시간을 줄여 TTL 전파를 기다리지 않는 즉시 전환에 가깝게 만듭니다. 여기에 앱 레벨 Circuit Breaker를 더하면 보완이 완성됩니다. Circuit

Breaker(회로 차단기)는 특정 엔드포인트에서 연속 실패가 임계치를 넘으면 해당 대상을 일정 시간 차단해 두었다가 재시도하는 패턴으로, 이미 죽은 사이트로 요청을 반복해 던지며 스스로 지연을 키우는 상황을 막고 대체 경로로 빠르게 넘어가게 합니다[S4]. 연결 타임아웃 단축(3~5초), 재시도 간격을 점진적으로 늘리는 지수 백오프, 연결 실패 시 OS DNS 캐시를 무시하고 새로 질의하는 재질의도 같은 목적의 보완 수단입니다[S4]. 종합하면 전환의 실질적 주체는 DNS가 아니라 클라이언트 재시도입니다. GSLB의 낮은 TTL과 다중 A레코드가 무대를 깔아 주고, Happy Eyeballs와 Circuit Breaker를 갖춘 클라이언트가 그 위에서 실제 전환을 수행합니다. 이 역할 분담을 지켜야 DNS는 페일오버를 보조하되 페일오버 자체가 되지 않는다는 원칙이 설계에 반영됩니다[S4].

7장: 하드웨어 어플라이언스형 GSLB

GSLB를 구현하는 첫 번째 계층은 전용 하드웨어(또는 가상) 어플라이언스입니다. 어플라이언스형은 ADC(Application Delivery Controller, 애플리케이션 전달 컨트롤러)라는 통합 장비 안에서 로컬 로드밸런싱과 글로벌 로드밸런싱을 함께 제공하며, 기업이 자체 데이터센터에 장비를 두고 직접 운영하는 온프레미스 방식입니다. 이 계층은 초고성능 DNS 처리와 세밀한 정책 제어, 그리고 데이터 주권·망분리 같은 규제 요건을 충족하는 데 강점이 있습니다. 반면 장비 도입에 따른 CapEx(자본적 지출) 부담이 크고, iRules 같은 커스터마이징을 다룰 전문 인력을 전제하며, 용량 증설이 하드웨어 교체·증설에 묶여 자동 확장이 어렵다는 한계가 함께 따라옵니다 [S5]. 본 장은 F5·Citrix·A10·Kemp 네 제품의 특성을 정리한 뒤, 어떤 조직이 어떤 요건에서 어플라이언스를 정당화할 수 있는지 판단 기준을 제시합니다. 8장(클라우드 매니지드형)·9장(쿠버네티스 네이티브형)과 대비되는 첫 번째 축으로 읽으시면 됩니다.

7.1 대표 제품과 특성

7.1.1 F5 BIG-IP DNS·Citrix NetScaler ADC

어플라이언스 시장에서 성능과 기능의 상한을 정하는 제품은 F5 BIG-IP DNS와 Citrix NetScaler ADC입니다. F5 BIG-IP DNS는 과거 GTM(Global Traffic Manager, 글로벌 트래픽 관리자)이라는 이름으로 알려졌던 제품으로, GSLB 분야의 사실상 기준선 역할을 해 왔습니다. 가장 두드러진 특성은 DNS 처리 성능입니다. F5는 DNS Express 기능으로 초당 최대 1억 쿼리(QPS, Queries Per Second)를 처리한다고 밝히고 있으며, 이는 대형 통신사나 포털급 질의 부하를 단일 장비 계층에서 감당하기 위한 수치입니다 [S5]. 여기에 위치·서버 상태·성능을 종합해 트래픽을 분배하는 지능형 GSLB, DNS 캐시 포이즈닝과 중간자 공격을 막는 DNSSEC(DNS Security Extensions), DNS 방화벽 기반의 DDoS 방어가 더해집니다. F5의 차별점은 iRules입니다. iRules는 TCL(Tool Command Language) 기반 스크립팅으로, 표준 정책만으로 표현하기 어려운 커스텀 트래픽 로직을 장비 위에서 직접 구현하게 해 줍니다. Wide IP와 Pool 구성을 통해 세밀한 분배 정책을 잡을 수 있고, BIG-IP LTM·APM·AFM·ASM과 통합되어 로컬 로드밸런싱부터 접근 제어·방화벽·웹 방화벽까지 하나의 ADC 플랫폼으로 묶입니다 [S5]. 자동화는 iControl REST API로 제공됩니다. 다만 이 모든 기능의 대가로 라이선스·하드웨어·지원 비용이 업계 최고 수준이며, 첫 구성 시 학습 곡선이 가파르다는 점이 반복적으로 지적됩니다 [S5]. Citrix NetScaler ADC(현 NetScaler)는 통합 ADC 안에서 GSLB를 제공하되, 사이트 간 협조 방식에서 F5와 결이 다릅니다. 핵심은 MEP(Metric Exchange Protocol, 메트릭 교환 프로토콜)로, GSLB 사이트들이 서로의 부하·상태 메트릭을 실시간으로 교환해 라우팅 결정에 반영합니다. 정책으로는 라운드로빈·최소연결(Least Connection)·근접성(Proximity)·정적 근접성·소스IP 해시를 지원하며, StyleBook이라는 템플릿 기반 관리로 온프레미스와 클라우드 설정을 통합하고 사이트 간 GSLB 구성을 자동 동기화합니다 [S5]. NetScaler의 결정적 강점은 Citrix Virtual Apps/Desktops(VDI, 가상 데스크톱 인프라) 환경과의 통합이며, 동적 서비스 디스커버리(DBS)로 클라우드 로드밸런서를 자동 탐지합니다. 자동화 인터페이스는 NITRO API입니다. 요약하면 초고성능·무한 커스터마이징·기존 F5 자산 연계가 필요하면 F5, VDI 통합과 사이트 간 실시간 메트릭 교환이 중요하면 NetScaler가 후보가 됩니다.

다음은 두 제품의 특성을 정리한 표입니다.

항목	F5 BIG-IP DNS	Citrix NetScaler ADC
DNS 처리 성능	최대 1억 QPS(DNS Express)	높음
커스터마이징	iRules(TCL, 매우 높음)	정책·StyleBook 기반
사이트 간 상태 교환	헬스체크 기반	MEP 실시간 메트릭 교환
통합 ADC	LTM/APM/AFM/ASM	L4-L7/SSL/WAF
자동화 API	iControl REST	NITRO API
대표 강점	초고성능·레퍼런스	VDI 통합·자동 동기화
라이선스	영구 + 연 유지보수, 구독(SaaS)	Advanced/Premium 에디션

출처: S5 (GSLB 제품 비교 분석 2.1·2.2)

7.1.2 A10 Thunder·Kemp LoadMaster GEO

F5·Citrix가 성능과 기능의 상한을 정한다면, A10 Thunder ADC와 Kemp LoadMaster GEO는 비용과 운영 편의성 축에서 현실적인 대안을 제시합니다. A10 Thunder ADC의 특징은 GSLB 구성 방식의 유연성입니다. A10은 DNS Proxy 모드와 DNS Server 모드라는 이중 모드를 제공해, 기존 DNS 인프라 앞단에 프록시로 얹을지, 장비 자체를 권한 DNS 서버로 둘지를 환경에 맞게 고를 수 있습니다 [S5]. 서버·사이트 상태를 실시간 메트릭으로 감지해 자동 전환하며, SLB(Server Load Balancing, 서버 로드밸런싱)와 GSLB를 단일 플랫폼에서 통합 운영합니다. 장비는 ADC 전용 운영체제인 ACOS(Advanced Core Operating System) 위에서 동작하고, aGalaxy/Harmony Controller로 중앙 집중 관리를, Thunder TPS 연동으로 DDoS 완화를 제공하며, 자동화는 aXAPI로 이루어집니다. 실무 관점의 강점은 F5 대비 구성이 간편하다는 점입니다. 와일드카드 비트 지정 같은 번거로운 절차 없이 GSLB를 잡을 수 있고, 장기 무중단 운영 사례가 많아 안정성 평판이 높으며, 가격이 F5보다 경쟁력이 있습니다 [S5]. 다만 로깅 용량 제한으로 고트래픽 상황에서 로그가 유실될 수 있고, 트래픽 분석 기능과 KB 문서가 부족해 기술지원 의존도가 높다는 점은 도입 전 확인이 필요합니다 [S5]. Kemp LoadMaster GEO는 네 제품 중 가성비를 대표합니다. GEO 기능은 위치 기반 라우팅과 근접성 라우팅을 중심으로 하며, 근접성 판단에 경도·위도 좌표를 사용해 클라이언트와 데이터센터를 매칭합니다. 실시간 트래픽 수요에 따라 세션을 동적으로 재분배하는 적응형 라우팅(Adaptive Routing), 클라이언트 서브넷 단위로 정밀하게 응답을 조정하는 EDNS(Extension Mechanisms for DNS, DNS 확장 메커니즘), DNSSEC를 지원합니다 [S5]. 자동화는 PowerShell·Ansible·RESTful API로 제공되고, WAF·IPS가 내장되며, Exchange·Skype for Business 같은 Microsoft 애플리케이션용 사전 구성 템플릿을 갖춘 점이 두드러집니다. Kemp의 강점은 F5·Citrix 대비 현저히 낮은 비용과 빠른 배포입니다. 가상 어플라이언스를 제공해 유연하게 배치할 수 있고, Microsoft 인프라(Exchange·Active Directory) 중심 환경에 잘 맞습니다 [S5]. 반대로 복잡한 구성에서는 문서가 부족하거나 오래되었을 수 있고, UI/UX가 다소 구식이며, 로깅·분석·실시간 가시성은 외부 도구 연동을 전제로 봐야 합니다 [S5]. 정리하면, F5 수준의 성능이 필요하지 않은 중견기업·ISP에는 A10이, 예산이 제한되고 빠른 도입과 Microsoft 통합이 중요한 중소·중견 환경에는 Kemp가 합리적인 선택지가 됩니다.

항목	A10 Thunder ADC	Kemp LoadMaster GEO
GSLB 구성	DNS Proxy/Server 이중 모드	위치·근접성·적응형
통합 기능	SLB+GSLB, DDoS(TPS 연동)	SLB, WAF/IPS
자동화	aXAPI	PowerShell/Ansible/REST
대표 강점	간편 구성·안정성·중간 가격	최고 가성비·빠른 배포·MS 통합
유의점	로깅·분석 제한	문서·가시성 부족
적합 환경	중견기업·통신사/ISP	중소기업·Microsoft 환경

출처: S5 (GSLB 제품 비교 분석 2.3·2.4)

7.2 도입 적합성과 운영 부담

7.2.1 CapEx·전문 인력 요건

어플라이언스형의 비용은 절대 금액보다 비용 구조로 이해해야 판단이 정확합니다. 이 계층의 지출은 대부분 CapEx(자본적 지출) 성격을 띱니다. 장비를 구매해 자산으로 계상하고, 소프트웨어는 영구 라이선스로 취득하며, 여기에 연간 유지보수 계약이 붙는 구조입니다 [S5]. F5는 이 구조에서 라이선스·하드웨어·지원 비용이 모두 업계 최고 수준으로, 초기 투자 부담이 네 제품 중 가장 큽니다. Citrix NetScaler는 GSLB를 쓰려면 Advanced 또는 Premium 에디션이 필요해 진입 비용이 올라가고, A10은 중간 수준, Kemp는 가장 낮은 비용대에 위치합니다 [S5]. 최근에는 구독형(SaaS)·가상 어플라이언스(VE/VPX) 배포 옵션이 늘어 초기 CapEx를 OpEx(운영적 지출)로 일부 전환할 여지가 생겼지만, 기본 성격은 여전히 선투자 중심입니다. 어플라이언스형 비용을 평가할 때 자주 과소평가되는 항목은 인력 비용입니다. 총소유비용에서 장비·라이선스 못지않게 비중이 큰 것이 운영 인력이며, 이는 제품이 제공하는 커스터마이징 자유도와 정비례합니다. F5의 iRules는 TCL 스크립트로 사실상 무한한 트래픽 로직을 구현할 수 있다는 것이 강점이지만, 뒤집어 보면 그 로직을 설계·검증·유지할 수 있는 전문 인력을 조직이 상시 확보해야 한다는 요건이기도 합니다 [S5]. F5는 첫 구성의 학습 곡선이 가파르다는 지적이 반복되고, Citrix NetScaler 역시 구성 복잡도가 높아 전문 인력을 필요로 합니다 [S5]. 반대로 A10과 Kemp는 구성이 간편해 진입 문턱이 낮은 대신, 심화 문제 대응 시 A10은 부족한 분석 기능과 KB 문서 때문에 기술지원 의존도가 높아지고, Kemp는 복잡한 구성에서 문서가 부족해 시행착오 비용이 발생할 수 있습니다 [S5]. 결과적으로 어플라이언스형의 총소유비용은 "장비값 + 라이선스 + 연 유지보수 + 이를 운용할 상시 인력"의 합으로 봐야 하며, 특히 고급 제품일수록 인력 비용의 비중이 커집니다. 도입 검토 시에는 3~5년의 운영 기간 전체에 걸쳐 이 네 항목을 함께 산정하고, 자사가 iRules급 커스터마이징을 실제로 활용할 조직 역량을 갖췄는지를 먼저 점검하는 것이 순서입니다. 활용하지 못할 자유도를 위해 상위 제품과 상위 인력을 동시에 사는 것은 과잉 투자가 되기 쉽습니다.

7.2.2 적합 조직과 확장 한계

어플라이언스형이 정당화되는 조직은 비교적 뚜렷합니다. 첫째는 초고성능이 필요한 대기업·통신사·금융권입니다. F5 BIG-IP DNS의 최대 1억 QPS급 DNS 처리와 iRules 커스터마이징은 대규

모 멀티 데이터센터 GSLB나 통신사·금융권 수준의 질의 부하를 단일 계층에서 감당하려는 환경에서 값을 합니다 [S5]. 둘째는 규제 요건이 계층 선택을 강제하는 경우입니다. 데이터 주권, 망 분리, 컴플라이언스 요구가 강한 조직은 트래픽 분산 계층 자체를 자사 통제 아래 온프레미스로 두어야 할 수 있고, 이때 매니지드 클라우드형은 선택지에서 빠집니다 [S5]. 셋째는 기존 자산과의 정합성입니다. 이미 F5 BIG-IP LTM 인프라를 보유했다면 BIG-IP DNS로, Citrix VDI를 운영 중이라면 NetScaler로 확장하는 편이 통합 운영 측면에서 유리합니다 [S5]. 반대로 어플라이언스형의 구조적 약점은 확장성에 있습니다. 하드웨어 어플라이언스는 용량이 장비 사양에 묶이므로, 트래픽이 장비 한계를 넘으면 상위 모델로 교체하거나 장비를 증설해 다시 구성해야 합니다. F5 문서에서도 하드웨어 어플라이언스 기반일 때 확장성이 제한된다는 점이, NetScaler에서도 용량 확장 시 추가 투자가 필요하다는 점이 단점으로 지적됩니다 [S5]. 이 확장 방식은 대체로 수동적입니다. 5장·6장에서 본 Active-Active의 확장 이점, 즉 사이트를 더하며 수평으로 넓히는 방식은 부하가 늘 때 인프라가 상시 활용되며 자연스럽게 흡수되는 그림이지만 [S2], 어플라이언스형에서는 그 수평 확장이 하드웨어 증설·재구성이라는 수동 절차로 치환됩니다. 자동 스케일링을 전제하는 클라우드 매니지드형이나, 클러스터를 더하는 방식으로 확장하는 쿠버네티스 네이티브형과 비교하면 자동화·탄력성에서 상대적으로 뒤처지는 지점입니다. 이 확장 한계는 10장의 방식별 종합 비교표에서 "확장·자동화" 항목으로 다시 다루므로, 여기서는 어플라이언스형이 성능·규제·기존 자산이라는 세 조건에서 강하고, 탄력적 확장이 필요한 환경에서는 약하다는 대비만 확인해 두면 됩니다. 다음은 적합 시나리오를 정리한 것입니다.

시나리오	권고 제품	근거
대규모 엔터프라이즈·통신사	F5 BIG-IP DNS	최고 성능·iRules·풍부한 레퍼런스
Citrix VDI·하이브리드	Citrix NetScaler	VDI 통합·MEP·자동 동기화
비용 효율 + 간편 운영	A10 Thunder ADC	F5 대비 저렴·구성 간편·안정성
예산 제한 + 빠른 도입	Kemp LoadMaster GEO	최저 비용·빠른 배포·MS 통합
규제·데이터 주권 요건	온프레미스 어플라이언스 전반	자사 통제하 트래픽 계층 확보

출처: S5(2.1~2.4·4·5), S2(4.1 확장성)

8장: 클라우드 매니지드형 GSLB

7장의 어플라이언스가 하드웨어 자산과 전문 인력을 전제로 한다면, 클라우드 매니지드형 GSLB는 그 전제를 걷어 낸다. 사업자가 전 세계 DNS 인프라를 대신 운영하고, 도입 조직은 라우팅 정책과 헬스체크만 선언하면 된다. 초기 투자와 운영 부담이 낮아 도입 문턱이 가장 낮은 방식이지만, 질의·트래픽 종량 과금이라는 성격상 규모가 커질수록 운영비(OpEx)가 누증하고, 선택한 사업자에 따라 CSP(Cloud Service Provider, 클라우드 서비스 제공사) 종속이 생긴다는 상충이 뒤따른다. 이 장은 매니지드형을 두 갈래로 나눠 살핀다. 하나는 특정 CSP가 자사 생태계에 맞춰 제공하는 네이티브 서비스이고, 다른 하나는 어느 클라우드에도 묶이지 않는 벤더 중립 매니지드 서비스다. 도입 조직이 단일 CSP 표준화 경로를 갈지, 멀티클라우드를 아우르는 중립 경로를 갈지 판단하는 데 필요한 기능·과금·적합성 정보를 제품 단위로 정리합니다.

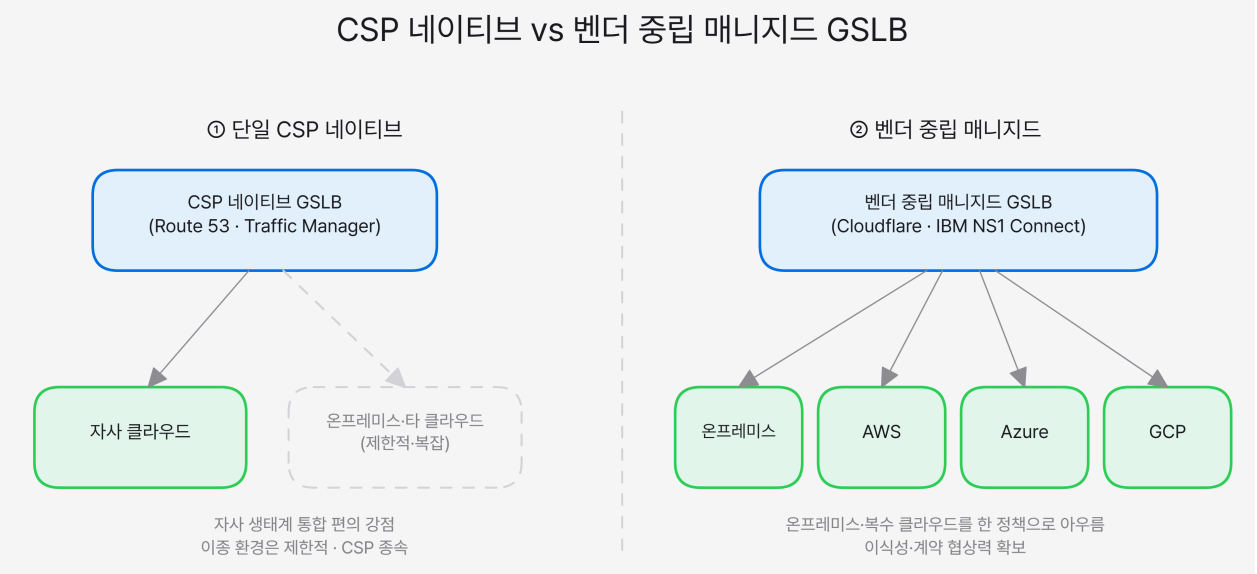

그림. 단일 CSP 네이티브 GSLB와 벤더 중립 매니지드 GSLB가 온프레미스·복수 클라우드를 각각 어떻게 아우르는지 대비한 배치도(11_multi_cloud.png 재활용).

8.1 CSP 네이티브 서비스

8.1.1 AWS Route 53·Azure Traffic Manager

AWS Route 53은 관리형 DNS와 GSLB를 결합한 서비스로, 라우팅 정책을 7종 제공합니다. 단순(Simple)·장애조치(Failover)·지리적 위치(Geolocation)·지리적 근접성(Geoproximity)·레이턴시(Latency)·다중값(Multivalue)·가중치(Weighted) 라우팅이며, 이들을 Traffic Flow라는 시각적 편집기로 조합해 복잡한 분산 정책을 트리 형태로 설계할 수 있습니다 [S5]. 헬스체크는 HTTP·HTTPS·TCP 기반 엔드포인트 점검을 지원하고 CloudWatch와 연동되며, Application Recovery Controller로 애플리케이션 복원력을 모니터링합니다. 무엇보다 ELB·S3·CloudFront 등 AWS 서비스와 네이티브로 맞물려, AWS 위에서 워크로드를 운영하는 조직은 별도 인프라를 세우지 않고도 멀티리전 재해복구(DR) 구성을 얹을 수 있습니다 [S5]. 다만 순수 DNS 방식이라 TTL(Time To Live, DNS 응답 캐시 유효시간) 캐싱에 따른 장애조치 지연이 있고, 가중치를 리전

단위로만 조정하는 All-or-None 성격 때문에 부분 비율 조정이 어려우며, AWS 외부 환경에서는 기능이 제약되고 실시간 트래픽 스티어링(non-DNS)을 지원하지 않습니다 [S5]. 이 서비스가 가장 잘 맞는 곳은 AWS 중심으로 인프라를 표준화한 조직, 그리고 중소 규모 글로벌 서비스의 DNS 기반 로드밸런싱입니다.

Azure Traffic Manager는 DNS 기반 트래픽 로드밸런서로, 라우팅 방식을 6종 제공합니다. 우선순위(Priority)·성능(Performance)·지리적(Geographic)·가중치 라운드로빈(Weighted)·서브넷(Subnet)·다중값(MultiValue)이며, 중첩 프로필(Nested Profiles)로 이들을 계층 구조로 조합할 수 있습니다 [S5]. 이 제품의 차별점은 외부 엔드포인트 지원입니다. Azure 외부, 즉 온프레미스나 타 클라우드에 있는 엔드포인트를 프로필에 등록할 수 있어, 버스트-투-클라우드나 클라우드 마이그레이션, 온프레미스-클라우드 간 페일오버 같은 하이브리드 시나리오를 단일 CSP 서비스로 감당합니다 [S5]. 실사용자 측정(Real User Measurements)으로 실제 레이턴시 기반 라우팅도 최적화합니다. 한계는 뚜렷합니다. DNS 계층만 담당하므로 L7 로드밸런싱은 Application Gateway로 분리되고, 헬스체크가 HTTP·HTTPS·TCP로 제한되어 UDP·ICMP를 다루지 못하며, SSL 오프로딩이나 WAF(Web Application Firewall, 웹 애플리케이션 방화벽) 같은 부가 기능이 없습니다. 더 정교한 진입 계층이 필요하면 Front Door를 함께 쓰는 구성이 일반적입니다. 두 제품 모두 완전 관리형이며 종량 과금이라 초기 비용 부담이 없다는 점은 공통이되, 하이브리드 유연성에서는 외부 엔드포인트를 정식 지원하는 Azure 쪽이 앞섭니다.

비교 항목	AWS Route 53	Azure Traffic Manager
라우팅 방식 수	7종	6종
정책 편집	Traffic Flow 시각 편집기	중첩 프로필 계층
헬스체크	HTTP/HTTPS/TCP + CloudWatch	HTTP/HTTPS/TCP
외부(Non-CSP) 엔드포인트	제한적	정식 지원
Anycast	지원	미지원
SLA	100%	99.99%
최적 환경	AWS 표준화 조직	Azure·하이브리드

8.1.2 GCP Cloud DNS+LB (Anycast)

GCP(Google Cloud Platform)의 GSLB는 앞의 두 제품과 동작 원리가 다릅니다. Route 53과 Traffic Manager가 DNS 응답으로 사이트 IP를 골라 주는 방식이라면, GCP는 Cloud DNS와 Cloud Load Balancing을 묶어 단일 Anycast IP를 앞세웁니다 [S5]. 전 세계에서 동일한 하나의 글로벌 IP를 광고하고, 클라이언트의 질의가 Google 백본 네트워크의 가장 가까운 진입점으로 흡수된 뒤 내부에서 건강한 백엔드로 분배됩니다. 이 구조의 이점은 3장에서 다룬 Anycast 특성 그대로입니다. 사이트를 바꾸는 결정이 DNS 응답이 아니라 네트워크 경로 재수렴으로 이뤄지므로, 클라이언트나 중간 리졸버에 캐시된 TTL이 만료되기를 기다릴 필요가 없습니다 [S5]. 결과적으로 순수 DNS형보다 장애조치가 빠르고, 사전 워밍(pre-warming)도 불필요하며, DNS 설정이

단일 IP 하나로 단순해집니다. Google 프리미엄 티어 네트워크와 202개가 넘는 PoP(Point of Presence, 네트워크 접속 거점)를 활용해 글로벌 레이턴시도 낮게 유지합니다 [S5].

3장에서 "지능적 정책 기반 분산은 DNS가, 빠른 네트워크 레벨 전환은 Anycast가 담당한다"고 정리한 역할 분담이 GCP 제품에서 실제로 구현된 셈입니다. 다만 이 방식이 만능은 아닙니다. 강점의 원천인 Google 백본에 그만큼 의존하므로 GCP 생태계 종속도가 높고, 외부(Non-GCP) 엔드포인트 연동이 복잡해 멀티클라우드 활용이 제한됩니다 [S5]. Cloud DNS만 단독으로 쓰면 GSLB 기능이 제한되어 Cloud LB와 반드시 결합해야 하고, 설정의 자유도가 높은 만큼 학습 곡선과 초기 구성 복잡도가 상대적으로 큽니다. 온프레미스와의 하이브리드 구성에도 추가 구성 요소가 필요합니다. 따라서 GCP 네이티브형은 GCP 위에서 워크로드를 운영하며 단일 IP 기반의 글로벌 저지연 서비스가 필요한 조직에 가장 잘 맞습니다. Anycast형 매니지드 서비스라는 같은 범주에는 뒤 절에서 다룰 Cloudflare도 포함되며, 순수 DNS형인 Route 53·Traffic Manager와는 페일오버 속도의 결정 요인이 근본적으로 다르다는 점을 기억해 둘 필요가 있습니다. 이 차이는 10장 비교표에서 페일오버 속도 항목으로 다시 수렴합니다.

8.2 벤더 중립 매니지드 서비스

8.2.1 Cloudflare·IBM NS1 Connect

Cloudflare Load Balancing은 DNS 기반 글로벌 로드밸런싱에 CDN(Content Delivery Network, 콘텐츠 전송망)과 보안을 통합한 올인원 서비스입니다. 330개가 넘는 도시의 PoP에서 트래픽을 분배해 DNS 응답이 매우 빠르고, 레이턴시·지리적 리전·GPS 좌표·가중치 기반 스티어링 정책을 제공하며, HTTP·HTTPS 헬스체크를 여러 데이터센터에서 교차 검증합니다 [S5]. 핵심 강점은 DDoS 방어와 WAF, CDN이 로드밸런싱과 한 계층에서 맞물린다는 점, 그리고 오리진 IP 마스킹으로 원본 서버를 가린다는 점입니다. 무엇보다 특정 CSP에 묶이지 않는 벤더 중립성이 두드러져, 클라우드와 온프레미스를 혼합한 멀티클라우드·하이브리드 환경의 로드밸런싱에 잘 맞습니다 [S5]. 한계는 비Enterprise 플랜에서 지원 DNS 레코드 유형이 A·AAAA·CNAME으로 제한되고, Proxy(Orange Cloud) 사용 시 DNS 라운드로빈 페일오버에 제약이 생기며, DNS 기반 특성상 세밀한 실시간 트래픽 제어에는 한계가 있다는 점입니다.

IBM NS1 Connect(구 NS1)는 관리형 DNS에 지능형 트래픽 스티어링을 얹은 서비스로, 접근 방식이 앞의 제품들과 결이 다릅니다. 라우팅을 고정된 정책 몇 종으로 제공하는 대신, Filter Chain이라는 다단계 필터 조합으로 라우팅 로직을 사실상 무제한으로 구성합니다 [S5]. 여기에 실사용자 모니터링(RUM, Real User Monitoring) 데이터를 결합해 실제 사용자 성능을 근거로 엔드포인트를 고르고, Pulsar로 실시간 디바이스 성능까지 반영합니다. 완전한 RESTful API를 앞세운 API 우선 설계라 DevOps·IaC(Infrastructure as Code, 코드형 인프라) 자동화에 친화적이고, 100% DNS 가동시간 SLA와 벤더 중립 멀티클라우드 지원을 갖췄습니다 [S5]. 다만 글로벌 PoP이 6대륙 26개로 Cloudflare(330+)에 비해 적고, L4·L7 로드밸런싱이나 WAF 같은 부가 기능 없이 DNS 스티어링에 특화되어 있으며, 소규모 조직에는 기능이 과할 수 있습니다. 두 제품 모두 벤더 중립이라는 공통점 아래, Cloudflare는 보안·CDN 통합과 방대한 PoP으로, NS1 Connect는 Filter Chain·RUM 기반의 정밀 스티어링과 API 우선으로 각자의 자리를 갖습니다.

비교 항목	Cloudflare	IBM NS1 Connect
라우팅 구성	4종+ 스티어링 정책	Filter Chain(무제한 조합)
글로벌 PoP	330+	26
보안 통합	CDN·DDoS·WAF 내장	DDoS 내장, WAF 없음
지능형 근거	헬스체크·지리	RUM·실시간 성능
자동화	API 지원	API 우선 설계
최적 환경	멀티클라우드·웹·보안	복잡한 트래픽 엔지니어링

8.2.2 OpEx 종량 비용과 하이브리드 적합성

매니지드형 GSLB의 과금은 어플라이언스의 CapEx와 성격이 정반대입니다. 하드웨어 구매와 영구 라이선스로 선투자하는 대신, 처리한 DNS 질의 수와 헬스체크·트래픽 볼륨에 비례해 매월 운영비(OpEx)로 지불합니다. Route 53·Traffic Manager·GCP는 종량제, Cloudflare는 플랜 기반, NS1 Connect는 쿼리 볼륨 기반으로 세부 방식은 다르지만, 사용량이 곧 비용이라는 구조는 공통입니다 [S5]. 이 구조는 도입 초기에 유리합니다. 초기 자본 지출과 유지보수 계약이 없고, 트래픽이 적은 파일럿 단계에서는 지불액도 작아 시작 문턱이 가장 낮습니다. 문제는 규모가 커질 때 드러납니다. 질의량과 트래픽이 선형으로 늘면 청구액도 함께 누증하므로, 대규모·고트래픽 서비스에서는 종량 비용이 어느 지점부터 어플라이언스의 CapEx 상각분을 역전할 수 있습니다. 초기의 저비용과 규모 확대 시의 비용 증가가 상충하는 셈이며, 이 손익분기 규모를 자사 트래픽 예측에 대입해 가늠하는 일이 매니지드형 선택의 핵심 판단입니다. 비용을 절대 금액이 아니라 트래픽 규모 대비 상대 부담으로 놓고 봐야 이 역전 지점이 보입니다.

하이브리드 적합성은 제품에 따라 갈립니다. 단일 CSP 네이티브 서비스는 자사 클라우드 내부 통합에 강하지만 이종 환경을 아우르기는 어렵고, Azure Traffic Manager처럼 외부 엔드포인트를 정식 지원하는 경우가 예외적으로 하이브리드에 열려 있습니다 [S5]. 반면 Cloudflare·NS1 Connect 같은 벤더 중립 서비스는 온프레미스와 복수 클라우드를 한 정책 아래 아우르도록 설계되어, 특정 사업자에 협상력을 뺏기지 않고 이식성을 확보하려는 조직에 실익이 큽니다. 이 벤더 중립의 실익은 단순한 기능 문제가 아니라, 계약 협상력과 워크로드 이동 자유도로 이어집니다. 쿠버네티스로 표준화된 환경이라면 매니지드형 대신 9장에서 다룰 클러스터 네이티브 방식(K8GB)도 하이브리드 선택지에 들어오는데, 이는 라이선스 비용이 없는 대신 플랫폼 운영 역량을 전제한다는 점에서 매니지드형과 트레이드오프가 다릅니다 [S6]. 정리하면 매니지드형은 운영 부담을 사업자에게 넘겨 초기 도입을 쉽게 하되, 규모에 따른 OpEx 누증과 CSP 종속이라는 대가를 함께 안습니다. 단일 CSP로 표준화된 조직은 네이티브 서비스의 통합 편의를, 멀티클라우드를 지향하는 조직은 벤더 중립 서비스의 이식성을 각각 우선해 방향을 정하면 됩니다.

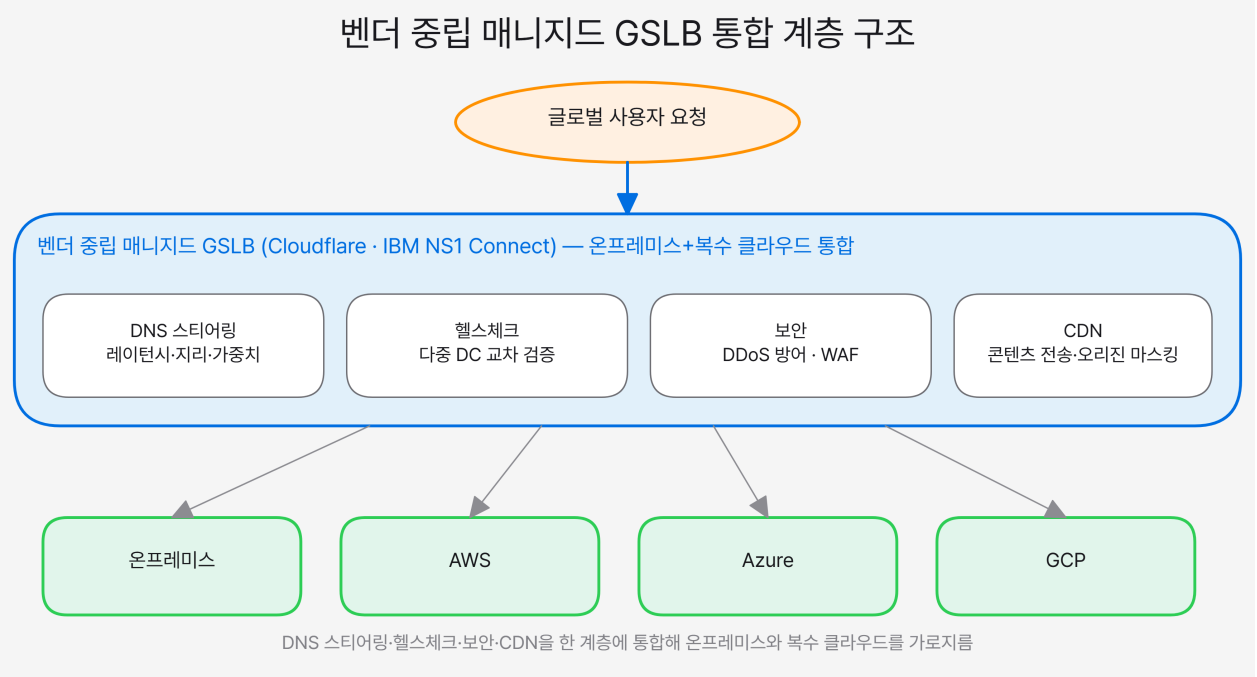

그림. 벤더 중립 매니지드 GSLB가 DNS 스티어링·헬스체크·보안(DDoS/WAF)·CDN 계층을 통합해 온프레미스와 복수 클라우드를 가로지르는 계층 구조(12_vendor_neutral_layers.png 재활용).

9장: 쿠버네티스 네이티브형 GSLB (K8GB)

7장의 하드웨어 어플라이언스와 8장의 클라우드 매니지드 서비스는 모두 클러스터 바깥에 트래픽 분산 계층을 두는 방식입니다. 쿠버네티스(Kubernetes)로 서비스를 표준화한 조직에서는 그 분산 계층 자체를 클러스터 리소스로 끌어들일 수 있습니다. K8GB(Kubernetes Global Balancer)는 CNCF(Cloud Native Computing Foundation) Sandbox 프로젝트로, 지리적으로 분산된 클러스터 사이의 글로벌 로드밸런싱을 완전 오픈소스로 구현합니다 [S6]. F5 BIG-IP나 Citrix 같은 상용 GSLB 장비를 대체해 쿠버네티스 네이티브(cluster-native, 클러스터 자원으로 동작하는) 방식으로 트래픽을 나눕니다. 라이선스 비용이 들지 않고 구성을 Git 저장소로 선언적 관리할 수 있어 일관성이 뛰어난 대신, 쿠버네티스와 플랫폼 엔지니어링 역량을 전제로 한다는 점이 이 장의 핵심입니다. 이 장은 K8GB의 구성 요소와 로드밸런싱 전략을 먼저 정리하고(9.1), GitOps(Git을 단일 소스로 삼는 운영 방식)와 서비스 메시(Service Mesh) 연계로 넘어갑니다(9.2).

9.1 K8GB 아키텍처

9.1.1 CoreDNS 임베드·컨트롤러·ExternalDNS

K8GB는 세 가지 핵심 구성 요소로 이루어집니다 [S6]. 첫째는 **k8gb Controller**로, 사용자가 정의한 Gslb CRD(Custom Resource Definition, 쿠버네티스에 추가하는 사용자 정의 리소스 유형)를 감시하면서 애플리케이션 Pod의 Readiness Probe(준비 상태 점검) 결과에 따라 DNS 레코드를 갱신합니다. 둘째는 각 클러스터에 **임베드된 CoreDNS**로, 로드밸런싱 존(zone)을 직접 호스팅하며 클라이언트 DNS 질의에 응답합니다. CoreDNS는 쿠버네티스 표준 DNS 서버로, 여기서는 GSLB 응답을 내보내는 권한 있는 네임서버 역할을 합니다. 셋째는 **ExternalDNS**로, 상위 Cloud DNS(Route 53, Google Cloud DNS, Infoblox 등)에 NS 위임(delegation) 레코드를 자동으로 만들어 상위 도메인의 질의가 K8GB의 CoreDNS로 흘러오도록 연결합니다.

이 구조의 이점은 별도 관리 계층이 없다는 데 있습니다. K8GB는 전용 관리 클러스터를 요구하지 않고 각 클러스터에 독립적으로 배포되므로, 특정 노드나 클러스터가 죽어도 전체 GSLB가 멈추는 단일 장애점(SPoF, Single Point of Failure)이 생기지 않습니다 [S6]. 어플라이언스형에서 GSLB 장비 자체가 이중화 대상이 되던 것과 대비됩니다. 클러스터 A(EU)와 클러스터 B(US)가 각자 k8gb Controller와 CoreDNS를 갖고 DNS로 서로의 상태를 교환하며, 어느 한쪽이 조율자가 되지 않는 대등 구조입니다.

또 하나의 특징은 헬스체크를 새로 만들지 않고 쿠버네티스가 이미 수행하는 점검을 그대로 재사용한다는 점입니다. Pod의 Readiness Probe가 실패하면 그 Pod는 Service Endpoint에서 빠지고, k8gb Controller가 이 변화를 감지해 CoreDNS의 A 레코드에서 해당 클러스터 IP를 제거합니다. 이후 DNS TTL(Time To Live, 캐시 유효 시간)이 만료되면 클라이언트는 건강한 다른 클러스터로 라우팅됩니다 [S6]. 4장에서 다룬 애플리케이션 레벨 헬스체크가 여기서는 Readiness Probe라는 쿠버네티스 기본 기능으로 곧장 이어지는 셈입니다. Probe의 `failureThreshold`(연속 실패 판정 횟수)를 2로 두면 두 번 연속 실패한 시점에 Endpoint에서 제거되므로, 감지 민감도를 애플리케이션 배포 매니페스트에서 직접 조정할 수 있습니다.

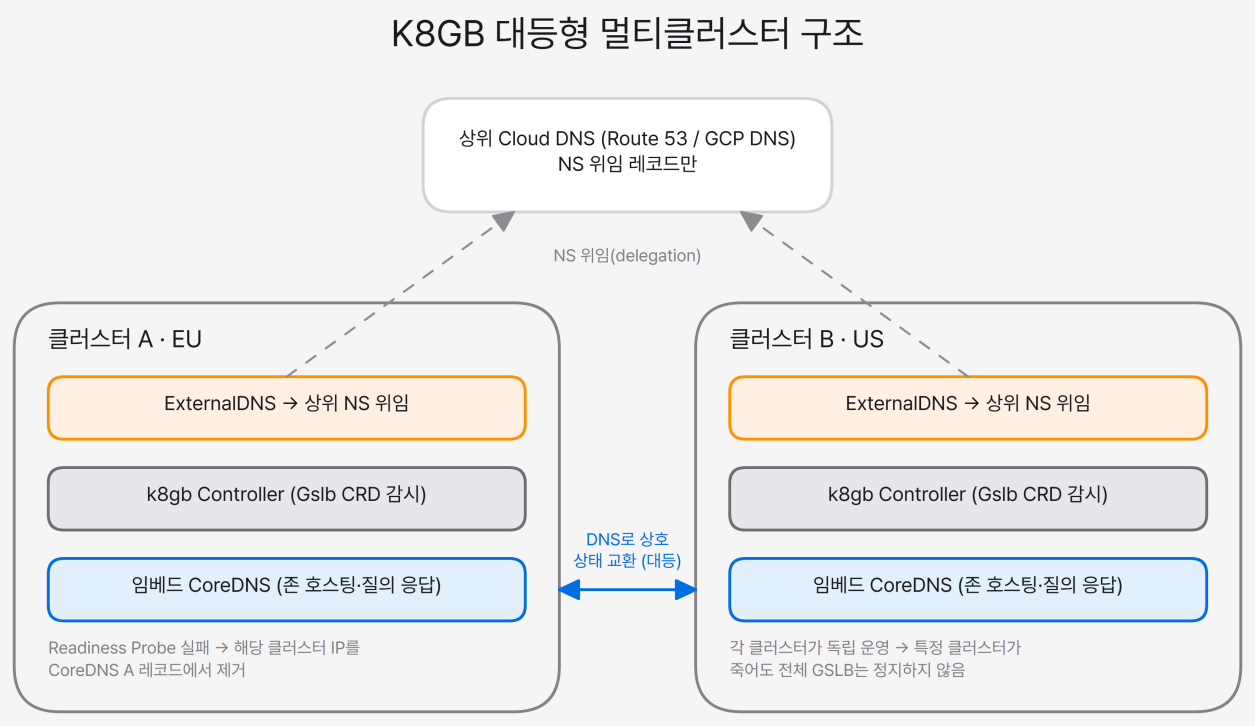

그림. K8GB의 대등형 멀티클러스터 구조 — 각 클러스터가 k8gb Controller, 임베드 CoreDNS, ExternalDNS를 독립적으로 운영하고 DNS로 상태를 교환하며, 상위 Cloud DNS에는 NS 위임 레코드만 둡니다.

9.1.2 로드밸런싱 전략 (failover·roundRobin·geoip)

K8GB의 분산 정책은 Gslb CRD의 `strategy.type` 필드로 지정하며, 네 가지 전략을 지원합니다[S6]. **failover**는 `primaryGeoTag`로 지정한 주요 리전을 우선 사용하고 그 리전이 장애일 때만 보조 리전으로 넘기는 방식으로, 평상시 한쪽에만 트래픽을 보내는 재해복구(DR) 시나리오에 맞습니다. **roundRobin**은 건강한 모든 클러스터에 질의를 균등하게 분배하므로 리전별 용량이 비슷한 글로벌 서비스에 적합합니다. **weightRoundRobin**은 클러스터마다 가중치를 부여해 트래픽 비율을 조절하며, 새 리전으로 트래픽을 조금씩 옮기는 카나리 배포나 점진적 마이그레이션에 씁니다. **geoip**는 클라이언트 위치를 기준으로 가장 가까운 클러스터로 라우팅해 지연을 줄입니다.

이 네 전략은 3장에서 정리한 DNS 분산 알고리즘과 그대로 대응합니다. failover는 Priority 기반 정책에, roundRobin은 Round Robin에, weightRoundRobin은 Weighted Round Robin에, geoip는 Geo/Proximity 정책에 1:1로 매핑됩니다. 즉 K8GB는 3장의 일반 분산 알고리즘을 쿠버네티스 CRD 필드로 옮겨 놓은 구현입니다. 전략과 함께 `dnsTtlSeconds`를 CRD에서 직접 지정하므로, 6장에서 다룬 낮은 TTL(예: 60초) 정책도 매니페스트 한 줄로 반영됩니다.

K8GB 전략	동작	3장 분산 알고리즘	적용 상황
`failover`	Primary 우선, 장애 시 Secondary 전환	Priority	DR(재해복구)

K8GB 전략	동작	3장 분산 알고리즘	적용 상황
roundRobin	건강한 클러스터 균등 분배	Round Robin	글로벌 트래픽 분산
weightRoundRobin	가중치 기반 비율 분배	Weighted RR	카나리·점진 마이그레이션
geoip	클라이언트 최근접 클러스터 라우팅	Geo/Proximity	지연 최소화

K8GB가 쿠버네티스 클러스터에 한정된다는 오해를 피하려면 CoreDNS-GSLB 플러그인을 함께 봐야 합니다. CoreDNS-GSLB는 CoreDNS에 GSLB 기능을 더하는 외부 플러그인으로, VM·베어메탈·하이브리드 클라우드 같은 비(非)쿠버네티스 인프라도 대상으로 삼습니다 [S6]. failover, roundrobin, random, weighted, geoip 모드를 지원하고, 백엔드를 YAML로 선언하며 HTTP·TCP 헬스체크 프로파일을 이름으로 참조해 재사용합니다. 또한 DNS 레코드가 일정 시간 조회되지 않으면 헬스체크 간격을 자동으로 늘려 불필요한 점검 트래픽을 줄이는 적응형 모니터링(Adaptive Monitoring)을 제공합니다 [S6]. 결국 쿠버네티스 워크로드는 K8GB로, 클러스터 밖 레거시 자원은 CoreDNS-GSLB로 같은 CoreDNS 기반 위에서 함께 다룰 수 있습니다.

9.2 GitOps·서비스 메시 연계

9.2.1 Helm·ArgoCD/Flux 선언적 운영

K8GB의 모든 설정은 쿠버네티스 CRD로 선언되므로, 기존 배포 워크플로우와 자연스럽게 맞물립니다 [S6]. 설치와 클러스터별 매개변수는 Helm 차트로 관리합니다. clusterGeoTag(자기 리전 태그), extGslbClustersGeoTags(상대 클러스터 태그), edgeDNSZone, dnsZone 등을 values.yaml로 넘기고, values-production-eu.yaml과 values-staging-eu.yaml처럼 환경별 파일을 분리해 운영과 스테이징을 같은 차트로 배포합니다 [S6]. Helm 대신 Kustomize를 쓰면 base/에 공통 Gslb 리소스를 두고 overlays/production-eu, overlays/production-us 오버레이에서 primaryGeoTag나 dnsTtlSeconds만 패치하는 방식으로 멀티클러스터 일관성을 유지합니다.

선언된 구성을 실제 클러스터에 반영하는 단계는 ArgoCD나 Flux 같은 GitOps 도구가 맡습니다. ArgoCD Application 정의에서 Git 저장소 경로(k8gb/overlays/production-eu)와 대상 클러스터를 지정하고 syncPolicy.automated에 prune: true, selfHeal: true를 켜면, Git 변경이 자동으로 클러스터에 동기화되고 누군가 클러스터를 손으로 바꿔 발생한 드리프트(drift, 선언 상태와 실제 상태의 어긋남)를 selfHeal이 원래대로 되돌립니다 [S6]. Flux Kustomization도 interval: 5m로 주기 동기화하며 healthChecks에 Gslb 리소스를 등록해 조정 결과를 확인합니다. 이렇게 Git이 GSLB 구성의 단일 소스(Single Source of Truth)가 되면, 누가 언제 무엇을 바꿨는지 커밋 이력으로 남아 감사 추적(Audit Trail)이 확보되고 롤백도 커밋 되돌리기로 처리됩니다.

여기서 더 나아간 사례가 Crossplane과의 결합입니다. 2025년 KubeCon China에서 제시된 레퍼런스 아키텍처는 Crossplane 기반 Global Control Plane과 K8GB를 묶어 멀티리전 인프라를 관리합니다 [S6]. Git 저장소에 Crossplane Composition, k8gb Helm Values, Gslb CRD 매니페스트를 함께 두고 ArgoCD/Flux가 각 리전의 Crossplane과 K8GB에 배포하는 구조로, 인프라 프로비

저닝과 글로벌 트래픽 관리를 하나의 선언적 파이프라인으로 통합합니다. 다만 이 모든 이점은 GitOps에 익숙한 조직을 전제로 합니다. Git 워크플로우, CRD 검증(kubeval·conftest), 오버레이 구조에 익숙하지 않은 팀에는 선언적 운영이 오히려 진입 장벽이 되므로, K8GB 채택 여부는 플랫폼 역량을 함께 저울질해 판단해야 합니다.

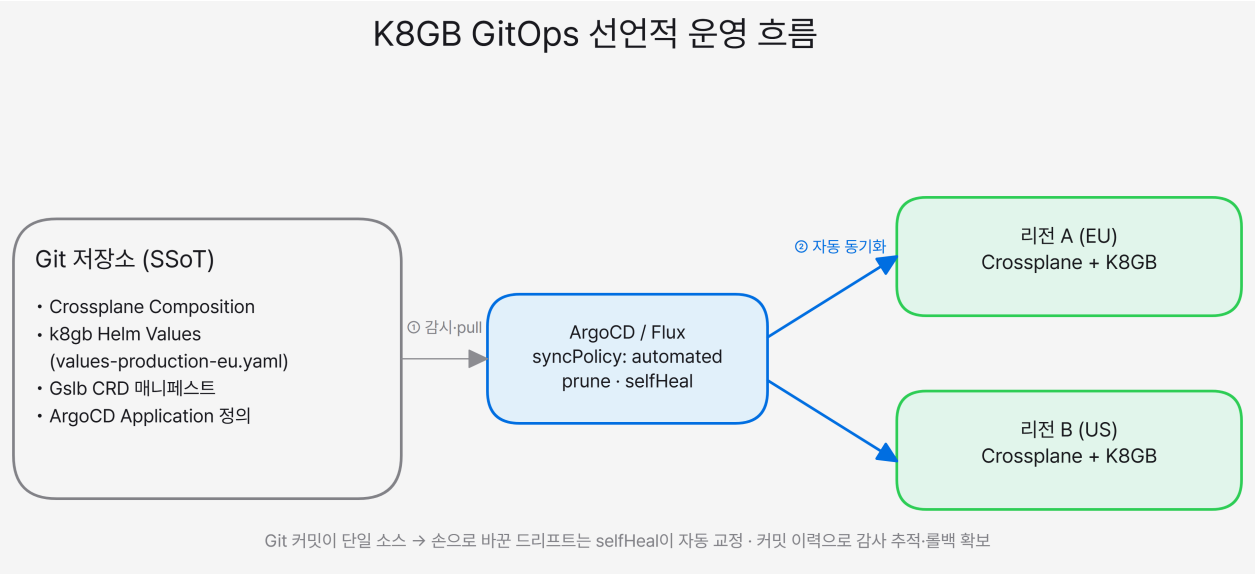

그림. Git 저장소(Crossplane Composition·Helm Values·Gslb CRD)를 단일 소스로 삼아 ArgoCD/Flux가 리전 A·B의 Crossplane+K8GB에 자동 동기화하는 GitOps 흐름입니다.

9.2.2 CoreDNS-GSLB·Istio 연계 (North-South / East-West)

K8GB는 클러스터로 트래픽을 들여보내는 데까지를 담당하고, 클러스터 안에서 서비스 사이를 오가는 트래픽은 서비스 메시가 맡습니다. 둘은 서로 다른 계층에서 동작하며 상호 보완합니다[S6]. GSLB는 DNS(L4) 계층에서 외부 클라이언트를 최적 클러스터로 보내는 North-South(외부→클러스터) 트래픽을, Istio나 Linkerd 같은 서비스 메시는 애플리케이션(L7) 계층에서 클러스터 내부와 클러스터 간 East-West 트래픽을 다룹니다. 라우팅 기준도 다릅니다. GSLB는 지역·가용성·가중치로 클러스터를 고르고, 서비스 메시는 헤더·경로·버전으로 세밀하게 나눕니다. 헬스체크는 GSLB가 DNS TTL 기반, 서비스 메시가 사이드카(Sidecar) 실시간 기반이며, 보안은 GSLB의 DNSSEC와 서비스 메시의 mTLS(mutual TLS, 양방향 상호 인증 암호화)로 갈립니다.

구분	GSLB (K8GB)	서비스 메시 (Istio/Linkerd)
동작 계층	DNS (L4)	애플리케이션 (L7)
트래픽 방향	North-South (외부→클러스터)	East-West (클러스터 내·간)
라우팅 기준	지역·가용성·가중치	헤더·경로·버전
헬스체크	DNS TTL 기반	사이드카 실시간 기반
보안	DNSSEC	mTLS

Istio와 결합하면 두 계층의 역할 분담이 분명해집니다. GSLB가 DNS 레벨에서 최적 클러스터를 고른 뒤, Istio가 그 클러스터 안에서 VirtualService로 버전·헤더 기반 L7 라우팅을 수행합니다

[S6]. 예컨대 GSLB가 EU 클러스터를 선택하면, Istio VirtualService가 `x-region: eu` 헤더를 보고 app-v2에 90%, app-v1에 10% 비율로 트래픽을 흘려보내는 식입니다. 특히 Istio의 Locality-Aware Load Balancing(지역 인지 로드밸런싱)은 리전 내 우선 라우팅으로 GSLB 결정을 보완하고, 클러스터 간 East-West Gateway 통신을 mTLS로 보호하며, 쿠버네티스 MCS(Multi-Cluster Services) API로 서비스 디스커버리를 표준화합니다 [S6]. Istio 멀티클러스터 토폴로지 중 각 클러스터에 독립 istiod를 두는 Multi-Primary 구성은 K8GB의 대등형 구조와 결이 맞아 특히 탄력적입니다.

경량 대안인 Linkerd도 GSLB와 협업합니다 [S6]. Linkerd는 통합된 신뢰 도메인(Unified Trust Domain)으로 클러스터 간 워크로드 ID를 검증하되 장애 도메인은 분리해 유지하며, 2.14 이상에서는 Flat Network 환경에서 Gateway를 우회한 Pod-to-Pod 직접 통신을 지원합니다. 이 조합에서 GSLB는 외부 트래픽의 글로벌 분배를, Linkerd는 클러스터 내부와 클러스터 간 서비스 통신의 보안·관측성을 담당합니다. 어느 서비스 메시를 쓰든 요점은 같습니다. K8GB는 문 앞까지 트래픽을 안내하는 North-South 도구이고, 문 안쪽의 흐름 제어는 East-West를 맡는 서비스 메시의 몫이며, 두 계층을 겹쳐야 멀티클러스터 Active-Active가 완성됩니다.

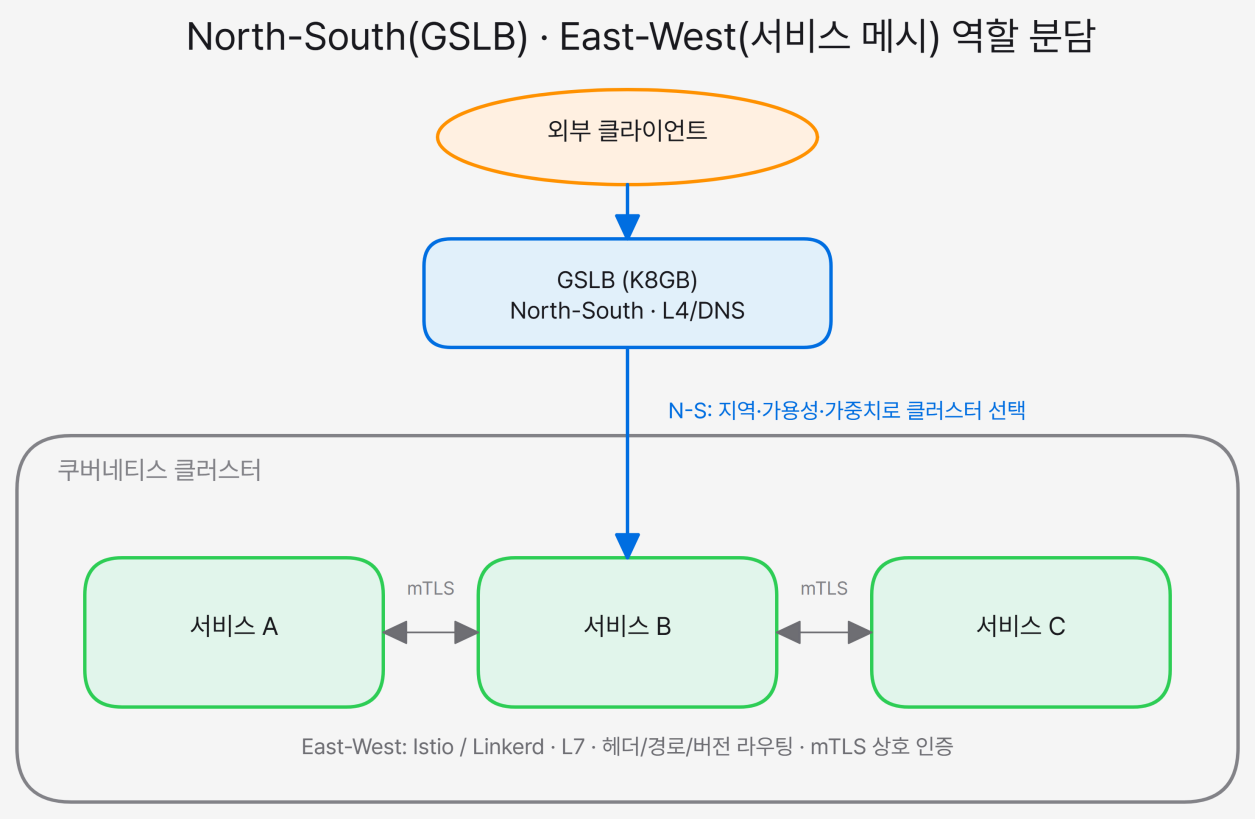

그림. North-South(GSLB, L4/DNS)와 East-West(Istio/Linkerd, L7/mTLS)의 역할 분담 — 외부 클라이언트는 GSLB로 클러스터를 배정받고, 클러스터 내부·간 서비스 통신은 서비스 메시가 관리합니다.

10장: 방식별 심층 비교와 결정 트리

앞선 세 장은 GSLB(Global Server Load Balancing, 글로벌 서버 부하 분산)를 구현하는 서로 다른 계층을 각각 다루었습니다. 7장은 하드웨어 어플라이언스, 8장은 클라우드 매니지드 서비스, 9장은 쿠버네티스 네이티브형 K8GB(Kubernetes Global Balancer)입니다. 세 방식은 같은 목적을 다른 비용 구조와 운영 모델로 달성하며, 어느 하나가 모든 조직에 우월하지 않습니다. 이 장은 새로운 주장을 더하지 않고 세 방식을 여러 평가 관점에서 한자리에 모아 정리한 뒤, 자사 조건을 좁혀 가는 결정 흐름을 제시합니다. 핵심은 단순합니다. 단일 정답을 찾는 것이 아니라 규제·쿠버네티스 표준화·멀티클라우드·클라우드 사업자(CSP, Cloud Service Provider) 종속이라는 조건을 차례로 물어 자사에 남는 선택지를 확인하는 것입니다 [S10].

10.1 방식별 비교표

10.1.1 비용·구현 난이도·유지보수 비교

세 방식을 가르는 첫 번째 관점은 비용 구조입니다. 비용은 절대 금액이 아니라 어떤 형태로 발생하는지, 규모가 커질 때 어떻게 변하는지로 봐야 실질적인 비교가 됩니다. 하드웨어 어플라이언스는 자본 지출(CapEx, Capital Expenditure) 중심입니다. 장비 구매비, 영구 라이선스, 연간 유지보수 계약이 초기에 크게 발생하며, F5 BIG-IP DNS가 최상위 비용대에, Kemp LoadMaster GEO가 최저 비용대에 위치합니다 [S5]. 클라우드 매니지드 서비스는 운영 지출(OpEx, Operating Expenditure) 중심입니다. 초기 비용이 사실상 없고 질의 수와 트래픽에 비례하는 종량 과금으로 시작하지만, 서비스 규모가 커지면 이 종량 비용이 누증합니다 [S5]. K8GB는 오픈소스로 라이선스 비용이 없는 대신, 비용이 사라지는 것이 아니라 쿠버네티스 클러스터 운영과 플랫폼 인력이라는 형태로 자리를 옮깁니다 [S6].

구현 난이도는 조직이 이미 보유한 역량에 따라 체감이 달라집니다. 하드웨어 어플라이언스 중 F5와 Citrix NetScaler ADC는 iRules 같은 커스터마이징과 초기 구성에 가파른 학습 곡선을 요구해 전문 인력이 필요하고, A10 Thunder ADC와 Kemp는 상대적으로 구성이 간편합니다 [S5]. 클라우드 매니지드 서비스는 콘솔과 API로 빠르게 배포할 수 있어 진입이 가장 낮은 편입니다 [S5]. K8GB는 중간에서 높은 수준으로, 쿠버네티스·Helm·GitOps에 대한 숙련과 멀티클러스터 이해를 전제합니다. 이 전제가 충족된 조직에는 자연스러운 확장이지만, 그렇지 않은 조직에는 오히려 장벽이 됩니다 [S6].

유지보수 부담도 세 방식이 뚜렷이 갈립니다. 하드웨어는 장비 수명주기, 펌웨어 관리, 용량 증설 같은 물리 인프라 운영 책임을 조직이 계속 떠안습니다 [S5]. 클라우드 매니지드는 벤더가 인프라를 운영하므로 이 부담이 가장 낮습니다 [S5]. K8GB는 선언적 관리로 상당 부분을 자동화할 수 있으나, 클러스터와 사용자 정의 리소스(CRD, Custom Resource Definition)의 운영 책임은 조직에 남습니다 [S6]. 결국 비용과 난이도는 따로 볼 수 없습니다. 라이선스가 무료여도 운영 인력이 없으면 K8GB의 총소유비용은 낮지 않고, 초기 비용이 없어도 트래픽이 크면 매니지드의 누적 OpEx가 하드웨어의 CapEx를 넘어설 수 있기 때문입니다 [S5][S6].

10.1.2 벤더 종속·멀티클라우드·GitOps·페일오버 속도

비용만으로 선택이 끝나지 않습니다. 전략적 관점이 결정을 뒤집기도 합니다. 아래 종합 비교표는 7~9장의 논의를 여덟 개 관점으로 수렴한 것입니다 [S5][S6].

평가 관점	하드웨어 어플라이언스 (F5·Citrix·A10·Kemp)	클라우드 매니지드 (Route 53·Azure TM·NS1·Cloudflare)	K8GB (쿠버네티스 네이티브)
비용 구조	높은 CapEx(장비·영구 라이선스·연 유지보수). F5 최상, Kemp 최저	낮은 초기비용, OpEx 종량(질의·트래픽). 규모 확대 시 누증	라이선스 0(오픈소스). 비용이 인력·클러스터 운영으로 이동
구현 난이도	높음(F5·Citrix), 중간(A10·Kemp). 전문 인력·학습곡선	낮음~중간. 콘솔·API로 빠른 배포	중간~높음. 쿠버네티스·Helm·GitOps 숙련 전제
유지보수·운영 부담	높음(장비 수명주기·펌웨어·용량 증설)	낮음(관리형, 벤더가 인프라 운영)	중간. 선언적 자동화 가능하나 클러스터·CRD 책임
벤더 종속성	높음(제품·iRules 종속)	CSP 네이티브는 높음 / Cloudflare·NS1은 중립	가장 낮음(CNCF 오픈소스, CoreDNS 표준)
멀티클라우드·하이브리드 적합성	중간(가상 어플라이언스로 확장)	Azure TM·Cloudflare·NS1 우수 / CSP 네이티브 제한적	우수(클러스터 위치와 무관하게 동일 방식)
확장·자동화(GitOps)	제한적(장비 용량·수동 구성 경향)	API·IaC 가능하나 상태는 콘솔 중심	최상(CRD 선언·ArgoCD/Flux·Crossplane, Git 단일 소스)
페일오버 속도	빠름(전용 장비·MEP), 단 DNS TTL 의존	Anycast형(GCP·Cloudflare) 빠름 / 순수 DNS형 TTL 의존	감지는 빠르나(Readiness Probe) 최종 전환은 DNS TTL 의존
적합 조직	대기업·통신사·금융, 규제·초고성능	클라우드 우선·글로벌 웹·빠른 도입	쿠버네티스 표준화·멀티클러스터·플랫폼 성숙 조직

벤더 종속성 관점에서 세 방식의 위치가 분명히 갈립니다. 하드웨어는 제품과 iRules 같은 벤더 고유 기능에 묶이고, Citrix의 MEP(Metrics Exchange Protocol, 메트릭 교환 프로토콜)는 타 벤더 장비와 상호 운용되지 않는 독점 프로토콜입니다 [S5]. CSP 네이티브 서비스는 해당 클라우드에 종속되지만, Cloudflare와 IBM NS1 Connect는 벤더 중립을 표방합니다 [S5]. K8GB는 CNCF 오픈소스이자 표준 CoreDNS 기반이라 종속성이 가장 낮습니다 [S6]. 멀티클라우드·하이브리드 적합성도 이와 맞물립니다. CSP 네이티브는 자기 클라우드 밖에서는 제한적인 반면, Azure Traffic Manager의 외부 엔드포인트 지원, Cloudflare·NS1의 벤더 중립성, K8GB의 클러스터 위치 무관 동작이 이질적 인프라를 아우르는 데 강합니다 [S5][S6].

확장과 자동화(GitOps) 관점에서는 K8GB가 두드러집니다. 모든 설정이 쿠버네티스 CRD로 선언되므로 ArgoCD·Flux를 통해 Git을 단일 소스로 삼고, selfHeal로 수동 변경에 따른 드리프트를 자동 복구합니다 [S6]. 하드웨어는 장비 용량과 수동 구성 경향으로 자동화가 제한적이고, 클라우드 매니지드는 코드형 인프라(IaC, Infrastructure as Code)가 가능하나 GSLB 상태 관리는 여전히 콘솔 중심인 경우가 많습니다 [S5][S6]. 마지막으로 페일오버 속도는 겉보기와 달리 세 방식이 결국 한 지점으로 수렴합니다. 전용 장비의 빠른 감지, Anycast형의 네트워크 레벨 전환, K8GB의 Readiness Probe 기반 즉시 감지 모두 마지막 전환은 DNS TTL(Time To Live, 캐시 유효시간)에 종속됩니다 [S5][S6]. 6장에서 다룬 낮은 TTL·다중 A레코드·클라이언트 재시도 보완이 어느 방식에서든 체감 전환 속도를 정하는 이유입니다.

10.2 의사결정 가이드

10.2.1 결정 트리 (규제 → K8s → 멀티클라우드 → CSP)

비교표가 방식의 특성을 펼쳐 보였다면, 결정 트리는 자사 조건으로 그 특성을 좁히는 도구입니다. 단일 정답이 없으므로, 아래 순서로 자문하며 남는 선택지를 확인하는 방식이 실질적인 결론에 이르는 경로입니다 [S10]. 각 분기의 근거는 앞 장의 논의에 이미 담겨 있습니다.

결정 트리. 규제 → 쿠버네티스 표준화 → 멀티클라우드 → 단일 CSP 순으로 좁혀 가는 4단계 자문 흐름. 각 분기에서 "예"이면 해당 계층으로 확정하고, "아니오"이면 다음 질문으로 내려갑니다. 아래 들여쓰기 텍스트가 그 흐름을 그대로 표현합니다.

- 질문 1. 규제(데이터 주권·망분리)로 인프라를 사내에 두어야 하는가?
  - 예 → 온프레미스 GSLB(하드웨어 어플라이언스)
    - 초고성능·대규모·기존 F5 자산 → F5 BIG-IP DNS
    - Citrix 가상 데스크톱(VDI) 환경 → NetScaler ADC
    - 비용 효율·간편 운영 → A10 Thunder / Kemp LoadMaster GEO
  - 아니오 → 질문 2로
- 질문 2. 워크로드가 쿠버네티스로 표준화되어 있고 멀티클러스터인가?
  - 예, 플랫폼·GitOps 역량이 충분 → K8GB (+ 서비스 메시 연계)
  - 아니오 또는 역량 불충분 → 질문 3으로
- 질문 3. 멀티클라우드 또는 벤더 중립이 중요한가?
  - 예 → Cloudflare / IBM NS1 Connect / Azure Traffic Manager(외부 엔드포인트)
  - 아니오(단일 CSP) → 해당 CSP 네이티브
    - AWS 중심 → Route 53
    - Azure 중심 → Traffic Manager / Front Door
    - GCP 중심 → Cloud DNS + Cloud LB (Anycast)

첫 번째 질문을 규제에 둔 이유는, 데이터 주권이나 망분리 요건이 있으면 다른 관점의 이점과 무관하게 인프라를 사내에 두어야 하기 때문입니다. 규제가 계층 선택을 강제하는 가장 강한 조건이므로 맨 앞에 놓입니다 [S10]. 이 관문을 통과하면 두 번째로 쿠버네티스 표준화 여부를 묻습니다. 워크로드가 이미 멀티클러스터 쿠버네티스로 표준화되어 있고 플랫폼·GitOps 역량이 뒷받침되면, K8GB는 라이선스 비용 없이 기존 배포 워크플로우에 자연스럽게 얹히는 선택입니다 [S6]. 역량이 부족하면 이 이점이 장벽으로 바뀌므로 다음 질문으로 내려갑니다. 세 번째는 멀티클라우드·벤더 중립의 중요도입니다. 여러 CSP를 동일한 수준으로 관리해야 한다면 CSP 네이티브 서비스는 구조적으로 제약이 있으므로, 벤더 중립 매니지드 계층이 답이 됩니다 [S5][S10]. 이 모든 조건에 해당하지 않는 단일 CSP 표준화 조직이라면, 해당 클라우드의 네이티브 서비스가 통합 편의와 비용 면에서 가장 합리적입니다 [S5]. 이 흐름의 미덕은 각 분기가 배타적 정답을 강요하지 않고, 자사 상황에 남는 선택지를 자연스럽게 드러낸다는 데 있습니다.

10.2.2 조직 규모·시나리오별 권고

결정 트리의 결과는 조직 유형으로 다시 확인할 수 있습니다. 같은 결론에 두 경로로 도달하면 선택의 확신이 높아지기 때문입니다. 아래 권고는 트리의 분기를 조직 규모·성격으로 재정리한 것이며, 여전히 "단일 정답은 없다"는 전제 위에 놓인 조건부 권고입니다 [S10].

조직 유형	권고 방식	근거
소규모·스타트업	클라우드 매니지드(Route 53·Cloudflare)	초기 비용 부담이 없고 콘솔·API로 빠르게 도입
대기업·금융·통신	온프레미스(F5·Citrix) 또는 하이브리드	규제·초고성능 요건, 기존 장비 자산 활용
멀티클라우드 조직	벤더 중립 매니지드 (Cloudflare·NS1·Azure TM)	특정 CSP 종속 회피, 이식성과 협상력 확보
쿠버네티스 표준화 조직	K8GB(+ 서비스 메시)	라이선스 비용 제거·GitOps 일관성, 단 운영 역량 전제

소규모·스타트업은 초기 자본 지출을 감당하기 어렵고 빠른 출시가 중요하므로, 종량 과금과 즉시 배포가 강점인 클라우드 매니지드가 자연스러운 출발점입니다 [S5]. 대기업·금융·통신은 전자금융감독규정 같은 재해복구 규제와 초고성능 요건을 안고 있고 이미 F5나 NetScaler 자산을 보유한 경우가 많아, 온프레미스 어플라이언스나 온프레미스와 클라우드를 결합한 하이브리드가 적합합니다 [S5][S10]. 멀티클라우드 조직은 특정 클라우드 사업자에 묶이는 위험을 피하고 이식성과 협상력을 지키는 것이 핵심 가치이므로, 벤더 중립을 표방하는 Cloudflare·NS1·Azure Traffic Manager가 우선 검토 대상입니다 [S5]. 쿠버네티스로 표준화된 조직은 K8GB로 라이선스 비용을 없애고 GitOps 일관성을 얻지만, 이 이점은 플랫폼 운영 역량이 뒷받침될 때만 실현됩니다 [S6].

이 권고들을 관통하는 결론은 백서 전체의 메시지와 같습니다. GSLB 방식 선택은 제품의 기능 목록을 비교하는 문제가 아니라, 규제·역량·인프라 표준이라는 자사 조건을 정직하게 확인하는 문제입니다. 어떤 방식을 고르든 페일오버 속도는 결국 DNS TTL과 클라이언트 계층 보완으로 수렴하고, Active-Active의 성패는 방식 선택 위 계층인 데이터 정합성 설계에서 갈립니다 [S6]

[S10]. 따라서 결정 트리는 종착점이 아니라 출발점입니다. 방식을 좁힌 뒤에는 앞선 장에서 다룬 복제 전략, 세션 관리, TTL 튜닝을 자사 서비스 등급에 맞춰 설계하는 작업이 이어집니다.

11장: 주요 실무 검토 항목 (RTO/RPO·TCO·규제)

앞선 장들은 GSLB로 Active-Active를 구성하는 개별 기술 요소를 다루었습니다. 트래픽 분산 계층, 헬스체크와 페일오버, 데이터 복제와 정합성, 세션 전략, 세 가지 제품 계열의 특성이 그것입니다. 그러나 도입을 앞둔 엔지니어가 실제로 답해야 할 질문은 개별 기술의 원리가 아니라 "우리 서비스에 무엇을 어느 값으로 설정하고, 얼마의 비용과 어떤 조직 역량으로 감당할 것인가"입니다. 이 장은 도입 전 반드시 점검해야 할 실무 항목을 목표 정의·운영 파라미터·비용·규제·조직 역량의 순서로 정리합니다.

핵심 메시지는 분명합니다. Active-Active의 성패는 GSLB 제품 선택이 아니라 목표 정의, 파라미터 튜닝, 조직 역량 매칭에서 갈립니다. 아무리 고성능 어플라이언스를 도입해도 RTO(Recovery Time Objective, 복구 목표 시간)와 RPO(Recovery Point Objective, 복구 목표 시점)를 정의하지 않으면 페일오버 품질을 검증할 근거가 없고, 관측 지표가 없으면 장애 전환이 실제로 일어났는지조차 알 수 없습니다. 반대로 목표와 지표가 명확하면 오픈소스 K8GB로도 목표 수준의 가용성을 달성할 수 있습니다 [S6]. 이 장의 각 절은 그대로 체크리스트로 옮겨 쓸 수 있도록 구성했습니다.

11.1 설계·운영 검토

11.1.1 RTO/RPO 정의와 관측성

모든 Active-Active 설계는 두 개의 숫자에서 출발합니다. RTO는 장애 발생 후 서비스가 복구되기까지 허용하는 최대 시간이고, RPO는 장애 시점 기준으로 허용하는 최대 데이터 손실 구간입니다. 이 두 값은 서비스 등급마다 다르게 잡아야 합니다. 결제·인증처럼 중단이 곧바로 매출·신뢰 손실로 이어지는 서비스는 RTO 수초~수십초, RPO 0에 가깝게 설정하며, 이는 동기 복제 기반 Active-Active를 정당화하는 근거가 됩니다 [S4]. 반면 사내 배치·리포팅처럼 수 분의 중단과 수 분의 데이터 손실이 허용되는 서비스는 굳이 Active-Active의 복잡도를 떠안을 필요가 없습니다. 목표 수치 없이 "최대한 빠르게"라는 정성적 요구만으로 설계하면 헬스체크 간격도 TTL도 근거 없이 정해지고, 결과적으로 과잉 설계나 과소 설계로 귀결됩니다.

목표를 정했다면 다음은 그 목표가 실제로 지켜지는지 관측하는 체계입니다. 목표와 관측이 함께 있어야 페일오버 품질을 검증할 수 있고, 관측이 없는 DR은 훈련되지 않은 DR과 마찬가지로 신뢰할 수 없습니다. Active-Active 환경에서 최소한으로 확보해야 할 관측 대상은 다음과 같습니다.

- **GSLB 사이트 상태**: 각 사이트의 Healthy/Down 판정, 헬스체크 연속 실패·성공 횟수, 현재 트래픽 분배 비율 [S4]
- **DNS 전파 현황**: 응답 레코드의 TTL, 다중 A레코드 구성, 사이트별 질의 응답 분포 [S4]
- **Failover 이벤트 알림**: 사이트 Down 마킹, DNS 응답에서 IP 제거, 복구 재편입 시점의 즉시 알림 [S4]
- **데이터 복제 지연**: 사이트 간 복제 랙(lag), RPO 목표 대비 실제 지연 구간 [S4]

그림. Active-Active 환경의 관측 대상 지표 — GSLB 사이트 상태, DNS 전파 현황, Failover 이벤트, 복제 지연을 한 화면에서 조망하는 대시보드 구성.

제품 계열별로 관측 방식은 다릅니다. 하드웨어 어플라이언스와 클라우드 매니지드는 각기 자체 대시보드·로그·SNMP 트랩을 제공하지만 벤더별로 형식이 다릅니다. 쿠버네티스 네이티브형 K8GB는 CoreDNS를 임베드하고 컨트롤러가 각 클러스터의 Readiness Probe 상태를 GSLB 라우팅 결정에 직접 반영하므로, 이 상태가 쿠버네티스 표준 메트릭 경로로 노출됩니다 [S6]. 즉 K8GB는 상태 지표를 Prometheus(오픈소스 메트릭 수집·시계열 데이터베이스)로 수집하고 Grafana로 시각화하는 클라우드 네이티브 관측 스택에 자연스럽게 편입됩니다. 별도 모니터링 인프라를 새로 구축하지 않고 클러스터가 이미 갖춘 관측 파이프라인을 재사용한다는 점이 K8GB 관측성의 실무적 이점입니다 [S6]. 다만 이는 조직이 Prometheus·Grafana 운영 역량을 이미 보유하고 있다는 전제 위에서 성립하며, 그 역량이 없다면 관측성 확보 자체가 새로운 학습 부담이 됩니다.

11.1.2 헬스체크·TTL·Circuit Breaker 실무값

목표와 관측 체계를 갖췄다면 이제 앞선 장에서 다룬 운영 파라미터를 한곳에 모아 그대로 참고할 수 있게 정리합니다. 여기 제시하는 값은 4장(헬스체크·페일오버)과 6장(TTL·클라이언트 재시도)에서 근거와 함께 설명한 수치를 재인용한 것이며, 중복 서술을 피하기 위해 결론값과 그 의미만 요약합니다.

계층	권장 실무값	목표	재인용
헬스체크	애플리케이션 레벨, 10초 간격, 3회 연속 실패로 Down 판정	30초 이내 장애 감지	4장 [S4]

계층	권장 실무값	목표	재인용
복구 판정	연속 성공 3~5회로 재편입(장애 판정보다 크게)	플래핑 방지	4장 [S4]
DNS TTL	60초, 계획 변경 시 Migration TTL 패턴 적용	1분 이내 DNS 갱신	6장 [S4]
GSLB 응답	정상 사이트 복수 IP를 담은 다중 A레코드	클라이언트 측 즉시 대체	6장 [S4]
클라이언트 재시도	연결 타임아웃 3초, 다음 IP 재시도(Happy Eyeballs)	DNS 전파 대기 없는 전환	6장 [S4]
Circuit Breaker	연속 실패 시 해당 엔드포인트 차단 후 재시도	불필요한 재시도 차단	6장 [S4]

이 값들의 상호 관계를 이해해야 개별 항목을 자사 환경에 맞게 조정할 수 있습니다. 헬스체크는 간격과 판정 횟수의 곱이 곧 감지 시간입니다. 10초 간격에 3회 연속 실패를 요구하면 최악의 경우 약 30초 만에 장애가 감지됩니다 [S4]. 간격을 줄이거나 판정 횟수를 낮추면 감지는 빨라지지만 순간적 네트워크 흔들림을 장애로 오판하는 위험이 커집니다. 반대로 값을 키우면 오탐은 줄지만 감지가 늦어집니다. 애플리케이션 레벨 점검을 권장하는 이유는 ICMP나 TCP 같은 얕은 점검이 서버는 살아 있으나 DB 연결이 끊긴 상태를 잡지 못하기 때문이며, 이는 실제 서비스 가용성을 판단하는 유일한 방법입니다 [S4].

TTL은 감지된 장애를 클라이언트에 전파하는 속도를 결정합니다. 60초는 페일오버 속도와 DNS 질의 부하 사이의 일반적 균형점입니다 [S4]. TTL을 무작정 낮추면 질의 부하가 비례해 증가하고, 일부 ISP 리졸버는 300초 미만 TTL을 무시하고 자체 최소값을 강제하기도 합니다 [S4]. 그래서 평상시에는 긴 TTL을 유지하다 계획된 전환 직전에만 짧게 낮추는 Migration TTL 패턴이 부하와 속도를 동시에 관리하는 실무 해법이 됩니다 [S4].

가장 중요한 원칙은 DNS가 페일오버를 보조할 뿐 페일오버 자체가 되어서는 안 된다는 것입니다 [S4]. 실질적인 즉시 전환은 클라이언트 계층에서 일어납니다. GSLB가 정상 사이트 복수 IP를 다중 A레코드로 응답하면, 클라이언트는 첫 IP 연결을 3초 타임아웃 안에 포기하고 다음 IP로 재시도하여 TTL 만료를 기다리지 않고도 정상 사이트에 도달합니다 [S4]. 여기에 앱 레벨 Circuit Breaker를 더하면 연속 실패한 엔드포인트를 일정 시간 차단해 불필요한 재시도를 막고 대체 경로로 빠르게 넘어갑니다 [S4]. 종합하면 헬스체크(감지)·TTL(전파)·클라이언트 재시도(실제 전환)의 세 계층이 각자 역할을 나눠 총 페일오버 시간을 압축합니다.

11.2 비용·규제·조직 역량

11.2.1 TCO 모델 (CapEx vs OpEx vs 오픈소스)

비용은 도입 시점의 초기 금액이 아니라 3~5년 총소유비용(TCO, Total Cost of Ownership) 관점에서 비교해야 실질적인 판단이 가능합니다. 세 가지 제품 계열은 비용이 발생하는 구조 자체가 다르므로, 절대 금액이 아니라 비용 구조와 규모별 상대 비율로 이해하는 것이 옳습니다. 다음 표는 7~9장에서 다룬 각 계열의 비용 성격을 총소유비용 관점으로 수렴한 것입니다.

계열	비용 구조	주요 비용 동인	규모 확대 시
하드웨어 어플라이언스	CapEx 중심	하드웨어, 영구 라이선스, 연 유지보수, 전문 인력	용량 증설 시 추가 장비 CapEx
클라우드 매니지드	OpEx 중심	질의·트래픽 종량 과금	규모에 비례해 OpEx 누증
쿠버네티스 네이티브 (K8GB)	라이선스 0	플랫폼 인력, 클러스터 운영 비용	인력·클러스터 역량에 종속

하드웨어 어플라이언스는 초기 자본 지출(CapEx, Capital Expenditure)이 큽니다. 장비 구매, 영구 라이선스, 연간 유지보수 계약이 고정 비용으로 발생하며, F5·Citrix 계열은 라이선스와 지원 비용이 업계 최고 수준입니다 [S5]. 여기에 잘 드러나지 않지만 결정적인 항목이 전문 인력 비용입니다. iRules 같은 커스터마이징에는 학습 곡선이 가파른 전문 인력이 필요하고, 총소유비용에서 이 인력 비용의 비중이 상당합니다 [S5]. 대신 성능과 규제 대응이 강하므로 초고성능·규제 환경에서는 이 CapEx 구조가 정당화됩니다.

클라우드 매니지드는 운영 지출(OpEx, Operational Expenditure) 중심입니다. AWS Route 53, Azure Traffic Manager, Cloudflare 등은 질의 수와 트래픽 볼륨에 따른 종량 과금이라 초기 비용 부담이 없습니다 [S5]. 소규모로 시작할 때 진입 장벽이 가장 낮은 선택지입니다. 그러나 트래픽이 커지면 종량 비용이 비례해 누증하고, 어느 규모를 넘어서면 온프레미스의 CapEx를 역전할 수 있습니다 [S5]. 따라서 현재 규모만이 아니라 3~5년 성장 곡선을 함께 넣고 계산해야 합니다.

K8GB는 라이선스 비용이 0입니다. 오픈소스이므로 소프트웨어 비용 자체가 없고, 별도 관리 클러스터나 어플라이언스도 요구하지 않습니다 [S6]. 다만 비용이 사라진 것이 아니라 인력과 클러스터 운영 역량으로 이동한 것입니다. 쿠버네티스·GitOps·서비스 메시를 감당할 플랫폼 팀이 전제되며, 그 역량이 없으면 라이선스 절감분보다 학습·운영 비용이 더 클 수 있습니다 [S6]. 결국 TCO 비교의 결론은 단일 정답이 아니라 조직의 규모와 보유 역량에 따라 최소 비용점이 달라진다는 것입니다.

11.2.2 규제·컴플라이언스와 DR 훈련

비용 다음으로 검토할 것은 규제입니다. 규제는 다른 요소와 달리 선택지를 좁히는 것을 넘어 특정 계층을 강제할 수 있습니다. 금융권이 대표적입니다. 전자금융감독규정·전자금융거래법에 따라 RTO 2시간 이내의 재해복구 체계가 의무화되어 있고, 고객 금융 데이터의 국내 저장 의무와 국외 이전 시 사전 승인 요건이 데이터 주권(data sovereignty, 데이터가 특정 국가 관할에 종속되는 원칙)으로 작용합니다 [S10]. 이런 요건은 국내 IDC 한정 구축을 요구하므로, 데이터를 국외 리전에 둘 수 있는 퍼블릭 클라우드 매니지드보다 온프레미스 어플라이언스나 국내 인프라 기반 구성이 선택을 강제받습니다 [S10]. 여기에 모든 트래픽 라우팅 변경 이력을 기록·보존

해 규제 기관 감사에 대응해야 하는 감사 추적성 요건이 더해집니다 [S10]. 망분리 규제가 적용되는 환경이라면 GSLB의 헬스체크 경로, 사이트 간 메트릭 교환, 데이터 복제 경로가 분리된 망 정책 안에서 성립하는지 사전에 검증해야 합니다. 도입 전 점검할 규제 항목은 다음과 같습니다.

- 데이터 저장 위치 제한(데이터 주권) 확인
- 규제 기관 보고 요건 확인(DR 훈련 결과, 장애 보고)
- 감사 로깅 요구사항 확인(라우팅 변경 이력 보존)
- SLA 요구사항 정의(RTO, RPO, 가용성 목표) [S10]

규제 요건을 충족하는 구성을 갖췄더라도, 그 DR이 실제로 동작하는지는 훈련으로만 검증됩니다. 훈련 없는 DR은 검증되지 않은 DR입니다. 실무에서는 다음 절차를 정기적으로 반복해야 합니다.

- **정기 페일오버 훈련**: 계획된 장애 전환을 분기 또는 반기 주기로 실시하고 Failover·Failback 절차서의 실효성을 검증합니다. 금융권은 연 1회 이상 DR 훈련이 규제 요건이며 결과를 감독기관에 보고합니다 [S10].
- **Migration TTL 롤백**: 전환 직전 짧게 낮춘 TTL을 안정화 후 원래 값으로 복원하는 절차를 롤백 계획에 포함해, 훈련 종료 후 평상시 부하 수준으로 되돌립니다 [S4].
- **GitOps selfHeal 드리프트 복구**: K8GB 환경이라면 ArgoCD·Flux의 selfHeal이 Git 선언 상태와 클러스터 실제 상태의 드리프트를 자동 복구하므로, 훈련 중 수동 변경이 남더라도 구성이 원래 선언으로 수렴합니다 [S6]. 이는 구성의 재현성과 감사 추적을 함께 확보하는 방식입니다.

여기서 Active-Active 구성의 근본적 이점이 드러납니다. Active-Passive는 평상시 유휴 상태인 Standby 사이트가 실제 장애 시 정상 기동하는지를 별도 DR 훈련으로 사전 검증해야 하며, 이 검증 부담과 불확실성이 상존합니다 [S4]. 반면 Active-Active는 모든 사이트가 상시 실 트래픽을 처리하므로 장애 복구 능력이 별도 훈련 없이도 지속적으로 검증됩니다 [S10]. 다시 말해 상시 운영 중인 Active-Active가 곧 상시 검증된 DR입니다. 이 특성은 규제 대응과 운영 신뢰도 양쪽에서 Active-Active를 정당화하는 가장 강력한 근거입니다. 물론 이 이점은 앞선 장에서 다룬 데이터 정합성 설계와 이 장에서 정리한 파라미터 튜닝, 그리고 조직의 운영 역량이 함께 갖춰졌을 때 비로소 실현됩니다. 제품이 아니라 목표 정의·튜닝·역량 매칭이 성패를 가른다는 이 장의 결론은 여기서 다시 확인됩니다.

12장: 결론 및 도입 로드맵

앞의 열한 개 장은 GSLB로 Active-Active를 구현하는 과정을 트래픽 분산 계층에서 데이터 계층, 제품 선택, 실무 검토 항목까지 층층이 짚었습니다. 본 장은 그 논의를 새로 확장하지 않고 하나의 결론으로 수렴합니다. 핵심은 단순합니다. GSLB 제품을 무엇으로 고르느냐가 아니라, 그 위와 아래 계층을 어떻게 설계하고 튜닝하며 어떤 조직 역량으로 운영하느냐가 Active-Active의 성패를 정합니다. 이 결론을 먼저 세 가지 성공 요인으로 정리하고, 조건별 방식 선택을 한눈에 압축한 뒤, 검증을 쌓아 가며 확장하는 도입 로드맵과 이어질 참조 자산을 제시합니다.

12.1 핵심 결론

12.1.1 성패를 가르는 세 가지 — 데이터 정합성·튜닝·조직 역량

Active-Active 구성의 성패는 세 가지 축에서 갈립니다. 첫째는 데이터 정합성 설계입니다. 1장에서 밝힌 대로 GSLB는 트래픽만 분산하며 데이터 계층의 일관성은 책임지지 않습니다. 5장이 이 백서의 핵심 난제로 데이터 복제와 정합성을 다룬 이유가 여기에 있습니다. 양쪽 사이트가 동시에 쓰기를 받는 순간 쓰기 충돌과 Split-Brain 위험이 상수로 등장하며, 이는 GSLB 제품을 아무리 잘 골라도 사라지지 않습니다. 동기·비동기·반동기 복제 중 무엇을 택할지, 데이터 유형별로 Last-Write-Wins·CRDT·분산 합의 중 무엇으로 충돌을 해소할지, Quorum과 Witness로 분단 상황을 어떻게 방어할지를 서비스 특성에 맞춰 사전에 결정해야 합니다. 사용자 인증과 금융 트랜잭션은 강한 일관성과 동기 복제로, 상품 카탈로그나 장바구니는 최종 일관성과 자동 병합으로 나누어 설계하는 결정 매트릭스가 그 출발점입니다 [S4]. 이 설계를 건너뛴 Active-Active는 평상시에는 잘 도는 것처럼 보이다가 네트워크 분단이나 페일오버 순간에 데이터 불일치로 무너집니다.

둘째는 TTL과 페일오버 튜닝입니다. 4장과 6장이 반복해 강조했듯, 총 페일오버 시간은 헬스체크 파라미터만으로 정해지지 않고 DNS TTL 캐싱·브라우저 캐싱·재귀 리졸버 캐싱이 층층이 얹혀 결정됩니다. 헬스체크는 애플리케이션 레벨까지 내려가야 실제 장애를 잡고, 장애 판정 횟수와 복구 판정 횟수를 비대칭으로 두어야 플래핑을 막습니다. TTL은 무작정 낮추는 대신 평상시 긴 TTL에서 전환 직전 짧게 내리는 방식으로 질의 부하와 전환 속도의 균형을 잡고, Anycast와 다중 A레코드, 클라이언트 재시도로 DNS 단독 의존을 보완합니다. DNS는 페일오버를 보조하되 페일오버 자체가 되어서는 안 된다는 원칙이 이 축의 요체입니다.

셋째는 조직 역량 매칭입니다. 7~9장이 어플라이언스·클라우드 매니지드·K8GB를 나란히 비교한 것은 우열을 가리기 위해서가 아니라, 각 방식이 전제하는 운영 역량이 다르기 때문입니다. 어플라이언스는 iRules 같은 전문 인력을, K8GB는 쿠버네티스와 GitOps 숙련을, 클라우드 매니지드는 종량 비용 관리 감각을 요구합니다 [S10]. 감당할 수 없는 역량을 전제하는 방식은 제품이 아무리 우수해도 운영 단계에서 실패합니다. 세 축을 관통하는 결론은 명확합니다. 초점을 제품 선택에서 설계와 역량으로 옮겨야 합니다. 그리고 1.3절에서 밝힌 대로 이 세 축은 클라우드 네이티브 스택 위에서 서로 다른 계층이 나눠 맡습니다. 상태 외부화가 데이터 정합성 설계의 전제를 만들고, 선언적 GitOps가 페일오버 구성을 코드로 재현 가능하게 하며, 쿠버네티스와 서비스

메시 운영 역량이 곧 조직 역량의 실체가 됩니다. Active-Active DR을 클라우드 네이티브 관점에서 설계할 때 이 세 축이 하나의 스택으로 수렴합니다 [S6].

12.1.2 방식 선택 요약

바쁜 의사결정자를 위해 10장의 결정 트리를 한 문단으로 압축하면 다음과 같습니다. 자문의 순서를 규제, 쿠버네티스 표준화, 멀티클라우드, 단일 CSP의 네 갈래로 좁혀 가면 자사에 맞는 계층이 드러납니다. 이 요약은 새로운 주장이 아니라 앞 장 결론의 재확인입니다.

가장 먼저 규제를 봅니다. 데이터 주권이나 망분리 요건이 데이터의 물리적 저장 위치를 국내 혹은 사내로 제한한다면, 온프레미스 어플라이언스나 하이브리드 구성이 사실상 강제됩니다. F5 BIG-IP DNS나 Citrix NetScaler가 대형 금융·통신 환경에서 여전히 선택되는 이유는 초고성능과 함께 이 규제 적합성에 있습니다 [S5]. 다음으로 쿠버네티스 표준화 여부를 봅니다. 조직이 이미 멀티클러스터를 쿠버네티스로 운영하고 플랫폼 역량을 갖췄다면, 별도 어플라이언스나 관리 클러스터 없이 CoreDNS 임베드와 컨트롤러로 동작하고 Readiness Probe를 헬스체크로 그대로 활용하는 K8GB가 라이선스 비용 없이 GitOps 일관성까지 제공합니다 [S6]. 다만 이는 쿠버네티스와 선언적 운영 숙련을 전제하므로, 미숙한 조직에는 오히려 진입 장벽이 됩니다.

쿠버네티스 전면화가 아니라면 멀티클라우드 여부로 넘어갑니다. 특정 CSP에 묶이지 않고 여러 클라우드와 온프레미스를 아우르려는 조직에는 Cloudflare나 IBM NS1 Connect, 외부 엔드포인트를 지원하는 Azure Traffic Manager 같은 벤더 중립 매니지드가 협상력과 이식성이라는 실익을 줍니다 [S5]. 마지막으로 단일 CSP에 표준화된 조직이라면 AWS Route 53나 GCP Cloud DNS 같은 네이티브 서비스가 통합 편의와 낮은 초기 비용으로 가장 합리적입니다. GCP처럼 Anycast형이면 순수 DNS형보다 전환이 빠르다는 3장의 특성도 함께 고려합니다 [S6]. 이 네 분기의 결론은 10장과 동일합니다. 단일 정답은 없으며, 규제·역량·멀티클라우드·비용이라는 조건을 좁혀 가는 과정 자체가 답입니다. 각 방식의 페일오버 속도가 결국 DNS TTL로 수렴한다는 점도 잊지 말아야 합니다.

12.2 도입 로드맵

12.2.1 단계적 도입 (파일럿 → 확장)

Active-Active 전환은 한 번에 전체 서비스를 바꾸는 방식이 아니라, 검증을 쌓아 가며 범위를 넓히는 방식이 위험을 줄입니다. 여기서 단계는 일자로 나눈 실행 일정이 아니라, 무엇을 검증했을 때 다음으로 넘어갈 수 있는가라는 검증 관점의 관문입니다.

첫 단계는 비핵심 서비스를 대상으로 한 파일럿입니다. 장애가 나도 매출과 신뢰에 직접 타격이 크지 않은 서비스를 골라 GSLB와 Active-Active 구성을 실제로 세워 봅니다. 이 단계의 목적은 성능이 아니라 학습입니다. DNS 위임과 헬스체크 경로, 방화벽 규칙, 복제 경로가 실제 환경에서 기대대로 동작하는지를 확인하고, 앞 장에서 정의한 파라미터가 자사 네트워크에서 어떤 값으로 안정되는지를 관찰합니다.

두 번째 관문은 헬스체크·TTL·복제 검증입니다. 애플리케이션 레벨 헬스체크가 부분 장애까지 잡아내는지, TTL을 낮췄을 때 질의 부하가 감당 범위에 있는지, 복제 지연이 목표 RPO를 지키는지를 실측으로 확인합니다. 이 검증이 없으면 페일오버 품질을 판단할 근거 자체가 없으므로, 관측

성 체계를 함께 세워 GSLB 상태·DNS 전파 현황·Failover 이벤트를 지표로 남깁니다. 세 번째 관문은 페일오버 훈련입니다. 정기적으로 사이트를 의도적으로 내려 트래픽이 정상 사이트로 전환되고 데이터가 일관되게 유지되는지, 복구 후 재편입이 플래핑 없이 이루어지는지를 반복 검증합니다. 훈련 없는 DR은 검증되지 않은 DR이며, 상시 운영 중인 Active-Active가 곧 상시 검증된 DR이라는 이점이 이 단계에서 실현됩니다 [S10].

마지막은 핵심 서비스 확장입니다. 앞 관문을 통과해 신뢰가 쌓인 구성을 결제·인증 같은 미션 크리티컬 서비스로 넓힙니다. 이 확장은 DR 성숙도가 Cold에서 Warm, Hot을 거쳐 Active-Active로 진화하는 흐름과 같으며, 각 단계에서 용량 부족·환경 드리프트·데이터 갭 같은 실패 요인을 미리 걸러 냅니다 [S3]. 앞 관문을 건너뛰고 곧장 핵심 서비스에 Active-Active를 적용하는 것이 가장 큰 위험이며, 검증을 관문으로 삼는 이 순서가 그 위험을 낮춥니다.

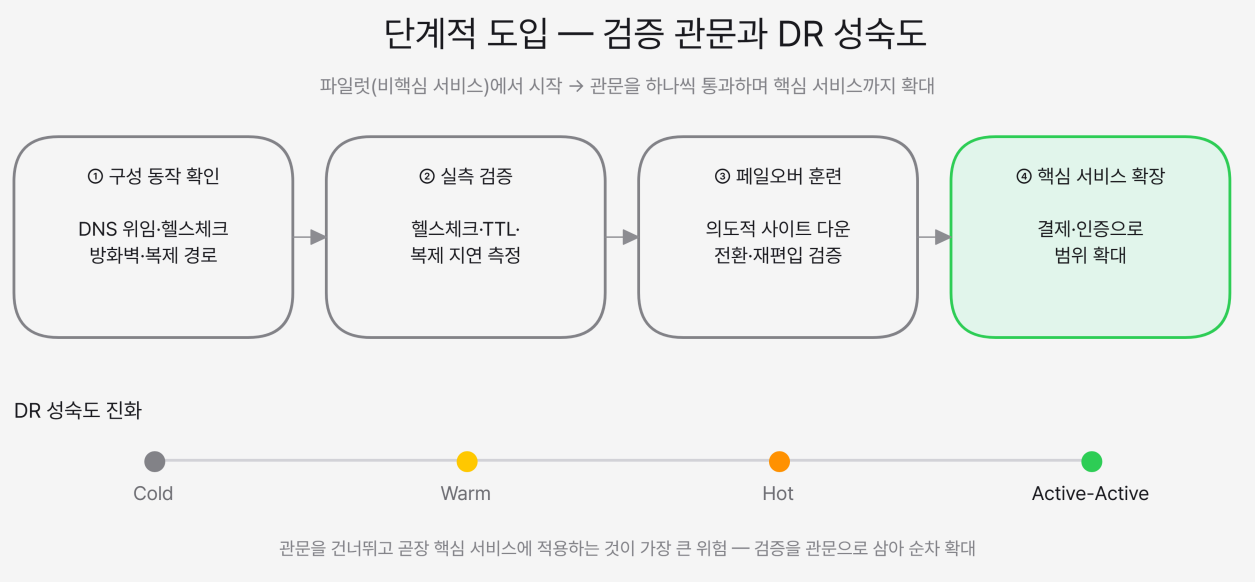

그림. 파일럿에서 핵심 서비스 확장까지, 각 단계를 잇는 네 개의 검증 관문(구성 동작 → 헬스체크·TTL·복제 실측 → 페일오버 훈련 → 확장)과 DR 성숙도(Cold→Warm→Hot→Active-Active)의 대응.

12.2.2 참조 자산과 다음 단계

본 백서는 GSLB로 Active-Active를 구현하는 의사결정에 집중했으며, 그 토대가 되는 개념과 다이어그램은 이전 자산을 계승했습니다. GSLB의 정의, DNS 기반 라우팅 흐름, 라우팅 알고리즘, 헬스체크 메커니즘의 기본기는 이전 "GSLB 이해하기" 백서에서 이미 상세히 다루었으므로, 개념을 처음부터 다시 세우려는 독자는 그 문서를 먼저 참조하는 편이 효율적입니다 [S1]. 본 백서 곳곳에서 재활용을 검토한 다이어그램들 — DNS 질의 흐름, 헬스체크 계층, 자동 페일오버 시퀀스, 데이터 동기화, TTL 페일오버, K8GB 아키텍처 — 역시 그 계보에서 이어진 자산입니다.

다음 단계로 이어 갈 경로는 독자의 위치에 따라 갈립니다. 방식 선택을 이미 좁힌 독자라면, 선택한 계층의 상세 구현 가이드로 넘어가는 것이 자연스럽습니다. 쿠버네티스 네이티브를 택했다면 K8GB의 Gslb CRD 선언과 failover·roundRobin·weightRoundRobin·geoip 전략 설정, ArgoCD·Flux를 통한 GitOps 운영, CoreDNS-GSLB와 Istio를 잇는 North-South/East-West 역할

분담이 다음 학습 대상입니다 [S6]. 클라우드 매니지드나 어플라이언스를 택했다면 각 제품의 라우팅 정책과 헬스체크 튜닝, TTL 전략을 자사 환경에 맞춰 구체화하는 작업이 뒤따릅니다.

어느 경로든 공통으로 남는 다음 단계는 앞 장의 실무 검토 항목을 체크리스트로 옮겨 실제 프로젝트에 적용하는 일입니다. 서비스 등급별 RTO/RPO 목표 정의, 헬스체크·TTL·Circuit Breaker의 실무값 확정, TCO를 3~5년 총소유비용 관점에서 비교하는 작업, 규제 요건과 DR 훈련 주기를 문서로 고정하는 작업이 여기에 포함됩니다. 이 백서가 제시한 결론 — 제품이 아니라 데이터 정합성 설계와 튜닝, 조직 역량이 성패를 가른다 — 은 그 자체로 끝이 아니라, 자사 서비스의 첫 파일럿 대상을 정하고 첫 검증 관문을 세우는 실무의 출발점입니다. 세 성공 요인을 점검 축으로 삼아 자사 도입의 첫 단계를 정하는 것이 이 백서를 읽은 독자가 이어 갈 가장 구체적인 행동입니다.

Appendix

References — 내부 사전조사 자료 (Primary Research)

> *본 백서는 아래 사내 리서치 자료(S1–S10)를 1차 근거로 삼았습니다. 각 사내 자료가 인용한 외부 공개 자료(벤더 공식 문서·IETF 표준·오픈소스 프로젝트)는 이어지는 「외부 참조 자료」에 정리했습니다. 모두 OPENMARU 사내 리서치 자료(2026)입니다.*

- [S1] GSLB 이해하기 백서 1-3장 — 기본 개념·헬스체크·라우팅.
- [S2] GSLB 이해하기 백서 4-5장 — 아키텍처·확장 토폴로지.
- [S3] GSLB 이해하기 백서 6-7장 — 운영·구축 사례.
- [S4] GSLB와 재해복구(DR) 심층 리서치 — Active-Active/Passive·데이터 복제.
- [S5] GSLB 제품 비교 분석 — 하드웨어·클라우드·K8GB 제품군.
- [S6] GSLB와 클라우드 네이티브 기술 연구 — K8GB·GitOps·Service Mesh.
- [S7] GSLB 아키텍처 패턴 리서치.
- [S8] GSLB 관련 표준 및 프로토콜 리서치 — DNS·EDNS·RFC.
- [S9] GSLB 구축 사례 및 레퍼런스 리서치.
- [S10] Active-Active vs Active-Passive DR — 멀티 클라우드 시대의 GSLB 전략.

External References — 외부 참조 자료

> *아래는 위 사내 리서치 자료가 근거로 인용한 외부 공개 자료입니다. 각 항목 끝의 `(← S#)`는 그 외부 자료를 참조한 사내 문서를 가리킵니다. 사내 자료에 있던 `example.com` 형태의 예시 URL은 실제 출처가 아니므로 제외했습니다. 접속일은 2026-07-13 기준입니다.*

표준·프로토콜 (IETF RFC)

- [E1] IETF. *RFC 6891 — Extension Mechanisms for DNS (EDNS(0))*. https://datatracker.ietf.org/doc/html/rfc6891 (← S8)
- [E2] IETF. *RFC 7871 — Client Subnet in DNS Queries (EDNS Client Subnet)*. https://datatracker.ietf.org/doc/html/rfc7871 (← S8)
- [E3] IETF. *RFC 8305 — Happy Eyeballs Version 2: Better Connectivity Using Concurrency*. https://datatracker.ietf.org/doc/html/rfc8305 (← S4, S8, S10)

DNS·TTL 운영 가이던스

- [E4] APNIC Blog (2019). *Stop using ridiculously low DNS TTLs*. https://blog.apnic.net/2019/11/12/stop-using-ridiculously-low-dns-ttls/ (← S4)

GSLB 개념·해설 (벤더 중립)

- [E5] F5. *Global Server Load Balancing* (Glossary). https://www.f5.com/glossary/global-server-load-balancing (← S4, S9)
- [E6] IBM. *Global Server Load Balancing*. https://www.ibm.com/think/topics/global-server-load-balancing (← S4)
- [E7] Cloudflare. *What is GSLB?* (Learning Center). https://www.cloudflare.com/learning/cdn/glossary/global-server-load-balancing-gslb/ (← S9)
- [E8] Splunk. *Global Server Load Balancing (GSLB)*. https://www.splunk.com/en_us/blog/learn/global-server-load-balancing-gslb.html (← S4, S9)
- [E9] Loadbalancer.org. *The Ultimate Guide to GSLB*. https://www.loadbalancer.org/blog/ultimate-guide-to-gslb/ (← S4, S9)
- [E10] Imperva. *DNS Load Balancing & Failover*. https://www.imperva.com/learn/availability/dns-load-balancing-failover/ (← S9)
- [E11] Network Computing. *GSLB Reality Check*. https://www.networkcomputing.com/data-centers/gslb-reality-check (← S9)
- [E12] Tenereillo. *GSLB — The Page of Shame* (DNS 기반 GSLB의 다중 A레코드 한계 비판, 고전 문헌). http://www.tenereillo.com/GSLBPageOfShame.htm (← S9)
- [E13] Dyn. *Why Client-Side GSLB Is the Fastest Load Balancing Solution*. https://www.dyncond.com/why-client-side-gslb-is-by-far-the-fastest-load-balancing-solution/ (← S9)

하드웨어 어플라이언스 (Citrix NetScaler)

- [E14] Citrix/NetScaler Docs. *Configuring Metrics Exchange Protocol (MEP)*. https://docs.netscaler.com/en-us/citrix-adc/current-release/global-server-load-balancing/configuring-metrics-exchange-protocol.html (← S4, S8, S9)
- [E15] NetScaler Docs. *Parent-child topology deployment*. https://docs.netscaler.com/en-us/citrix-adc/current-release/global-server-load-balancing/deployment-types/parent-child-topology-deployment.html (← S4)
- [E16] NetScaler Docs. *How DNS works with GSLB*. https://docs.netscaler.com/en-us/citrix-adc/current-release/global-server-load-balancing/how-dns-works-with-gslb.html (← S8)
- [E17] NetScaler Docs. *Configure GSLB for disaster recovery*. https://docs.netscaler.com/en-us/citrix-adc/current-release/global-server-load-balancing/how-to/configure-gslb-disaster-recovery.html (← S9)
- [E18] NetScaler Docs. *GSLB Monitoring*. https://docs.netscaler.com/en-us/citrix-adc/current-release/global-server-load-balancing/monitoring.html (← S9)
- [E19] Citrix Support. *CTX111081 — GSLB*. https://support.citrix.com/article/CTX111081 (← S8)

- [E20] Citrix Tech Zone. *NetScaler ADC GSLB Reference Architecture*. https://community.citrix.com/tech-zone/design/reference-architectures/adc-gslb/ (← S9)
- [E21] Citrix Tech Zone. *NetScaler ADC AWS GSLB Deployment Guide*. https://community.citrix.com/tech-zone/build/deployment-guides/netscaler-adc-aws-gslb/ (← S9)
- [E22] Citrix Tech Zone. *NetScaler ADC Azure GSLB Deployment Guide*. https://community.citrix.com/tech-zone/build/deployment-guides/netscaler-adc-azure-gslb/ (← S9)
- [E23] NetScaler Community. *Cheat Sheet: ADC Troubleshooting — GSLB*. https://community.netscaler.com/articles/netscaler/cheat-sheet-adc-troubleshooting-gslb/ (← S9)

클라우드 매니지드 (AWS·Google Cloud)

- [E24] AWS. *Best practices for Amazon Route 53 DNS*. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/best-practices-dns.html (← S4)
- [E25] AWS Architecture Blog. *Disaster Recovery of Workloads on AWS — Part I: Strategies for Recovery in the Cloud*. https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-i-strategies-for-recovery-in-the-cloud/ (← S9)
- [E26] AWS Architecture Blog. *Disaster Recovery of Workloads on AWS — Part IV: Multi-site Active/Active*. https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-iv-multi-site-active-active/ (← S9)
- [E27] AWS. *Disaster Recovery Options in the Cloud* (Whitepaper). https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html (← S9)
- [E28] Google Cloud. *Health check concepts*. https://cloud.google.com/load-balancing/docs/health-check-concepts (← S4)

쿠버네티스 네이티브 (K8GB·CoreDNS·Service Mesh)

- [E29] K8GB (CNCF). *Official documentation*. https://www.k8gb.io (← S6, S10)
- [E30] K8GB. *GitHub — k8gb-io/k8gb*. https://github.com/k8gb-io/k8gb (← S6)
- [E31] K8GB. *Crossplane GlobalApp example*. https://www.k8gb.io/examples/crossplane/globalapp/ (← S6)
- [E32] CoreDNS. *geoip plugin*. https://coredns.io/plugins/geoip/ (← S6)
- [E33] dmachard. *CoreDNS-GSLB* (GitHub). https://github.com/dmachard/CoreDNS-GSLB (← S6)
- [E34] dmachard. *CoreDNS-GSLB — Health checks*. https://github.com/dmachard/CoreDNS-GSLB/blob/main/docs/healthchecks.md (←

S6)
- [E35] Istio. *Deployment Models*. https://istio.io/latest/docs/ops/deployment/deployment-models/ (← S6)
- [E36] Linkerd. *Multi-cluster communication*. https://linkerd.io/2.15/features/multicluster/ (← S6)

가용성 아키텍처·Split-Brain

- [E37] SIOS. *Split-Brain Scenarios*. https://us.sios.com/blog/split-brain-scenarios/ (← S9)
- [E38] Serverion. *Active-Active Architecture: The Ultimate Guide*. https://www.serverion.com/uncategorized/active-active-architecture-ultimate-guide/ (← S4)

장별 외부 출처 매핑

장	근거로 삼은 대표 외부 출처
1. 개요	E5, E6, E7, E8, E9
2. AA vs AP 의사결정	E25, E26, E27, E38
3. DNS·Anycast	E1, E2, E10, E12, E13, E16
4. 헬스체크·페일오버	E24, E28, E14, E18
5. 데이터 복제·정합성	E37, E38, E27
6. 세션·TTL	E2, E3, E4
7. 하드웨어 어플라이언스	E5, E14, E15, E19, E20, E23
8. 클라우드 매니지드	E7, E24, E28
9. K8GB	E29, E30, E31, E32, E33, E34, E35, E36
10. 비교·결정 트리	E21, E22, E25, E26
11. 실무·TCO·규제	E24, E27, E28
12. 결론·로드맵	E5, E27, E29

> *벤더 제품 상세(AWS Route 53·Azure Traffic Manager·GCP Cloud DNS·Cloudflare·IBM NS1·F5 BIG-IP DNS·A10 Thunder·Kemp LoadMaster GEO 등)의 1차 근거는 각 벤더 공식 문서이며, 본 백서는 사내 제품 비교 분석(S5)이 정리한 내용을 인용했습니다.*

Glossary

용어	정의
GSLB (Global Server Load Balancing)	지리적으로 분산된 데이터센터·클라우드 리전에 DNS 기반으로 트래픽을 지능적으로 분배하는 광역 서버 부하 분산 기술.
Active-Active	모든 사이트가 동시에 트래픽을 처리하는 구성. 재해 시에도 서비스가 멈추지 않으며 RTO가 수초~수십초로 짧다.
Active-Passive	Primary만 트래픽을 처리하고 Standby가 대기하는 구성. 단순하지만 전환에 수분~수십분이 걸린다.
RTO (Recovery Time Objective)	장애 발생 후 서비스를 정상 복구하기까지 허용하는 목표 복구 시간.
RPO (Recovery Point Objective)	장애 시 허용 가능한 데이터 손실 범위를 시간으로 표현한 목표 복구 시점.
DNS (Domain Name System)	도메인 이름을 IP 주소로 변환하는 인터넷 이름 해석 체계. GSLB의 트래픽 분산 기반.
Anycast	동일 IP를 여러 지역에서 광고하고 BGP가 최근접 노드로 전달하는 방식. TTL과 무관하게 네트워크 레벨에서 경로를 전환한다.
TTL (Time To Live)	DNS 응답이 캐시에 유지되는 유효 시간. 낮출수록 전환이 빠르지만 질의 부하가 증가한다.
EDNS Client Subnet	클라이언트의 대략적 위치 정보를 리졸버가 전달해 근접 사이트 판단을 돕는 DNS 확장.
헬스체크 (Health Check)	L3/L4/L7·애플리케이션 레벨로 사이트 상태를 주기적으로 점검해 장애를 감지하는 절차.
페일오버 (Failover)	장애 사이트를 트래픽 대상에서 제외하고 정상 사이트로 요청을 전환하는 동작.
Split-Brain	네트워크 분단으로 양쪽이 각자 주도권을 갖고 데이터가 갈라지는 상태. Quorum·Witness·Fencing으로 방지한다.
Quorum	분산 노드 과반의 합의로 주도권을 결정해 Split-Brain을 막는 방식.
LWW (Last-Write-Wins)	타임스탬프가 가장 최근인 쓰기를 채택하는 단순 충돌 해소 방식. 데이터 유실 위험이 있다.
CRDT (Conflict-free Replicated Data Type)	병합 시 충돌 없이 수렴하도록 설계된 복제 자료형.

용어	정의
Stateless	서버가 세션 상태를 보관하지 않는 설계. JWT 등 토큰으로 상태를 클라이언트가 지녀 Active-Active에 유리하다.
K8GB (Kubernetes Global Balancer)	CoreDNS 임베드와 컨트롤러로 전용 관리 클러스터 없이 멀티클러스터 GSLB를 구현하는 CNCF 프로젝트.
GitOps	Git을 단일 소스로 삼아 ArgoCD/Flux 등으로 선언적으로 배포·복구하는 운영 방식.
TCO (Total Cost of Ownership)	하드웨어·라이선스·인력·운영을 포함한 총소유비용. CapEx와 OpEx 관점으로 비교한다.
클라우드 네이티브 (Cloud Native)	컨테이너·마이크로서비스·불변 인프라·선언적 API를 기반으로 오케스트레이터가 워크로드를 동적으로 운영하는 시스템 구축 방식(CNCF 정의).
마이크로서비스 (Microservices)	애플리케이션을 독립 배포 가능한 작은 서비스들로 분해한 구조. 상태 외부화와 결합하면 Active-Active 구성에 유리하다.
서비스 메시 (Service Mesh)	Istio·Linkerd 등으로 서비스 간(East-West) 통신의 라우팅·mTLS·재시도를 담당하는 계층. GSLB(North-South)와 보완 관계.
불변 인프라 (Immutable Infrastructure)	실행 중 자원을 수정하지 않고 새 버전으로 교체·재생성하는 운영 방식. 사이트 재구축을 반복 가능하게 만든다.
North-South / East-West	North-South는 외부와 클러스터 사이의 진입 트래픽(GSLB 담당), East-West는 클러스터 내부·간 서비스 통신(서비스 메시 담당).

GSLB 기반 Active-Active 구성

CONTACT

WEB
msap.ai
www.msap.ai/

EMAIL
hello@msap.ai

TEL
02-6953-5427
0269535427

YOUTUBE
@msaptv
www.youtube.com/@msaptv

LINKEDIN
linkedin.com/showcas...
www.linkedin.com/showcase/msap-ai/

FACEBOOK
facebook.com/opennaru
www.facebook.com/opennaru

SCAN