서비스 가용성을 높이기 위한 엔지니어링

대규모 서비스 개발 및 운영을 위한 실전 엔지니어링 워크숍

목차

1. 서비스 가용성 개요

1.1 서비스 가용성이란?

- 서비스가 요구되는 기능을 **지속적으로 제공하는 능력**
- 사용자에게 **서비스 중단 없이** 안정적인 기능 제공

1.2 서비스 가용성의 중요성

- **사용자 만족도 향상** 및 **이탈 방지**
- **비즈니스 신뢰성** 확보 및 **매출 손실 방지**
- **브랜드 이미지** 향상과 경쟁 우위 확보

1. 서비스 가용성 개요

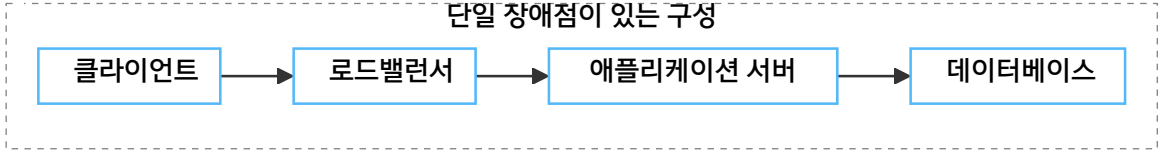
1.3 가용성 측정 지표

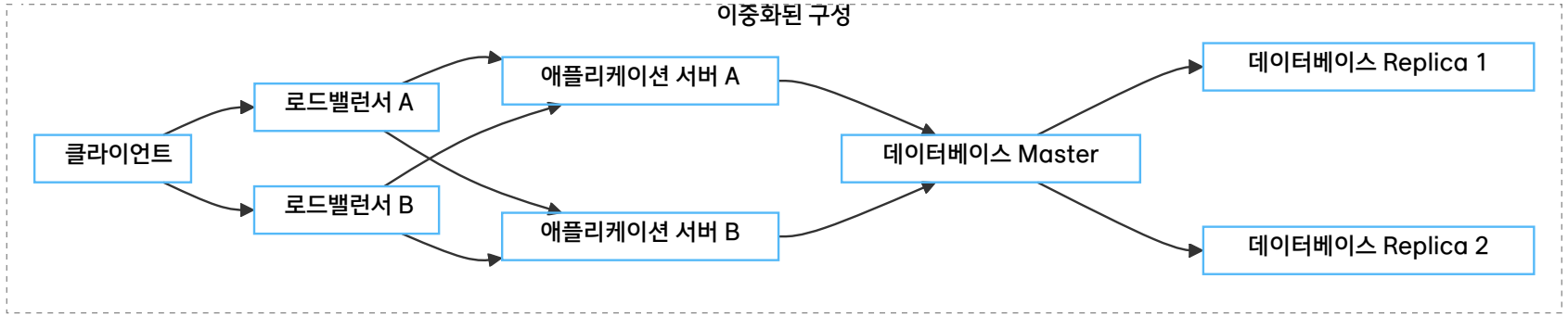
- **SLA (Service Level Agreement)**: 서비스 수준 협약
- **가용성 퍼센트**: 연간 서비스 중단 허용 시간
  - 예) 99.99% 가용성은 연간 약 52.6분의 다운타임 허용
- **MTBF (Mean Time Between Failures)**: 평균 고장 간격
- **MTTR (Mean Time To Repair)**: 평균 복구 시간
- **MTBF와 MTTR를 통한 가용성 계산**: 가용성 = $\frac{MTBF}{MTBF+MTTR}$

2. 고가용성 설계 원칙

2.1 단일 장애점 제거

- **단일 장애점(SPOF, Single Point of Failure)** 제거하여 시스템 안정성 향상
- 모든 구성 요소의 **이중화** 또는 **다중화** 설계

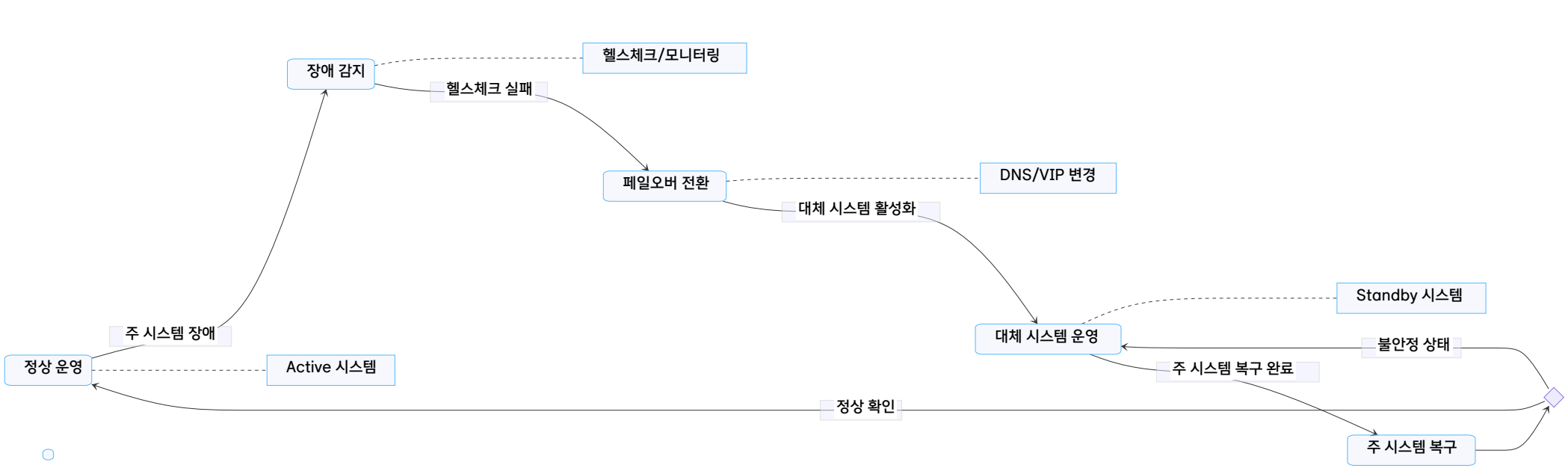




2. 고가용성 설계 원칙

2.2 중복성(Redundancy) 및 페일오버(Fail Over)

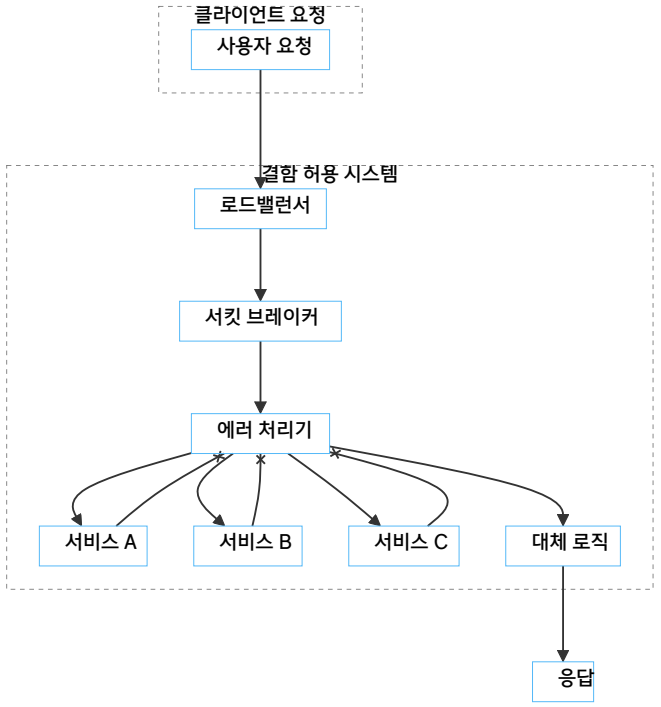
- **하드웨어 및 소프트웨어 중복**을 통한 페일오버 지원
- 장애 발생 시 자동으로 **대체 시스템**으로 전환



2. 고가용성 설계 원칙

2.3 결함 허용 (Fault Tolerance)

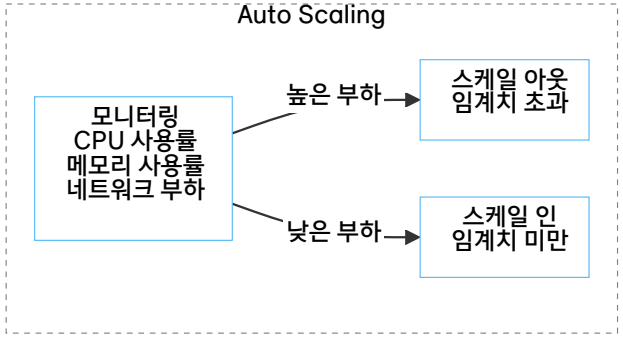
- 시스템 일부에 장애가 발생해도 전체 서비스는 지속
- **에러 처리** 및 **예외 처리** 강화

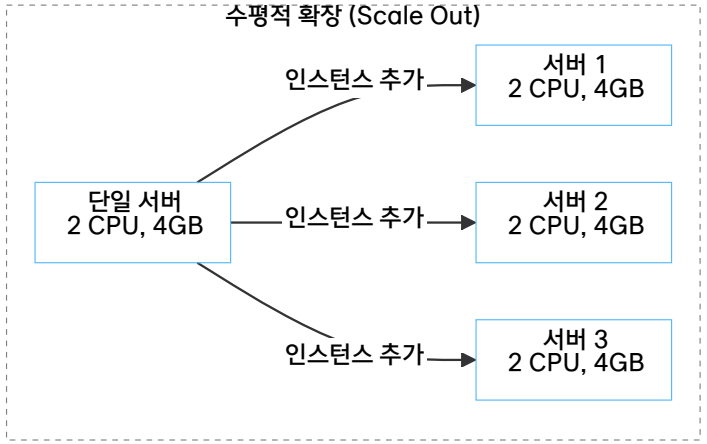


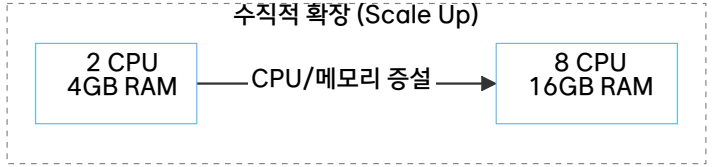
2. 고가용성 설계 원칙

2.4 확장성 있는 설계

- 수평 및 수직적 **확장성(Scalability)** 고려
- **Auto Scaling**을 통한 동적 자원 관리



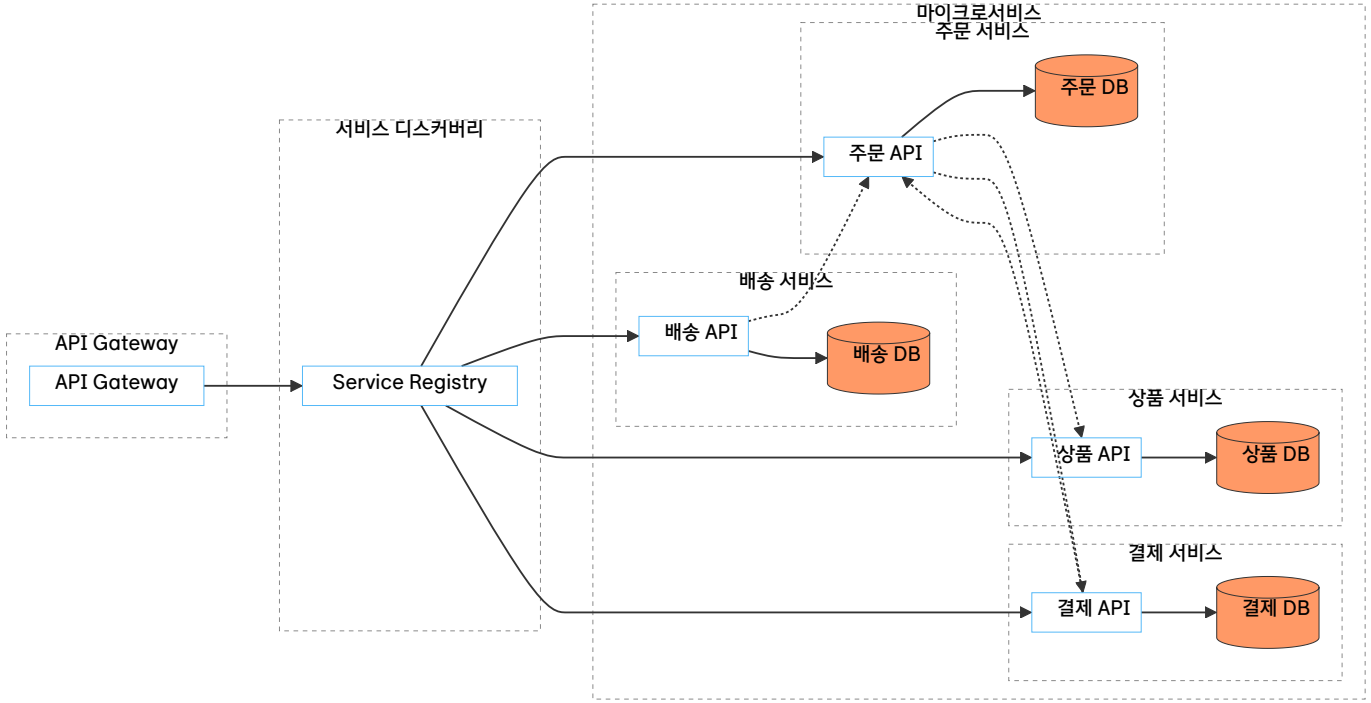




2. 고가용성 설계 원칙

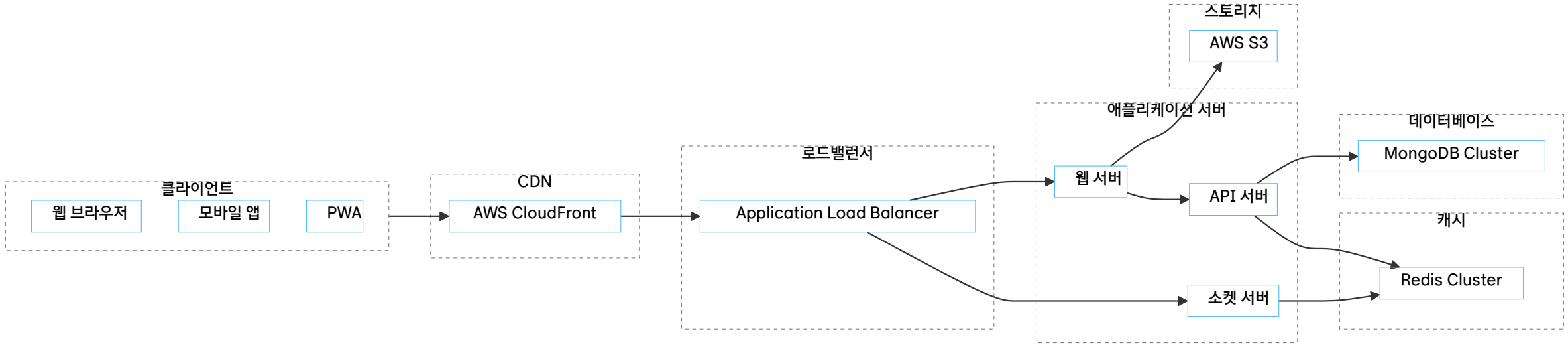
2.5 분산 시스템 구조

- **마이크로서비스 아키텍처(MSA)** 도입
- 서비스 간 **의존성 최소화**로 장애 격리



3. 고가용성 아키텍처 구축

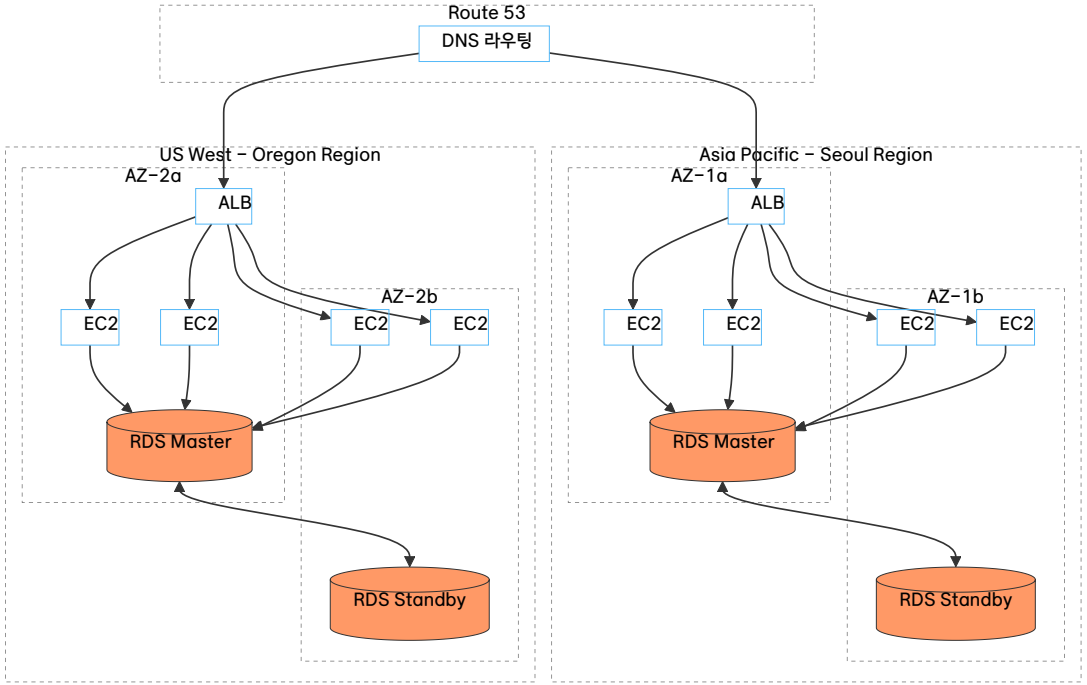
3.1 기본적인 아키텍처 다이어그램



3. 고가용성 아키텍처 구축

3.2 멀티 AZ 및 멀티 리전 배포

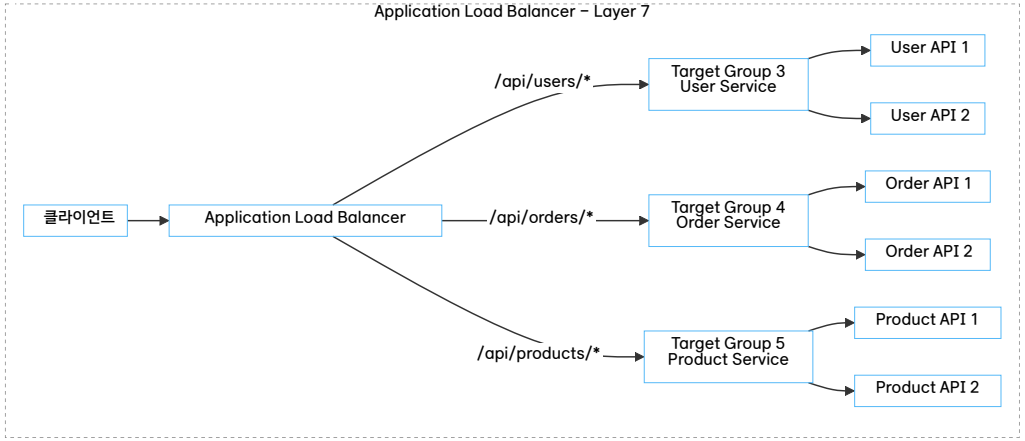
- **AWS 가용 영역(AZ)** 및 **리전**을 활용한 서비스 분산
- 지역적 장애 및 자연 재해 대비
- **Route53**을 이용한 지리적 라우팅 설정

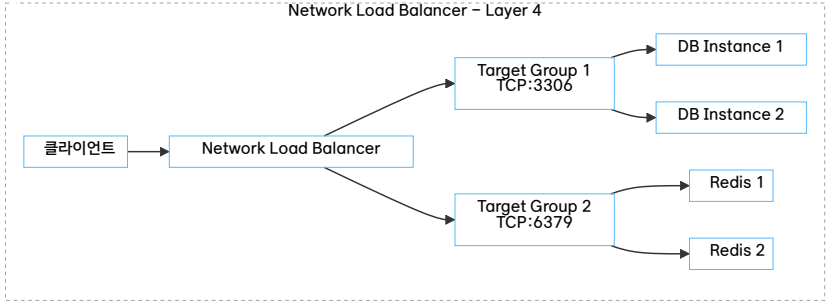


3. 고가용성 아키텍처 구축

3.3 로드 밸런싱

- **Load Balancer** 활용한 트래픽 분산
- 용도에 맞게 적절한 **계층** 선택 가능
  - **NLB(Network Load Balancer – Layer 4)**: TCP/UDP 기반으로 트래픽을 분산하며 상대적으로 속도가 빠름
  - **ALB(Application Load Balancer – Layer 7)**: HTTP/HTTPS을 기반으로 URL, Hedaer, Cookie를 통해 고급 라우팅 기능 지원
- **Target Group**을 사용하여 서비스별 로드 밸런싱

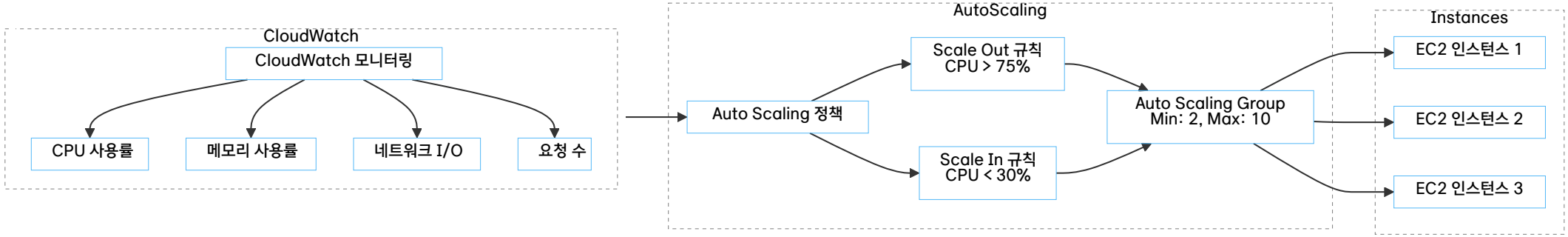




3. 고가용성 아키텍처 구축

3.4 Auto Scaling 그룹

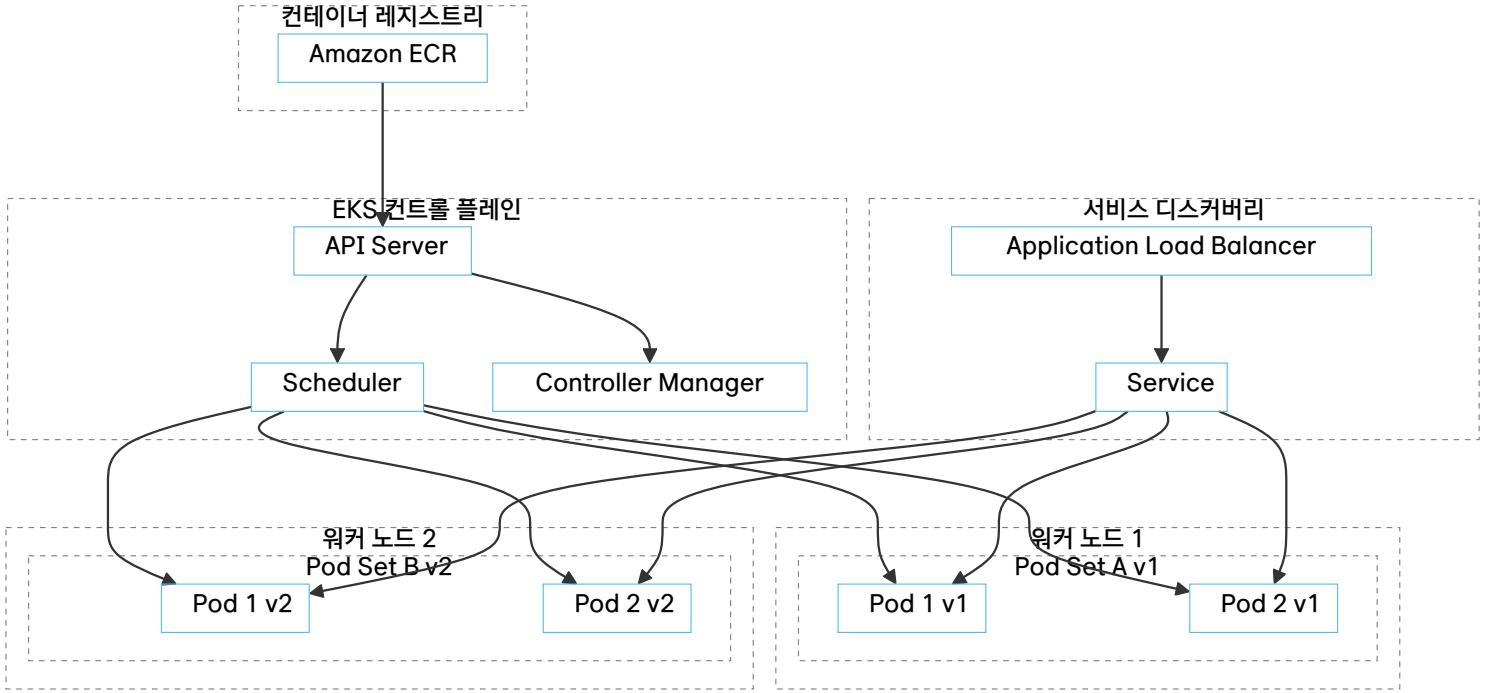
- Auto Scaling을 통한 동적 인스턴스 관리
- 트래픽 증가 시 자동으로 인스턴스 수 확장
- CloudWatch 지표 기반으로 스케일링 정책 설정



3. 고가용성 아키텍처 구축

3.5 컨테이너 오케스트레이션

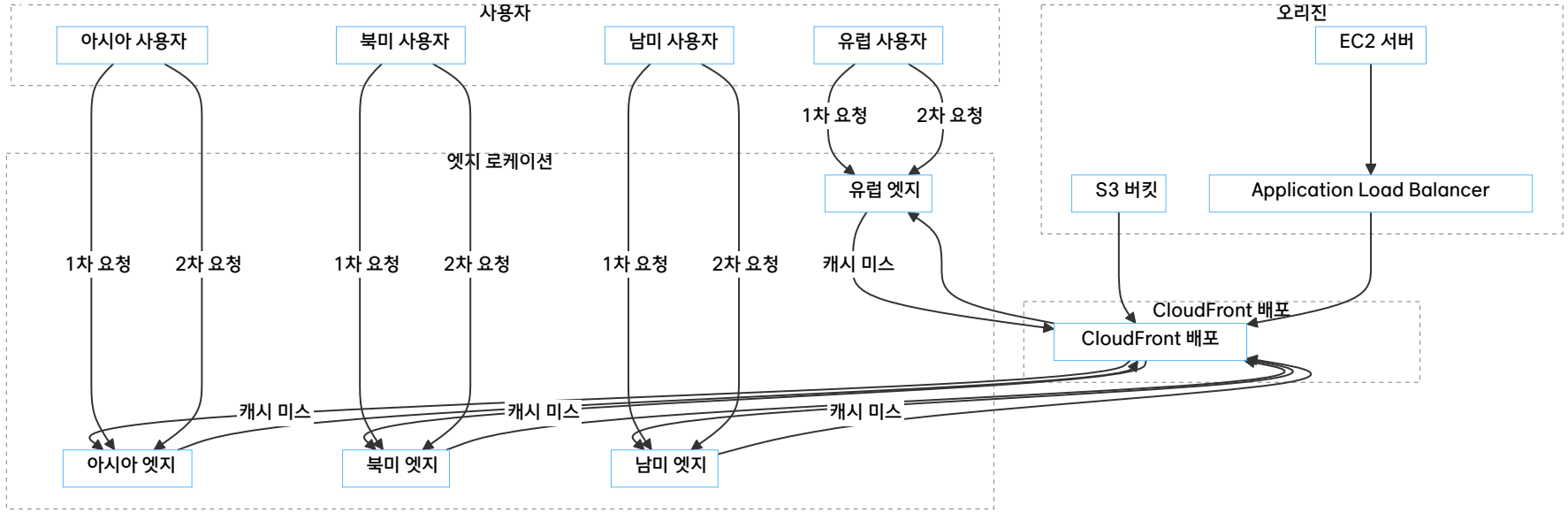
- K8S, AWS ECS, EKS를 통한 컨테이너 관리
- **무중단 배포** 및 **자동 복구** 지원



3. 고가용성 아키텍처 구축

3.6 콘텐츠 전송 네트워크(CDN) 활용

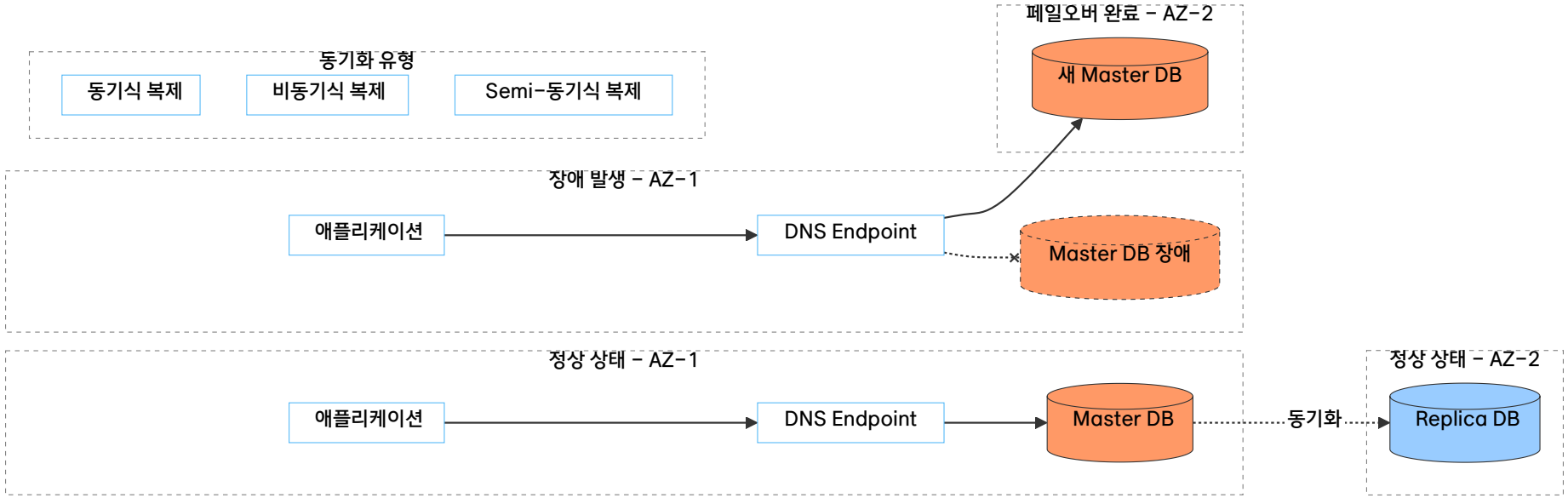
- CloudFront를 통한 정적 콘텐츠 캐싱
- 글로벌 엣지 로케이션을 통한 지연 시간 감소
- **오리진 서버** 부하 감소로 가용성 향상



4. 데이터 관리 전략

4.1 데이터베이스 이중화

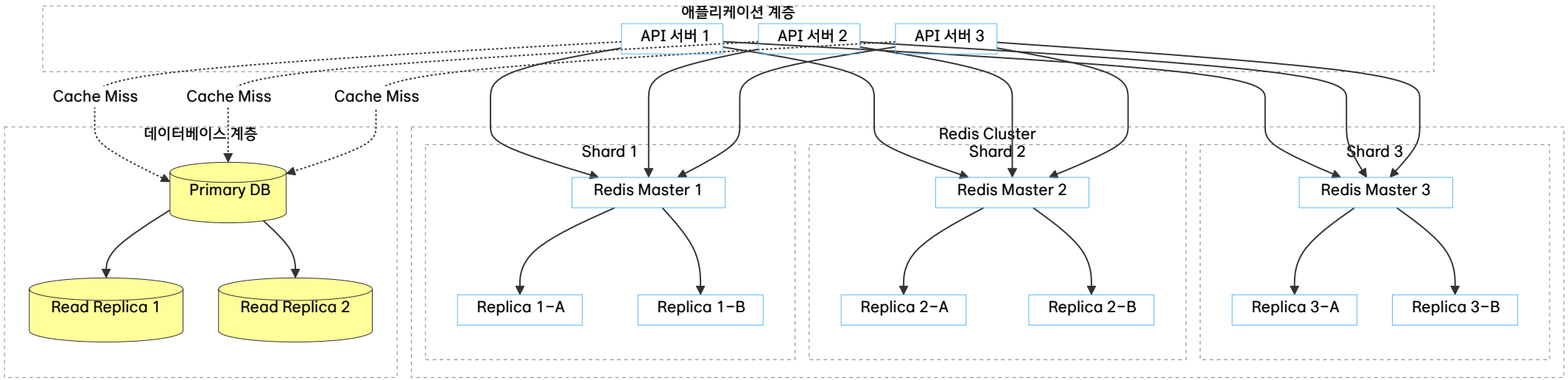
- Replication / Clustering을 통한 안정성 확보
- **다중 AZ 배포**로 데이터베이스 가용성 확보
- **자동 장애 조치(Failover)** 설정



4. 데이터 관리 전략

4.2 캐시 전략

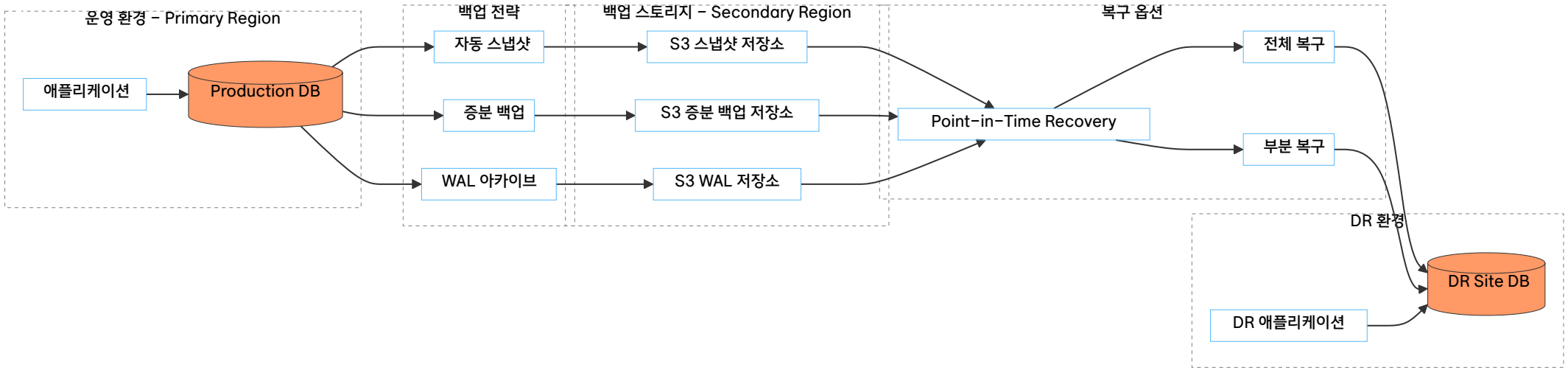
- **Redis Cluster**를 이용한 **분산 캐시** 구현
- **데이터베이스 부하 감소** 및 **응답 속도 향상**
- **캐시 이중화**로 캐시 장애 대비



4. 데이터 관리 전략

4.3 데이터 백업 및 복구

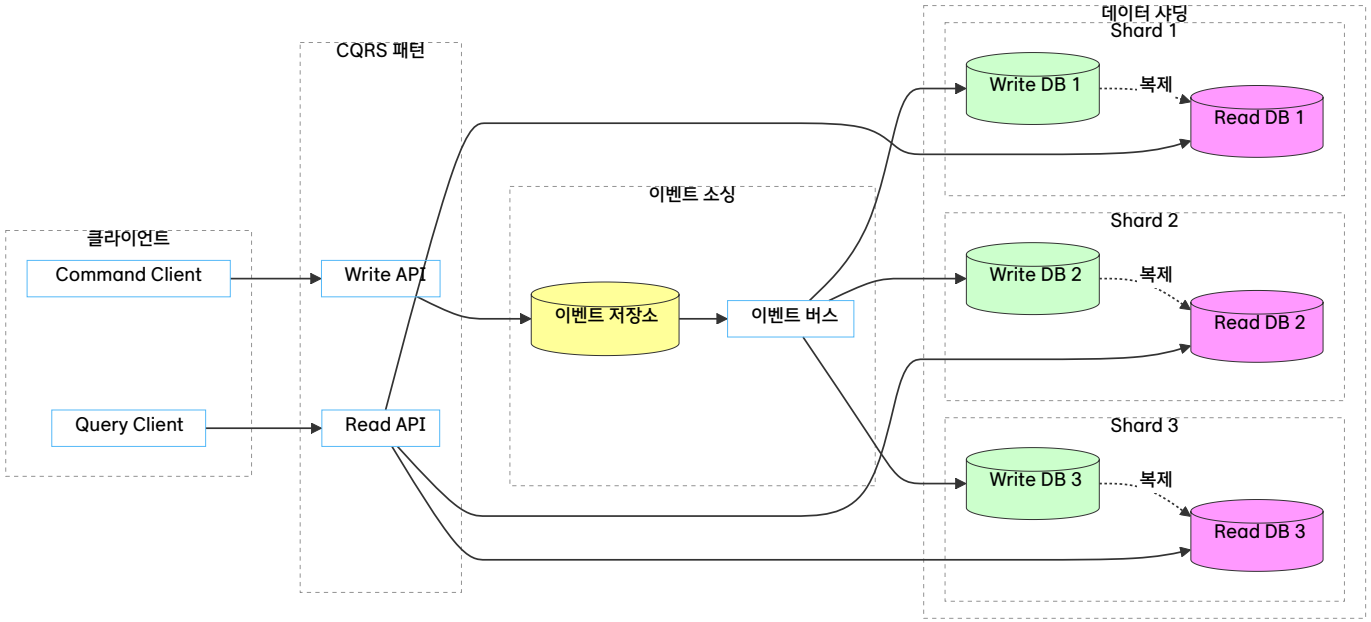
- **정기적인 스냅샷 및 백업 정책** 수립
- **Point-in-Time Recovery** 기능 활용
- **재해 복구(DR)** 계획 수립 및 테스트



4. 데이터 관리 전략

4.4 데이터 일관성 및 분산 처리

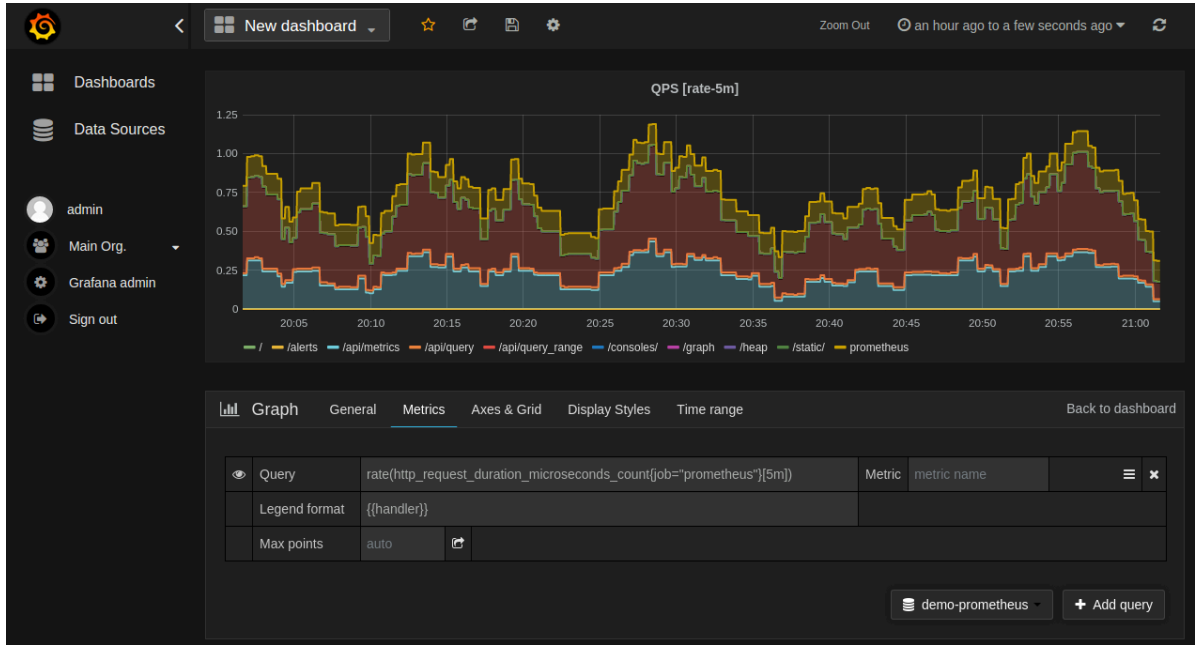
- **데이터 샤딩(Data Sharding)** 을 통한 수평 확장
- **CQRS 패턴** 적용으로 읽기/쓰기 분리
- **이벤트 소싱(Event Sourcing)** 으로 데이터 일관성 유지



5. 모니터링 및 알림 시스템

5.1 실시간 모니터링

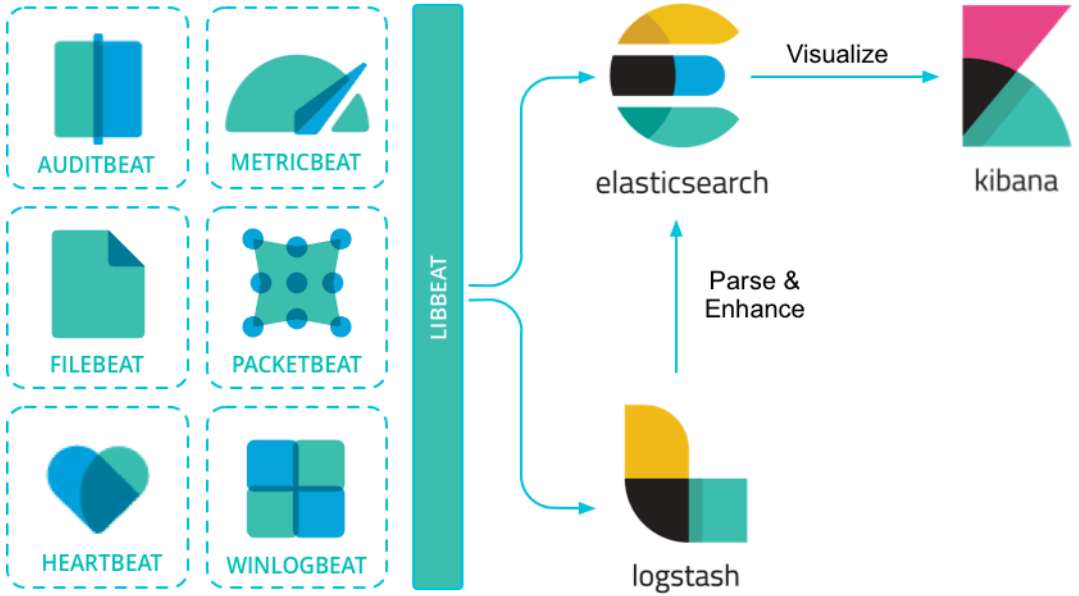
- CloudWatch, Prometheus 등을 통한 메트릭 수집
- CPU, 메모리, 네트워크 등 **시스템 자원 사용량** 모니터링
- **애플리케이션 성능 지표(APM)** 모니터링



5. 모니터링 및 알림 시스템

5.2 로그 관리

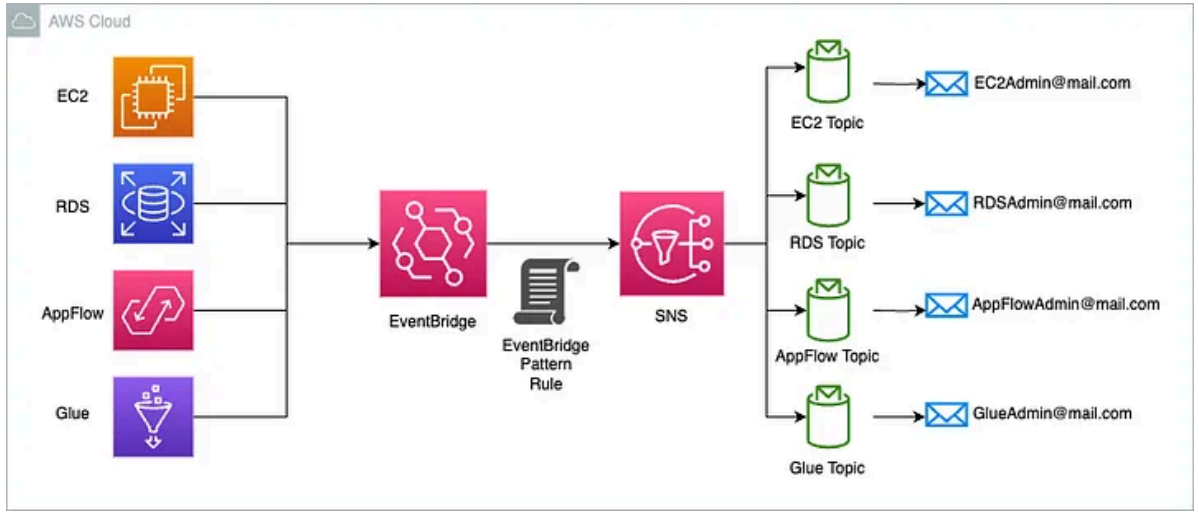
- CloudWatch Logs, ELK Stack 등을 활용한 중앙 집중식 로그 관리
- **로그 분석**을 통한 문제 원인 파악 및 추세 분석
- **로그 지표화**로 서비스 상태 모니터링



5. 모니터링 및 알림 시스템

5.3 알림 시스템 구축

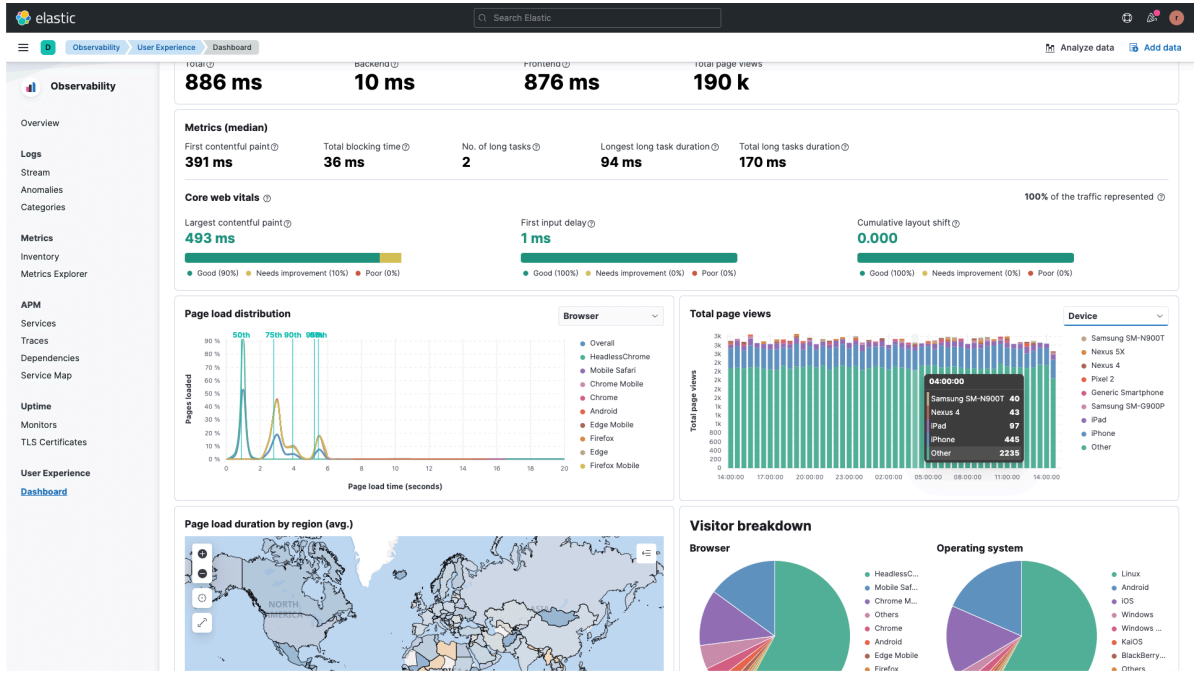
- Amazon EventBrdige + AWS SNS, PagerDuty 등을 이용한 알림 설정
- 임계치 초과 시 **자동 알림** 발송
- On-Call 체계 및 대응 프로세스 마련



5. 모니터링 및 알림 시스템

5.4 사용자 경험 지표 모니터링

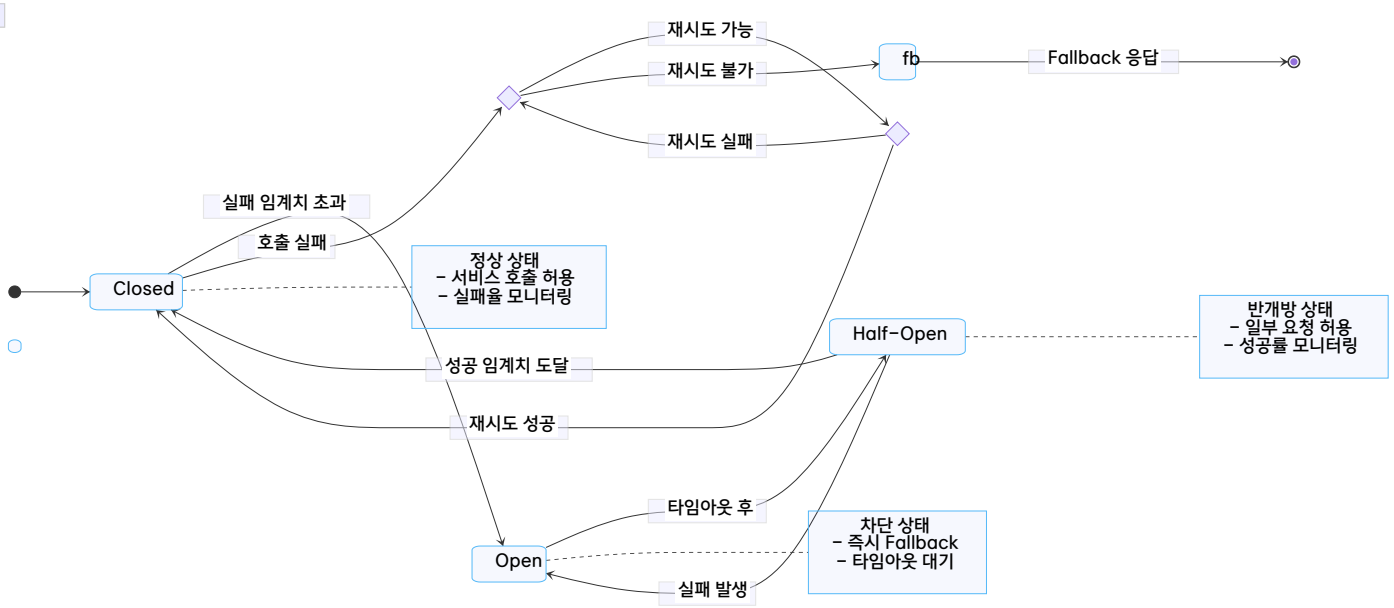
- **응답 시간, 에러율 등 애플리케이션 레벨 메트릭** 수집
- 사용자 관점에서의 **서비스 품질** 평가
- **RUM(Real User Monitoring) 도구 활용**



6. 장애 격리 및 복구 전략

6.1 서킷 브레이커 패턴

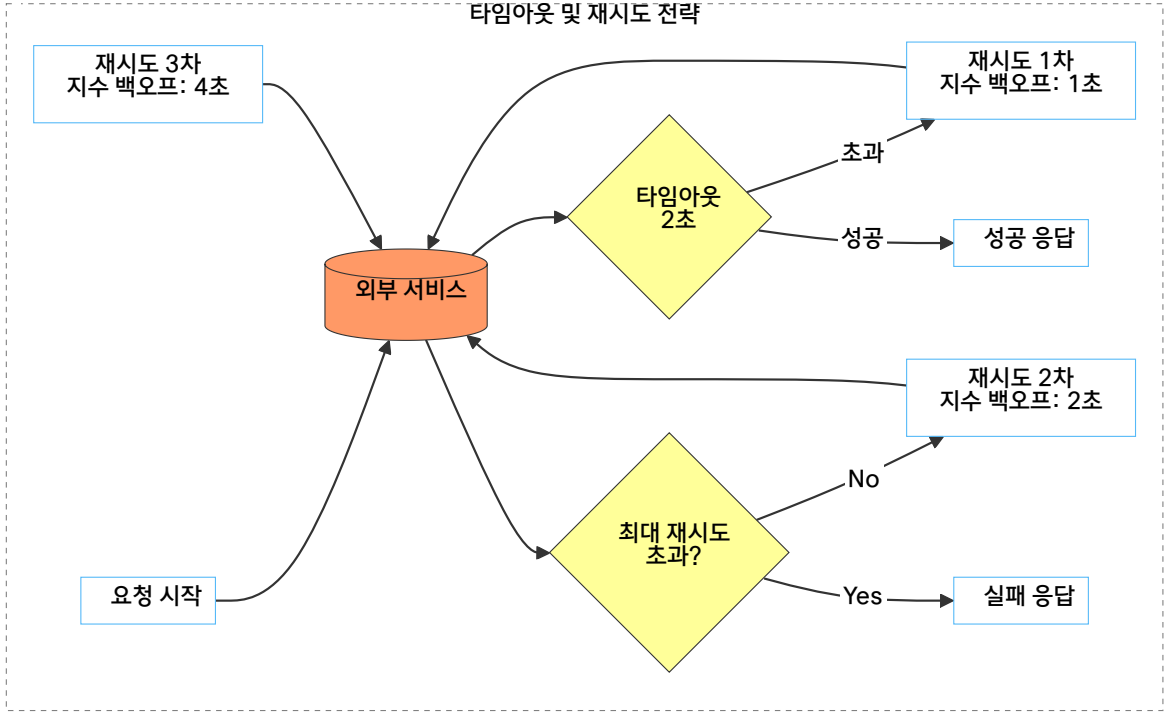
- **네트워크 호출 실패 시** 자동으로 회로 차단
- 장애 전파를 막고 시스템 안정성 유지
- **재시도 로직** 및 **후퇴(fallback) 전략** 구현



6. 장애 격리 및 복구 전략

6.2 타임아웃 설정 및 재시도 정책

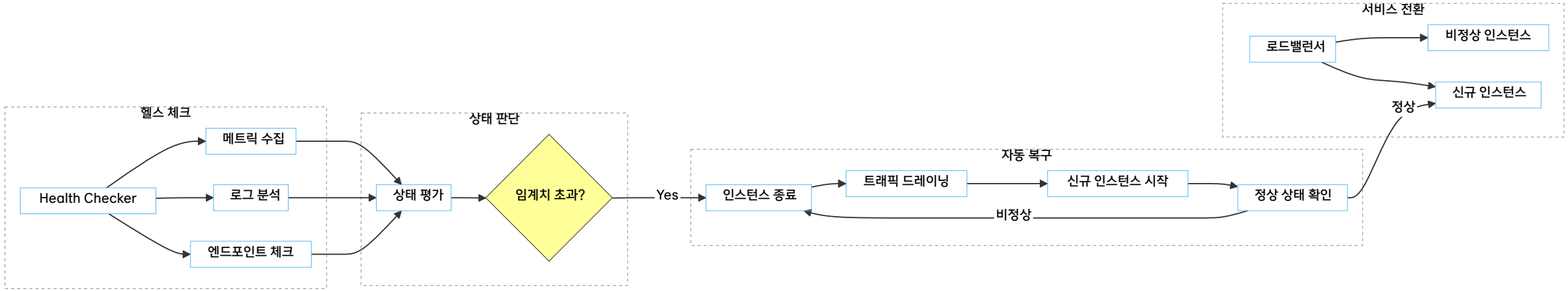
- 외부 서비스 호출 시 **적절한 타임아웃** 설정
- **재시도 횟수** 및 **대기 시간** 조절로 시스템 부하 관리



6. 장애 격리 및 복구 전략

6.3 장애 감지 및 자동 복구

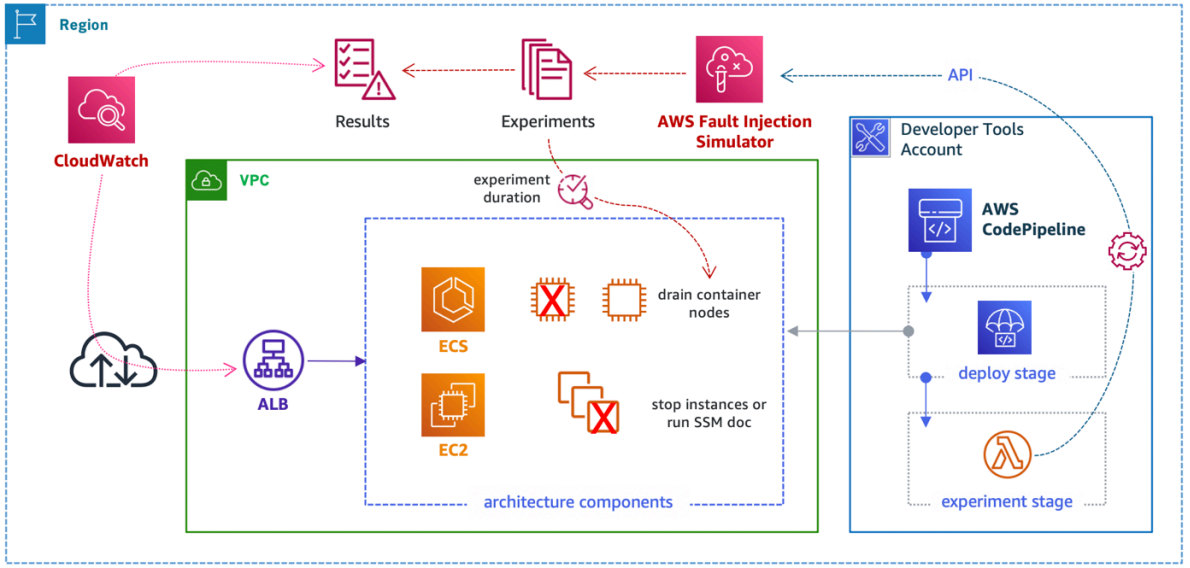
- **헬스 체크**를 통한 인스턴스 상태 모니터링
- 비정상 인스턴스 **자동 종료 및 교체**
- **Auto Healing** 메커니즘 도입



6. 장애 격리 및 복구 전략

6.4 장애 시나리오 테스트

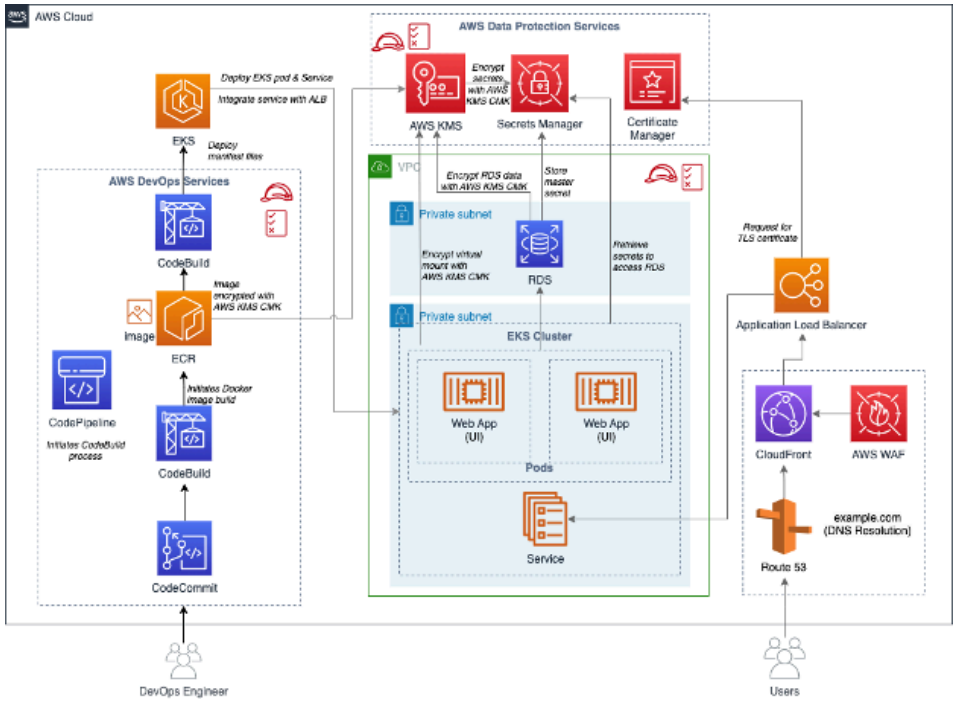
- Chaos Engineering 도입
- Chaos Monkey 등을 활용한 장애 환경 시뮬레이션
- 시스템 강건성 및 대응 능력 향상



7. 배포 및 운영 자동화

7.1 CI/CD 파이프라인 구축

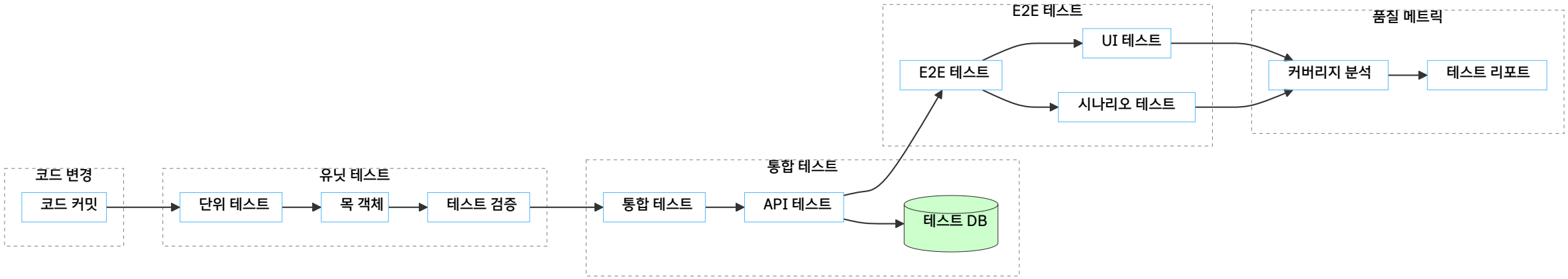
- 자동 빌드, 테스트, 배포를 통한 효율성 향상
- CodeBuild, CodeDeploy 등 도구 활용



7. 배포 및 운영 자동화

7.2 테스트 자동화

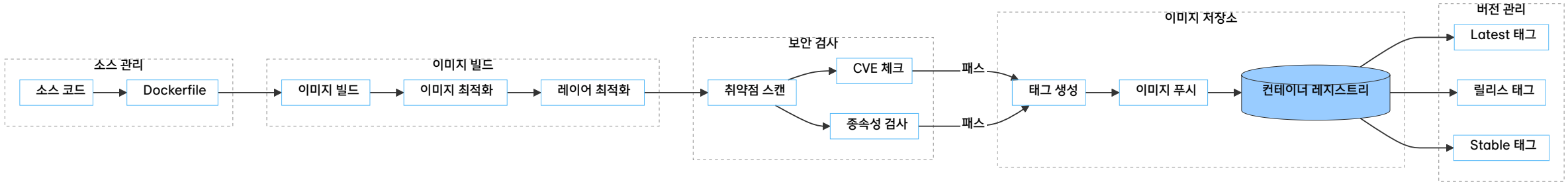
- **유닛 테스트, 통합 테스트, E2E 테스트** 자동화
- **테스트 커버리지** 향상으로 품질 보증



7. 배포 및 운영 자동화

7.3 컨테이너 이미지 관리

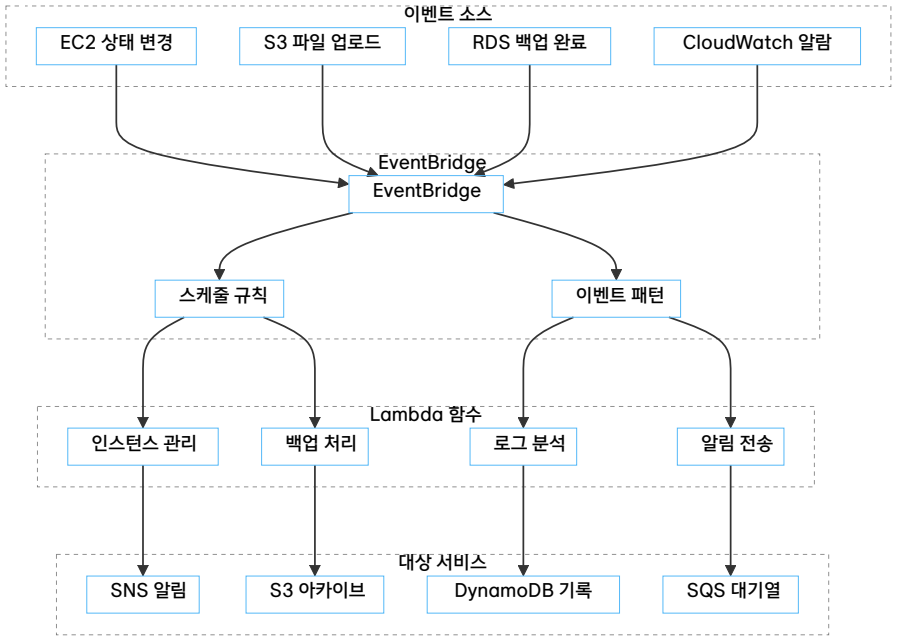
- **Docker 이미지 최적화 및 버전 관리**
- **이미지 스캔**으로 보안 취약점 사전 제거



7. 배포 및 운영 자동화

7.4 운영 자동화

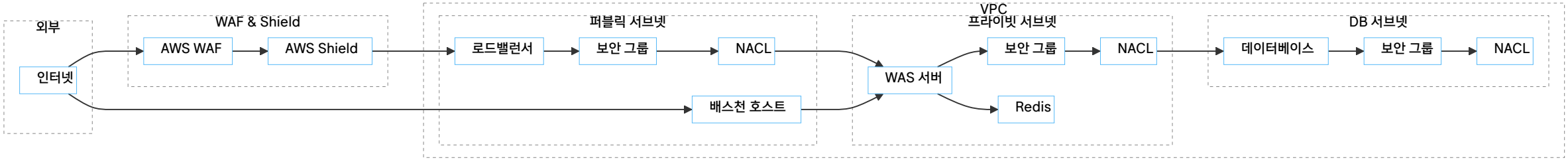
- AWS EventBridge + AWS Lambda 등을 활용한 서버리스 운영
- **스케줄링 작업** 및 **이벤트 기반 처리** 자동화



8. 보안 및 개인정보 보호

8.1 네트워크 보안

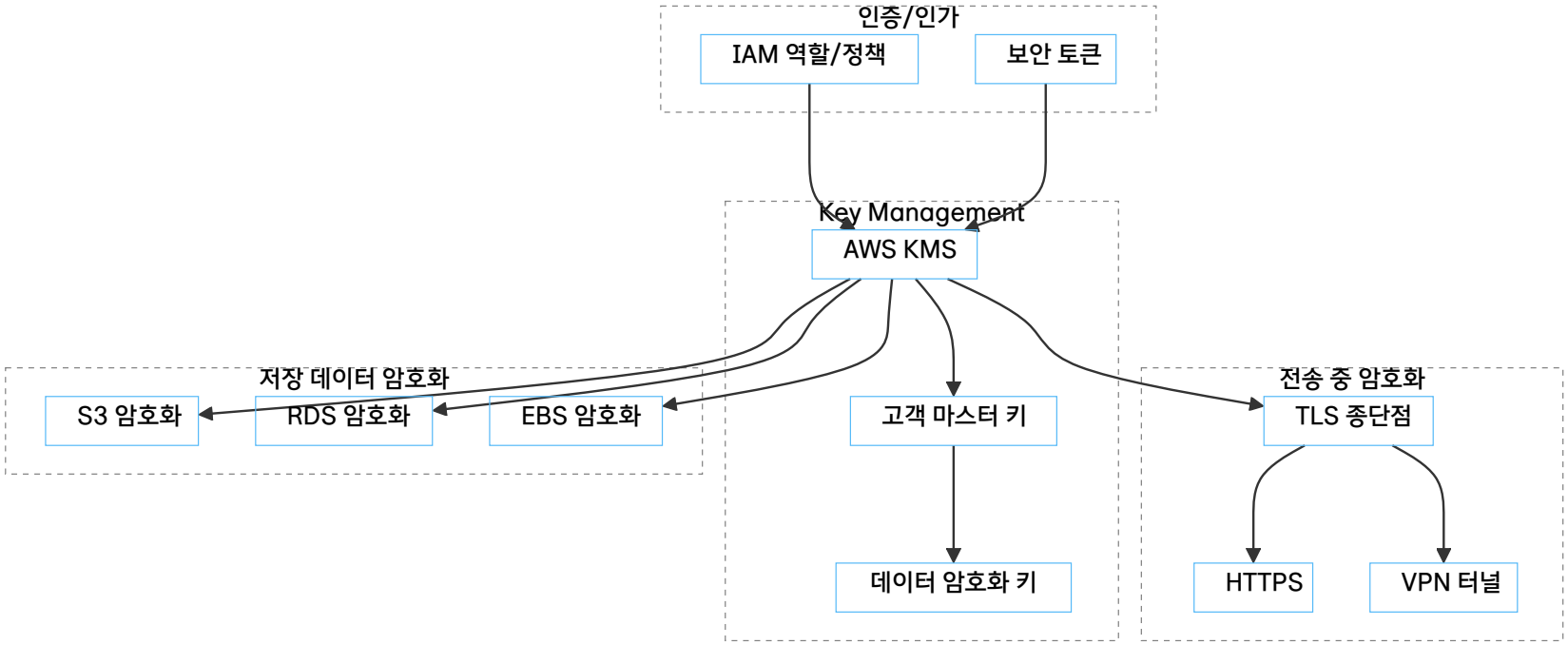
- VPC, 서브넷, 보안 그룹 설정을 통한 접근 제어
- NACL 및 AWS WAF를 통한 공격 방어



8. 보안 및 개인정보 보호

8.2 데이터 보안

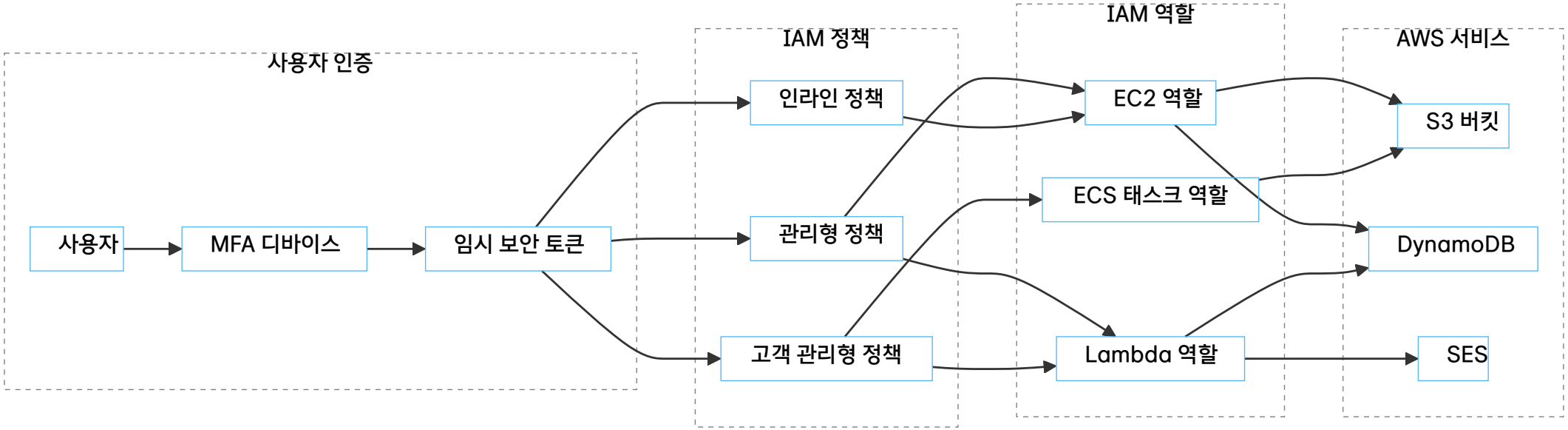
- **데이터 암호화 및 전송 계층 보안(TLS)**
- **KMS(Key Management Service) 를 통한 키 관리**



8. 보안 및 개인정보 보호

8.3 인증 및 권한 관리

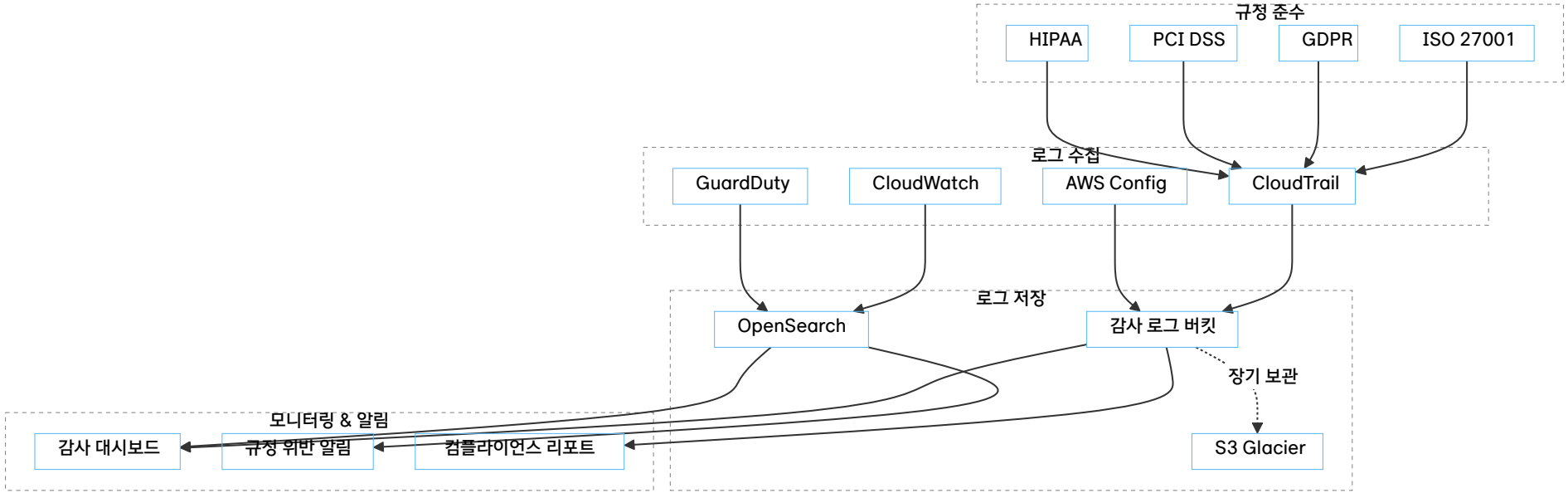
- **IAM 역할** 및 **정책**을 통한 권한 분리
- **MFA(Multi-Factor Authentication)** 적용



8. 보안 및 개인정보 보호

8.4 컴플라이언스 준수

- ISO 27001, GDPR 등 관련 법규 및 표준 준수
- 감사 로그(Audit Log) 를 통한 추적 가능성 확보



9. 이번 워크샵에서 다루는 범위

고가용성 기법	EC2	ELB	Auto Scaling	Route 53	ACM	S3	CloudFront	CloudWatch
로드 밸런싱	△	○	○	○	△	×	△	△
오토 스케일링	○	△	○	×	×	×	×	○
데이터베이스 캐싱	○	×	△	×	×	×	×	△
CDN 활용	×	×	×	○	○	○	○	△
서버 경량화	○	△	○	×	×	×	×	△
MSA 적용	○	○	○	△	△	△	△	○
메시지 큐 활용	○	×	△	×	×	×	×	△
비동기 프로세싱	○	×	○	×	×	△	×	○

- 도움이 될 수 있는 문서 : https://goorm.notion.site/AWS-1574e6997fb08081879acca350d9225d?pvs=4

10. 결론

핵심 요점 정리

- **서비스 가용성**은 사용자 만족도와 비즈니스 성공의 핵심
- 다양한 **엔지니어링 기법**과 **AWS 서비스**를 활용하여 고가용성 달성
- 지속적인 **모니터링**, **자동화**, **보안 강화**로 안정적인 서비스 운영

엔지니어로서 해야할 일

- **지속적인 개선**과 **최신 기술 도입**으로 경쟁력 강화
- **DevOps 문화** 정착 및 팀 역량 향상
- 사용자 피드백을 반영한 **서비스 품질 향상**

감사합니다!

Q&A

질문이 있으시면 자유롭게 해주세요.