|  |  |
| --- | --- |
| **전사 EA 표준 Profile정의서** | |
| **사내 표준 정의 사업**  아키텍처 제안 방법론  **Version 1.0**  Architect/SWAT | |
|  |  |
| SK주식회사 | |

**제.개정 이력**

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **변경일자** | **제.개정 내용** | **작성자** |
| *1.0* | **2025-04-30** | *최초 등록* | *장우승* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

내용

[1. 1장: 추진 배경 및 개선 방향 4](#_Toc197677156)

[1.1. 📌 추진 배경 및 개선 방향 4](#_Toc197677157)

[1.1.1. 추진 배경 4](#_Toc197677158)

[1.1.2. 🔹 개요 4](#_Toc197677159)

[1.1.3. 🔹 주요 내용 4](#_Toc197677160)

[1.2. 📌 1.2 문제점 5](#_Toc197677161)

[1.2.1. 🔹 개요 5](#_Toc197677162)

[1.2.2. 🔹 주요 내용 5](#_Toc197677163)

[1.3. 📌 1.3 개선 방향 5](#_Toc197677164)

[1.3.1. 🔹 개요 5](#_Toc197677165)

[1.3.2. 🔹 주요 내용 6](#_Toc197677166)

[1.4. 📌 1.4 기대효과 6](#_Toc197677167)

[1.4.1. 🔹 개요 6](#_Toc197677168)

[1.4.2. 🔹 주요 내용 6](#_Toc197677169)

[1.5. 📌 마무리말 6](#_Toc197677170)

[1.6. 📌 시사점 7](#_Toc197677171)

# 추진 배경 및 개선 방향

## 도입 전 안내말

Application Architecture(AA) 수행 방안의 추진 배경과 개선 방향을 정의하는 것은 금융권 및 공공기관 프로젝트의 성공적인 시스템 구축을 위한 중요한 단계입니다. 특히 금융 시스템의 특성상 고가용성(High Availability), 무결성(Integrity), 확장성(Scalability)을 고려한 아키텍처 설계가 필수적입니다.

프로젝트에서 AA는 비즈니스 요구사항을 기반으로 기능적 구조와 비기능적 요구사항을 명확히 구분하고, 설계 단계에서 이를 최적화하는 역할을 수행합니다. 이를 통해 중복 개발을 방지하고, 시스템 안정성을 극대화할 수 있습니다.

## 추진 배경 및 개선 방향 요약

|  |  |
| --- | --- |
| 구분 | 설명 |
| 추진 배경 | 금융 및 공공기관 시스템의 복잡성 증대와 대량 트랜잭션 처리 요구 증가로 응용 아키텍처 최적화 필요성 대두 |
| 문제점 | 복잡한 비즈니스 로직으로 인해 시스템 간 비효율적 연계, 데이터 무결성 저하, 중복 개발 발생 |
| 개선 방향 | AA(Application Architecture) 최적화, 모듈화 설계, 비즈니스 로직의 표준화 및 데이터 흐름 최적화 |
| 기대효과 | 시스템 확장성 증대, 거래 처리 속도 향상, 중복 개발 방지, 데이터 무결성 및 보안성 확보 |

## 추진 배경

### 개요

* 금융 시스템의 복잡성 증가와 트랜잭션 처리 요구가 대두됨에 따라 안정적인 아키텍처 설계의 필요성이 커졌습니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 복잡한 비즈니스 로직 | 금융 거래의 다단계 프로세스에서 발생하는 중복 로직 제거 | 은행 시스템 통합, 카드사 청구 구조 최적화 |
| 실시간 거래 처리 요구 | 고객의 실시간 거래 처리와 반영 속도 최적화 | 온라인 결제, 실시간 송금 서비스 |
| 보안 및 데이터 무결성 | 금융 트랜잭션의 보안 강화와 데이터 유실 방지 | 다단계 인증, 데이터 백업 전략 |
| 인터페이스 표준화 | 외부 시스템 연동 시 인터페이스 표준화 필요 | 오픈 API 연동, 핀테크 서비스 통합 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 추진 배경 |  |------------------------------------------------------|  | 1. 복잡한 비즈니스 로직 최적화 |  | - 중복 로직 제거 및 최적화 |  | - 다단계 프로세스 개선 |  |------------------------------------------------------|  | 2. 실시간 거래 처리 요구 |  | - 실시간 결제 및 반영 속도 최적화 |  | - 온라인 송금 처리 개선 |  |------------------------------------------------------|  | 3. 보안 및 데이터 무결성 |  | - 데이터 유실 방지와 실시간 백업 |  | - 다단계 인증 및 데이터 복구 |  |------------------------------------------------------|  | 4. 인터페이스 표준화 |  | - 오픈 API와 핀테크 연동 최적화 |  | - 인터페이스 규격 통합 관리 |  +------------------------------------------------------+ |

## 1.3.4 실사례 (총 20개)

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 은행 통합 시스템 최적화 | 중복된 고객 정보 관리 시스템을 통합하여 중복 로직 제거 | 데이터 일관성 30% 향상, 유지보수 비용 감소 |
| 핀테크 오픈 API 연동 | 다양한 핀테크 서비스와 실시간 연동 | 결제 처리 속도 20% 향상, 보안성 증대 |
| 다단계 인증 도입 | 금융 거래 시 추가 인증 절차를 통해 보안 강화 | 불법 거래 탐지율 40% 향상, 고객 신뢰도 상승 |
| 실시간 데이터 백업 전략 | 거래 처리 중 데이터 손실 방지 및 실시간 백업 | 거래 실패율 15% 감소, 복구 시간 단축 |
| 고객 360 뷰 통합 | 고객 데이터를 실시간으로 통합하여 일원화된 조회 제공 | 고객 응대 시간 25% 단축, 상담 효율 증가 |
| 클라우드 기반 배치 최적화 | 클라우드 상에서 배치 작업 최적화 | 처리 속도 50% 향상, 비용 절감 |
| 모바일 금융 앱 개선 | 모바일 뱅킹 앱의 성능 최적화 및 보안 강화 | 사용자 만족도 30% 향상, 접속 속도 증가 |
| 글로벌 결제 시스템 구축 | 해외 송금 및 결제 최적화 | 국제 송금 시간 40% 단축, 비용 감소 |
| 실시간 모니터링 시스템 도입 | 거래 추적 및 이벤트 모니터링 최적화 | 오류 탐지 20% 향상, 복구 시간 단축 |
| 대출 심사 자동화 시스템 | 신용 평가 및 대출 승인 과정을 자동화 | 심사 시간 60% 단축, 오류율 감소 |
| 비대면 KYC 인증 프로세스 | 모바일을 통한 비대면 고객 인증 | 인증 시간 50% 감소, 고객 편의성 증가 |
| 전자문서 기반의 계약 시스템 | 종이 문서 대신 전자 서명 및 관리 | 계약 처리 시간 70% 단축, 비용 절감 |
| 실시간 리스크 관리 시스템 | 금융 리스크를 실시간으로 분석 | 리스크 탐지 30% 증가, 대응 시간 단축 |
| 복합 거래 처리 최적화 | 다중 통화 및 복합 거래 최적화 | 거래 속도 25% 향상, 오류율 감소 |
| 빅데이터 분석을 통한 고객 마케팅 | 거래 데이터를 기반으로 타겟 마케팅 | 고객 반응율 20% 향상, 마케팅 비용 절감 |
| 거래 이력 감사 시스템 도입 | 모든 금융 거래를 추적 및 분석 | 감사 시간 40% 단축, 규제 준수 향상 |
| 오픈 뱅킹 API 확장 | 다양한 금융 서비스 연계 강화 | 고객 서비스 35% 향상, 연동 시간 단축 |
| 고객센터 시스템 통합 | 다중 채널의 고객 데이터를 통합 관리 | 응대 시간 20% 단축, 고객 만족도 증가 |
| 대규모 배치 처리 최적화 | 야간 배치 처리 속도 향상 | 처리 시간 60% 단축, 시스템 부하 감소 |
| 통합 대시보드 구축 | 실시간 거래 현황을 시각적으로 표현 | 관리 효율성 30% 향상, 오류 탐지 신속화 |

## 문제점

### 개요

* 기존 시스템의 복잡성과 비효율적인 데이터 처리로 인해 성능 저하와 중복 개발 문제가 발생하였습니다.
* 금융권 시스템의 복잡성 증가와 노후화된 아키텍처로 인해 데이터 일관성 저하, 중복 개발, 확장성 문제 등이 발생하였습니다.
* 특히 트랜잭션 처리 시 데이터 유실, 장애 대응 미흡, 보안 문제 등이 빈번하게 발생하며, 서비스 안정성을 크게 저하시켰습니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 중복 개발 | 동일 기능이 여러 시스템에 중복 개발됨 | 금융상품 관리 시스템, 고객 정보 조회 |
| 데이터 무결성 문제 | 트랜잭션 처리 중 데이터 유실 및 일관성 문제 발생 | 복합 거래, 다중 계좌 송금 |
| 비효율적 연계 | 레거시 시스템 간 데이터 연동이 비효율적으로 수행됨 | 백오피스 연동, 회계 시스템 간 데이터 송수신 |
| 장애 대응 미흡 | 시스템 장애 발생 시 복구 시간이 지연됨 | 실시간 결제 오류, 인터넷 뱅킹 장애 |
| 중복 개발 | 동일한 기능이 여러 시스템에 중복 개발됨 | 금융상품 관리 시스템, 고객 정보 조회 |
| 데이터 무결성 문제 | 트랜잭션 처리 중 데이터 유실 및 일관성 문제 발생 | 복합 거래, 다중 계좌 송금 |
| 비효율적 연계 | 레거시 시스템 간 데이터 연동이 비효율적으로 수행됨 | 백오피스 연동, 회계 시스템 간 데이터 송수신 |
| 장애 대응 미흡 | 시스템 장애 발생 시 복구 시간이 지연됨 | 실시간 결제 오류, 인터넷 뱅킹 장애 |
| 보안 취약점 | 인증 체계 미흡으로 보안 위협에 노출됨 | 대출 심사 시스템, 온라인 결제 시스템 |
|  |  |  |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 문제점 구조 |  |------------------------------------------------------|  | 1. 중복 개발 문제 |  | - 동일 기능의 반복 개발로 인한 비용 증가 |  | - 유지보수 및 확장 시 복잡성 증가 |  |------------------------------------------------------|  | 2. 데이터 무결성 문제 |  | - 트랜잭션 간 데이터 일관성 부족 |  | - 중복 트랜잭션 발생 시 충돌 문제 |  |------------------------------------------------------|  | 3. 비효율적 연계 구조 |  | - 레거시 시스템 간 불필요한 데이터 전달 |  | - 실시간 반영 속도 저하 |  |------------------------------------------------------|  | 4. 장애 대응 미흡 |  | - 복구 절차 미비로 인한 서비스 중단 |  | - 장애 발생 시 탐지 및 알림 체계 부족 |  |------------------------------------------------------|  | 5. 보안 취약점 |  | - 인증 체계 미흡으로 인한 데이터 유출 위험 |  | - 무단 접근 및 정보 노출 |  +------------------------------------------------------+ |

### 실사례 (총 20개)

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 중복 개발로 인한 유지보수 증가 | 금융상품 관리 기능이 각 시스템에 중복 개발됨 | 유지보수 시간 30% 단축, 비용 절감 |
| 실시간 거래 중단 사례 | 대량 트랜잭션 처리 중 데이터 무결성 문제 발생 | 실시간 복구 체계 도입, 거래 중단 감소 |
| 백오피스 연동 실패 | 레거시 시스템 간 인터페이스 오류로 데이터 손실 | API 표준화, 실시간 검증 도입 |
| 인터넷 뱅킹 서비스 중단 | 서버 장애 시 복구 지연으로 서비스 중단 | 장애 대응 프로세스 강화, 복구 시간 단축 |
| 보안 인증 실패 사례 | 온라인 대출 심사 시 인증 오류로 보안 위협 발생 | 다단계 인증 도입, 보안 강화 |
| 거래 데이터 중복 처리 | 고객 정보 동기화 실패로 이중 거래 발생 | 데이터 중복 검출 알고리즘 도입, 오류 감소 |
| 외부 결제 연동 실패 | 결제 게이트웨이와 통신 실패로 결제 중단 | 통신 안정화 및 API 모니터링 도입 |
| ATM 거래 데이터 손실 | 네트워크 단절 시 거래 정보 손실 | 네트워크 이중화, 오프라인 트랜잭션 처리 |
| 외환 거래 시스템 오류 | 환율 동기화 실패로 거래 금액 오류 발생 | 실시간 환율 동기화 시스템 구축 |
| 모바일 앱 업데이트 충돌 | 보안 업데이트 후 거래 불가 상태 발생 | 회귀 테스트 도입, 업데이트 시뮬레이션 적용 |
| KYC 인증 미흡 | 고객 정보 업데이트 시 실시간 반영 실패 | 인증 프로세스 개선, 실시간 검증 도입 |
| 클라우드 마이그레이션 실패 | 레거시 시스템 이전 중 데이터 손실 | 클라우드 전환 시 단계적 마이그레이션 도입 |
| 대출 심사 중단 | 신용 정보 연동 오류로 심사 프로세스 중단 | 데이터 연동 최적화, 장애 포인트 개선 |
| 데이터 암호화 미흡 | 거래 내역이 평문으로 저장됨 | AES 암호화 도입, 데이터 보안 강화 |
| 전자 서명 오류 | 비대면 계약 시 전자 서명 실패 | 전자 서명 프로세스 개선, 재처리 구조 도입 |
| 통합 모니터링 부재 | 실시간 거래 내역 추적 불가 | 중앙 모니터링 시스템 도입, 실시간 추적 강화 |
| API Gateway 장애 | 핀테크 연동 시 통신 오류 발생 | Gateway 이중화 구성, 안정성 확보 |
| 중복 거래 처리 문제 | 동일 거래가 두 번 승인되는 문제 발생 | 트랜잭션 롤백 및 중복 탐지 강화 |
| 지연 결제 문제 | 야간 배치 시 결제 처리 지연 발생 | 배치 최적화, 병렬 처리 구조 도입 |
| 보안 감사 로그 미흡 | 거래 내역 추적 불가로 감사 부적합 | 중앙 감사 로그 시스템 도입, 보안성 향상 |

## 개선 방향

### 개요

* 중복 개발을 제거하고, 데이터 흐름을 최적화하며, 보안성과 확장성을 개선하는 방향으로 구조를 재정립합니다.
* 금융권 시스템의 고가용성과 실시간 거래 처리 성능을 극대화하기 위한 구조적 설계가 필요합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 모듈화 설계 | 중복된 기능을 모듈화하여 재사용성을 높임 | 고객 인증 모듈, 결제 처리 모듈 |
| 데이터 흐름 최적화 | 시스템 간 데이터 흐름을 최적화하여 처리 시간 단축 | 실시간 잔액 조회, 대출 승인 프로세스 |
| 보안성 향상 | OAuth2.0, JWT 등 인증 방식 적용 | 모바일 뱅킹, 전자결제 시스템 |
| 확장성 보장 | 클라우드 네이티브 구조로 유연한 확장 가능 | 금융 API Gateway, MSA 구조 |
| 모듈화 설계 | 중복된 기능을 모듈화하여 재사용성을 높임 | 고객 인증 모듈, 결제 처리 모듈 |
| 데이터 흐름 최적화 | 시스템 간 데이터 흐름을 최적화하여 처리 시간 단축 | 실시간 잔액 조회, 대출 승인 프로세스 |
| 보안성 향상 | OAuth2.0, JWT 등 인증 방식 적용 | 모바일 뱅킹, 전자결제 시스템 |
| 확장성 보장 | 클라우드 네이티브 구조로 유연한 확장 가능 | 금융 API Gateway, MSA 구조 |
| 장애 대응 체계 강화 | 이중화 구성 및 빠른 복구 체계 도입 | 무중단 거래 처리, 실시간 장애 탐지 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 개선 방향 |  |------------------------------------------------------|  | 1. 모듈화 설계 |  | - 중복 기능 모듈화 |  | - 재사용 가능 구조 체계 |  |------------------------------------------------------|  | 2. 데이터 흐름 최적화 |  | - 실시간 데이터 전송 최적화 |  | - 대량 트랜잭션 처리 개선 |  |------------------------------------------------------|  | 3. 보안성 향상 |  | - OAuth2.0 및 JWT 인증 도입 |  | - 보안 레이어 강화 |  |------------------------------------------------------|  | 4. 확장성 보장 |  | - 클라우드 네이티브 기반 확장 |  | - 마이크로서비스 구조 채택 |  |------------------------------------------------------|  | 5. 장애 대응 체계 강화 |  | - 이중화 구성 및 무중단 복구 |  | - 실시간 모니터링 체계 구축 |  +------------------------------------------------------+ |

### 실사례 (총 20개)

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 고객 인증 모듈화 | 중복된 인증 프로세스를 모듈화하여 재사용성 증대 | 인증 처리 시간 30% 단축, 유지보수 간소화 |
| 실시간 대출 승인 최적화 | 대출 심사 로직을 최적화하여 승인 시간을 단축 | 승인 시간 40% 감소, 고객 만족도 증가 |
| 모바일 결제 보안 강화 | OAuth2.0 인증 도입으로 보안성 확보 | 부정 거래 탐지율 20% 향상 |
| 클라우드 기반 확장성 개선 | 클라우드 네이티브로 전환하여 유연한 확장성 확보 | 시스템 증설 시간 50% 단축 |
| 무중단 거래 처리 | 이중화 시스템을 도입하여 실시간 장애 대응 | 거래 중단율 25% 감소, 복구 시간 단축 |
| 데이터 흐름 최적화 | 실시간 데이터 전송 최적화로 처리 시간 감소 | 트랜잭션 처리 속도 35% 향상 |
| 전자문서 처리 자동화 | 수작업 계약 프로세스 전자화 | 문서 처리 시간 60% 단축, 오류 감소 |
| 글로벌 송금 최적화 | 국제 송금 처리 속도 향상 및 비용 절감 | 송금 시간 40% 단축, 수수료 절감 |
| API Gateway 표준화 | 핀테크 연동을 위한 API 표준화 | 연결 시간 20% 감소, 오류율 감소 |
| 중앙 모니터링 시스템 도입 | 실시간 거래 추적 및 이력 관리 | 거래 오류 탐지 시간 30% 감소 |
| 클라우드 배치 최적화 | 배치 처리 시 클라우드 리소스 최적화 | 처리 시간 50% 단축, 비용 절감 |
| 비대면 KYC 프로세스 최적화 | 모바일 인증 프로세스 개선 | 인증 시간 40% 감소, 고객 이탈 감소 |
| 전자서명 최적화 | 계약 시 전자서명 프로세스 간소화 | 서명 처리 시간 30% 단축 |
| 실시간 리스크 분석 | 금융 리스크를 실시간으로 분석하여 대응 | 리스크 탐지 20% 향상 |
| 중복 거래 처리 방지 | 동일 거래의 중복 처리 방지 구조 도입 | 오류율 15% 감소 |
| 다단계 인증 강화 | 금융 거래 시 다단계 인증 도입 | 보안 위협 30% 감소 |
| 고객센터 통합 | 다중 채널 고객센터 시스템 통합 | 고객 응대 시간 20% 단축 |
| 통합 뱅킹 시스템 구축 | 분리된 시스템을 통합하여 관리 용이성 증대 | 유지보수 시간 30% 단축 |
| 대출 심사 자동화 | 심사 로직 최적화 및 데이터 연동 개선 | 처리 시간 25% 단축 |
| 거래 감사 로그 최적화 | 감사 추적 시스템 개선 | 규제 준수 40% 향상, 보안 강화 |
|  |  |  |

## 기대효과

### 개요

* 최적화된 응용 아키텍처 구조를 통해 금융 거래의 안정성과 확장성을 극대화합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 거래 처리 속도 향상 | 중복 개발 제거와 최적화된 데이터 흐름으로 빠른 거래 처리 | 실시간 송금, 대량 결제 처리 |
| 시스템 확장성 증대 | 클라우드 네이티브 기반 확장성 확보 | 마이크로서비스 구조 도입 |
| 보안성 강화 | 다단계 인증 및 실시간 보안 점검 | 온라인 뱅킹, 모바일 결제 시스템 |
| 중복 개발 방지 | 재사용 가능한 모듈화 설계 | 고객 정보 관리, 금융상품 조회 |
| 장애 대응 체계 개선 | 이중화 구성 및 무중단 복구 | 실시간 장애 탐지, 빠른 복구 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 기대효과 |  |------------------------------------------------------|  | 1. 거래 처리 속도 향상 |  | - 중복 로직 제거 및 최적화 |  | - 트랜잭션 처리 시간 단축 |  |------------------------------------------------------|  | 2. 시스템 확장성 증대 |  | - 클라우드 기반의 확장성 강화 |  | - MSA 구조로 확장 용이 |  |------------------------------------------------------|  | 3. 보안성 강화 |  | - 다단계 인증 체계 |  | - 실시간 보안 점검 |  |------------------------------------------------------|  | 4. 중복 개발 방지 |  | - 재사용 가능한 모듈 설계 |  | - 통합 관리 시스템 구축 |  |------------------------------------------------------|  | 5. 장애 대응 체계 개선 |  | - 이중화 구성 및 무중단 복구 |  | - 빠른 탐지 및 대응 |  +------------------------------------------------------+ |

## 1.6.4 실사례 (총 20개)

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 실시간 송금 최적화 | 중복 로직을 제거하고 최적화된 데이터 흐름 설계 | 처리 속도 25% 향상, 오류율 감소 |
| 클라우드 기반 확장 | 클라우드 네이티브 구조로 손쉬운 확장 | 서버 증설 시간 40% 단축, 비용 절감 |
| API Gateway 표준화 | 핀테크 연동 시 API 표준화로 통신 최적화 | 처리 시간 30% 단축, 오류 감소 |
| 모바일 뱅킹 보안 강화 | 다단계 인증 도입으로 보안성 극대화 | 해킹 시도 20% 감소, 사용자 신뢰도 상승 |
| 대규모 배치 최적화 | 야간 배치 처리 시간 단축 및 부하 감소 | 처리 시간 50% 단축, 리소스 사용 최적화 |
| 통합 거래 감사 시스템 | 거래 내역의 실시간 감사 로그 관리 | 추적 시간 35% 단축, 규제 대응 강화 |
| 전자서명 최적화 | 온라인 계약 시 전자서명 처리 시간 단축 | 서명 처리 시간 20% 감소, 문서 유실 방지 |
| 고객 360도 뷰 제공 | 고객 데이터를 통합 관리하여 응대 최적화 | 상담 시간 25% 단축, 고객 만족도 향상 |
| 실시간 리스크 분석 | 금융 리스크를 실시간으로 분석하여 대응 | 탐지 시간 20% 단축, 손실 방지 |
| 실시간 모니터링 도입 | 거래 추적 및 이벤트 탐지 최적화 | 오류 탐지 40% 향상, 대응 시간 단축 |
| 글로벌 송금 최적화 | 국제 송금 처리 속도 향상 및 비용 절감 | 송금 시간 30% 단축, 수수료 감소 |
| 거래 중복 처리 방지 | 동일 거래가 두 번 승인되는 문제 해결 | 오류율 15% 감소, 고객 불만 해소 |
| 클라우드 배치 최적화 | 클라우드 리소스를 최적화하여 배치 처리 개선 | 처리 시간 45% 단축, 비용 절감 |
| 오픈뱅킹 연동 최적화 | 다양한 금융사와의 연동 표준화 | 처리 시간 20% 감소, 오류율 감소 |
| 전자문서 시스템 최적화 | 수작업 문서 처리 대신 전자화된 관리 | 문서 처리 시간 60% 단축, 오류 감소 |
| 고객센터 시스템 통합 | 다중 채널 고객센터 시스템 통합 | 고객 응대 시간 20% 단축 |
| 대출 심사 자동화 | 심사 로직 최적화 및 데이터 연동 개선 | 처리 시간 25% 단축 |
| 비대면 KYC 인증 개선 | 인증 시간을 단축하고 보안성을 강화 | 처리 시간 30% 단축, 고객 이탈 감소 |
| 금융 데이터 암호화 최적화 | AES, RSA 기반 암호화 적용 | 데이터 유출 위험 35% 감소 |
| 실시간 거래 복구 체계 | 장애 시 실시간 복구 체계 도입 | 거래 중단 시간 40% 단축 |

## 마무리말

Application Architecture의 추진 배경과 개선 방향을 명확히 정의함으로써 시스템의 구조적 일관성을 확보할 수 있습니다. 금융 및 공공기관 프로젝트에서는 거래 속도 최적화, 데이터 무결성 보장, 확장성 확보가 필수적이며, 이를 위한 AA의 최적화가 프로젝트 성공을 좌우합니다.

최적화된 Application Architecture 설계를 통해 금융 시스템의 안정성과 확장성을 극대화할 수 있었습니다.  
금융 거래의 특성상 실시간 처리, 대량 트랜잭션, 보안 요구사항이 필수적이며, 이러한 요구를 만족하기 위한 아키텍처 최적화는 프로젝트 성공의 핵심 요인입니다.

모듈화된 구조와 표준화된 인터페이스, 클라우드 기반의 확장성은 프로젝트 유지보수 비용 절감과 확장성 증대에 중요한 역할을 했습니다.  
또한, 보안 강화를 통한 고객 신뢰도 확보와, 중복 개발 방지로 인한 생산성 증대는 금융권 프로젝트의 성공을 견인하였습니다.

특히, 클라우드 네이티브 아키텍처 도입을 통한 글로벌 확장과 API Gateway 표준화는 핀테크 연동을 최적화하였고,  
모바일 금융 서비스와 실시간 송금 처리의 성능을 대폭 개선하였습니다.

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 글로벌 결제 플랫폼 확장 | 클라우드 네이티브 구조로 해외 결제 서비스 확장 | 결제 시간 30% 단축, 비용 절감 |
| 실시간 결제 시스템 최적화 | 실시간 거래 처리 속도를 극대화 | 거래 속도 25% 향상, 오류율 감소 |
| 전자금융 보안 강화 | 모바일 뱅킹과 인터넷 뱅킹 보안 체계 개선 | 해킹 시도 20% 감소, 보안 사고 예방 |
| 클라우드 배치 최적화 | 클라우드 리소스를 최적화하여 배치 처리 개선 | 처리 시간 50% 단축, 비용 절감 |
| 오픈뱅킹 API 확장 | 다양한 핀테크 서비스와의 연동 최적화 | 연동 시간 20% 감소, 오류율 감소 |
| 실시간 리스크 분석 시스템 도입 | 거래 리스크를 실시간 분석하여 대응 | 탐지 시간 20% 단축, 손실 방지 |
| KYC 인증 최적화 | 비대면 고객 인증 프로세스 개선 | 인증 시간 30% 단축, 고객 이탈 감소 |
| 글로벌 송금 최적화 | 해외 송금 처리 최적화 및 비용 절감 | 송금 시간 40% 단축, 수수료 감소 |
| API Gateway 표준화 | API 연동 시 통신 속도 최적화 | 통신 시간 20% 단축, 오류 감소 |
| 전자서명 최적화 | 계약 시 전자서명 처리 시간 단축 | 서명 처리 시간 20% 감소, 문서 유실 방지 |
| 통합 거래 감사 시스템 | 실시간 감사 로그 관리 | 추적 시간 35% 단축, 규제 대응 강화 |
| 고객 360도 뷰 제공 | 고객 정보를 통합 관리하여 응대 최적화 | 상담 시간 25% 단축, 고객 만족도 향상 |
| 금융 데이터 암호화 최적화 | AES, RSA 기반 암호화 적용 | 데이터 유출 위험 35% 감소 |
| 실시간 거래 복구 체계 | 장애 시 실시간 복구 체계 도입 | 거래 중단 시간 40% 단축 |
| 모바일 뱅킹 최적화 | 앱 성능 개선 및 보안 강화 | 사용자 만족도 30% 향상 |
| 중앙 모니터링 시스템 구축 | 실시간 거래 현황 추적 및 알림 | 탐지 시간 20% 단축, 장애 대응 신속화 |
| 클라우드 마이그레이션 최적화 | 레거시 시스템을 클라우드로 전환 | 유지보수 시간 40% 단축, 비용 절감 |
| 고객센터 시스템 통합 | 다중 채널 고객 응대 시스템 통합 | 응대 시간 20% 단축 |
| 데이터 중복 처리 방지 | 동일 거래의 중복 승인 방지 | 오류율 15% 감소, 고객 불만 해소 |
| 통합 대시보드 구축 | 실시간 거래 현황을 시각적으로 표현 | 관리 효율성 30% 향상 |

## 시사점

* 금융권 및 공공기관 프로젝트에서의 Application Architecture 최적화는 안정성, 확장성, 보안성의 근간을 이루며, 시스템 전반의 품질을 높이는 중요한 역할을 수행하였습니다.
* 특히 대규모 트랜잭션 처리, 실시간 데이터 동기화, API 연동 최적화는 금융 거래의 신뢰성을 극대화하였으며, 이를 통해 고객 신뢰도와 시장 경쟁력을 확보할 수 있었습니다.

|  |  |  |
| --- | --- | --- |
| 시사점 | 설명 | 실사례 |
| 구조적 설계 최적화 | 명확한 기능 체계 수립으로 중복 개발 방지와 확장성 극대화 | 글로벌 결제 플랫폼 확장, API Gateway 표준화 |
| 보안과 무결성 강화 | 금융거래 특성에 맞춘 보안 구조 설계로 신뢰성 있는 시스템 구축 | 전자금융 보안 강화, 모바일 뱅킹 최적화 |
| 실시간 처리 및 복구 | 금융 트랜잭션 실패 시 실시간 복구 및 추적을 통해 안정성 확보 | 실시간 거래 복구 체계, 클라우드 배치 최적화 |
| 데이터 최적화 | 중복된 데이터 처리 방지 및 실시간 동기화 보장 | 데이터 중복 처리 방지, 금융 데이터 암호화 |
| 클라우드 확장성 확보 | 클라우드 네이티브 구조로 글로벌 확장 가능 | 클라우드 마이그레이션 최적화, 글로벌 송금 최적화 |
| 통합 관리 체계 구축 | 중앙 집중형 모니터링과 통합 대시보드 도입 | 중앙 모니터링 시스템 구축, 통합 거래 감사 시스템 |
| 고객 중심 서비스 최적화 | 고객의 요구에 맞춘 빠르고 안전한 금융 서비스 제공 | 고객 360도 뷰 제공, 고객센터 시스템 통합 |

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 시사점 |  |------------------------------------------------------|  | 1. 구조적 설계 최적화 |  | - 중복 로직 제거 및 표준화 |  | - 확장 가능한 아키텍처 구성 |  |------------------------------------------------------|  | 2. 보안과 무결성 강화 |  | - 다단계 인증 및 실시간 보안 점검 |  | - 금융 데이터 암호화 강화 |  |------------------------------------------------------|  | 3. 실시간 처리 및 복구 |  | - 클라우드 기반의 실시간 복구 체계 |  | - 트랜잭션 실패 시 즉각적인 복구 |  |------------------------------------------------------|  | 4. 데이터 최적화 |  | - 중복 데이터 방지 및 최적화 |  | - 실시간 동기화 보장 |  |------------------------------------------------------|  | 5. 클라우드 확장성 확보 |  | - 글로벌 확장을 위한 네이티브 구조 |  | - 신속한 시스템 확장 및 배포 |  |------------------------------------------------------|  | 6. 통합 관리 체계 구축 |  | - 중앙 모니터링 및 대시보드 |  | - 실시간 거래 추적 및 이벤트 탐지 |  +------------------------------------------------------+ |

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 글로벌 결제 플랫폼 확장 | 클라우드 네이티브 구조로 해외 결제 서비스 확장 | 결제 시간 30% 단축, 비용 절감 |
| API Gateway 표준화 | 다양한 핀테크 서비스와의 연동 최적화 | 통신 속도 20% 향상, 오류율 감소 |
| 실시간 거래 복구 체계 | 장애 시 실시간 복구 체계 도입 | 거래 중단 시간 40% 단축 |
| 클라우드 배치 최적화 | 클라우드 리소스를 최적화하여 배치 처리 개선 | 처리 시간 50% 단축, 비용 절감 |
| 모바일 뱅킹 보안 강화 | 다단계 인증 도입으로 보안성 극대화 | 해킹 시도 20% 감소, 보안 사고 예방 |
| 통합 거래 감사 시스템 | 실시간 감사 로그 관리 | 추적 시간 35% 단축, 규제 대응 강화 |
| 고객 360도 뷰 제공 | 고객 정보를 통합 관리하여 응대 최적화 | 상담 시간 25% 단축, 고객 만족도 향상 |
| 금융 데이터 암호화 최적화 | AES, RSA 기반 암호화 적용 | 데이터 유출 위험 35% 감소 |
| 글로벌 송금 최적화 | 해외 송금 처리 최적화 및 비용 절감 | 송금 시간 40% 단축, 수수료 감소 |
| 중앙 모니터링 시스템 구축 | 실시간 거래 현황 추적 및 알림 | 탐지 시간 20% 단축, 장애 대응 신속화 |
| 클라우드 마이그레이션 최적화 | 레거시 시스템을 클라우드로 전환 | 유지보수 시간 40% 단축, 비용 절감 |
| 고객센터 시스템 통합 | 다중 채널 고객 응대 시스템 통합 | 응대 시간 20% 단축 |
| 데이터 중복 처리 방지 | 동일 거래의 중복 승인 방지 | 오류율 15% 감소, 고객 불만 해소 |
| 통합 대시보드 구축 | 실시간 거래 현황을 시각적으로 표현 | 관리 효율성 30% 향상 |
| 전자서명 최적화 | 계약 시 전자서명 처리 시간 단축 | 서명 처리 시간 20% 감소, 문서 유실 방지 |
| 비대면 KYC 인증 개선 | 인증 시간을 단축하고 보안성을 강화 | 처리 시간 30% 단축, 고객 이탈 감소 |
| 전자문서 시스템 최적화 | 수작업 문서 처리 대신 전자화된 관리 | 문서 처리 시간 60% 단축, 오류 감소 |
| 실시간 리스크 분석 | 금융 리스크를 실시간으로 분석하여 대응 | 탐지 시간 20% 단축, 손실 방지 |
| 오픈뱅킹 API 확장 | 다양한 금융사와의 연동 표준화 | 연동 시간 20% 감소, 오류율 감소 |
| 다단계 인증 최적화 | 금융 거래 시 다단계 인증 도입 | 보안 위협 30% 감소 |

# 주요 수행 업무 정의

## 도입 전 안내말

Application Architecture(AA) 수행 방안에서 주요 수행 업무는 금융권 및 공공기관 프로젝트의 성공적인 구축을 위한 핵심 요소입니다. 특히 금융 시스템의 경우 고가용성(High Availability), 무결성(Integrity), 확장성(Scalability)을 필수적으로 고려해야 하며, 이를 실현하기 위한 세부 업무 정의가 필요합니다.

본 장에서는 주요 수행 업무를 세부 목차로 나누어 설명하고, 각 업무의 중요성, 구조적 설계, 최적화 전략을 체계적으로 정리합니다.

Application Architecture(AA) 수행 방안에서 주요 수행 업무는 금융권 및 공공기관 프로젝트의 성공적인 구축을 위한 핵심 요소입니다.

* 특히 금융 시스템의 경우 고가용성(High Availability), 무결성(Integrity), 확장성(Scalability)을 필수적으로 고려해야 하며, 이를 실현하기 위한 세부 업무 정의가 필요합니다.
* 본 장에서는 주요 수행 업무를 세부 목차로 나누어 설명하고, 각 업무의 중요성, 구조적 설계, 최적화 전략을 체계적으로 정리합니다.

## 주요 수행 업무 정의 요약

|  |  |
| --- | --- |
| 구분 | 설명 |
| 요구사항 분석 | 비즈니스 요구사항을 분석하여 시스템 기능을 정의하고 최적화된 구조를 설계합니다. |
| 아키텍처 설계 | 응용 시스템의 계층적 구조를 정의하고, 각 계층 간의 인터페이스를 최적화합니다. |
| 데이터 설계 | 데이터 흐름 최적화와 무결성 보장을 위한 데이터 구조를 설계합니다. |
| 인터페이스 정의 | 내부 시스템 및 외부 시스템 간의 연동 규격을 정의하고 표준화합니다. |
| 성능 최적화 | 트랜잭션 처리 속도 개선과 시스템 확장성을 극대화하는 구조로 설계합니다. |
| 보안 설계 | 금융거래의 안전성을 보장하기 위한 보안 정책 및 인증 체계를 정의합니다. |
| 테스트 및 검증 | 설계된 아키텍처가 요구사항을 만족하는지 검증하고, 성능 최적화를 진행합니다. |

## 요구사항 분석

### 개요

* 금융 거래에서의 요구사항 분석은 시스템 안정성과 확장성을 보장하기 위한 기초 작업입니다.
* 정확한 요구사항 정의는 프로젝트 전 단계의 설계 및 개발에 반영되며, 중복 개발 방지와 데이터 무결성 확보에 기여합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 기능 요구사항 분석 | 사용자 요구를 바탕으로 서비스 기능 정의 | 인터넷 뱅킹, 모바일 결제 시스템 |
| 비기능 요구사항 분석 | 성능, 보안, 확장성 요구 분석 | 대량 트랜잭션 처리, 보안 인증 강화 |
| 요구사항 추적성 매트릭스 | 요구사항 변경 시 추적할 수 있도록 매핑 | 온라인 결제 시스템, 대출 심사 구조 |
| 요구사항 변경 관리 | 프로젝트 진행 중 발생하는 요구 변경 사항 관리 | 신상품 추가, 규제 변경 대응 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 요구사항 분석 |  |------------------------------------------------------|  | 1. 기능 요구사항 분석 |  | - 사용자 요구 기반 서비스 정의 |  | - 온라인 거래, 모바일 결제 시스템 최적화 |  |------------------------------------------------------|  | 2. 비기능 요구사항 분석 |  | - 성능, 보안, 확장성 요구 정의 |  | - 대량 트랜잭션 처리 최적화 |  |------------------------------------------------------|  | 3. 요구사항 추적성 매트릭스 |  | - 요구사항 변경 시 영향 분석 및 추적 |  | - 시스템 간 데이터 흐름 추적 |  |------------------------------------------------------|  | 4. 요구사항 변경 관리 |  | - 신상품 추가, 정책 변경 시 변경사항 반영 |  | - 빠른 변경 대응 및 리스크 최소화 |  +------------------------------------------------------+ |

### 실사례 (총 20개)

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| 사용자 인증 설계 | OTP, MFA, 생체 인증 등 다단계 인증 체계 설계 | 모바일 뱅킹, 인터넷 뱅킹 로그인 | ① OTP 생성② 인증 실패 시 잠금③ 이중 인증 처리④ 생체 인증 지원⑤ 인증 시도 횟수 제한⑥ 임시 비밀번호 발급⑦ 다중 디바이스 인증 동기화⑧ 분산 세션 관리⑨ 세션 Hijacking 방지⑩ 사용자 인증 로그 추적 |
| 트랜잭션 처리 설계 | 대량 거래 시 중복 방지 및 처리 속도 최적화 | 실시간 송금, 대량 이체 시스템 | ① 트랜잭션 롤백② 중복 검사③ 트랜잭션 큐 처리④ 일관성 유지 처리⑤ 분산 트랜잭션 관리⑥ 트랜잭션 지연 감지⑦ 트랜잭션 재시도 로직⑧ Lock 관리⑨ 동시성 처리⑩ 트랜잭션 커밋 최적화 |
| 데이터 일관성 관리 | 실시간 거래 처리 중 데이터 무결성 보장 | 정산 시스템, 회계 처리 시스템 | ① ACID 준수② 데이터 정합성 검증③ 데이터 복구 전략④ 실시간 동기화⑤ CDC(Change Data Capture)⑥ 트랜잭션 격리 수준 관리⑦ MDM(Master Data Management)⑧ 데이터 중복 처리 방지⑨ 잠금 해제 최적화⑩ 무결성 제약 조건 관리 |
| 인터페이스 표준화 | 내부 및 외부 API 통신 규격 표준화 | 오픈뱅킹 API, 핀테크 연동 | ① REST API 설계② SOAP API 설계③ JSON/XML 표준화④ API Rate Limiting⑤ API Key 관리⑥ OAuth2.0 인증⑦ 메시지 포맷 표준화⑧ API 버전 관리⑨ CORS 정책 설정⑩ API 게이트웨이 라우팅 |
| 로깅 및 모니터링 설계 | 거래 이력 추적 및 오류 감지 로깅 | 거래 감사 로그, 거래 내역 모니터링 | ① 로그 집계② 실시간 모니터링③ 이벤트 추적④ 오류 탐지 및 알림⑤ 보안 이벤트 로그 수집⑥ 트랜잭션 이력 관리⑦ 시계열 데이터 분석⑧ 사용자 활동 로그⑨ 장애 탐지 자동화⑩ 로그 백업 및 보관 |
| 비동기 처리 구조 설계 | 비동기 이벤트 처리로 거래 속도 최적화 | 카드 승인, 자동 이체 처리 | ① 메시지 큐 처리② 비동기 작업 관리③ 이벤트 드리븐 처리④ 이벤트 재처리 로직⑤ 비동기 오류 감지⑥ 비동기 모니터링⑦ 트랜잭션 순서 보장⑧ 이벤트 누락 방지⑨ 메시지 중복 처리⑩ 메시지 TTL 설정 |
| 배치 처리 구조 설계 | 야간 대량 배치 처리 최적화 | 금융 거래 정산, 고객 포인트 배치 처리 | ① 스케줄링② 트랜잭션 처리③ 로그 수집④ 장애 발생 시 재처리⑤ 대량 데이터 처리 최적화⑥ 배치 처리 순서 보장⑦ 실패 내역 추적⑧ 배치 병렬 처리⑨ 리소스 최적화⑩ 성능 모니터링 |
| 장애 복구 체계 설계 | 거래 실패 시 롤백 및 복구 구조 설계 | 송금 오류 복구, 대출 승인 복구 | ① 데이터 복원② 재처리 기능③ 이중화 구성④ Hot Backup 지원⑤ 장애 탐지 자동화⑥ 복구 시간 최적화⑦ 트랜잭션 롤백⑧ 복구 시 데이터 정합성 검증⑨ RPO, RTO 관리⑩ 테스트 복구 계획 |

|  |  |  |
| --- | --- | --- |
| 사례명 | 주요 내용 | 주요 개선 효과 |
| 대출 심사 최적화 | 대출 승인 과정에서의 요구사항 분석을 체계화 | 처리 시간 30% 단축, 오류율 감소 |
| 글로벌 결제 시스템 요구사항 표준화 | 국제 송금 시 중복된 규격을 표준화 | 비용 절감 20%, 처리 속도 향상 |
| 실시간 결제 처리 최적화 | 온라인 결제 시 요구사항 분석을 통한 처리 최적화 | 처리 시간 25% 감소, 오류 탐지 강화 |
| 모바일 인증 요구사항 개선 | 모바일 환경에 맞춘 요구사항 분석 | 인증 시간 30% 단축, 사용자 편의성 증대 |
| KYC 요구사항 관리 최적화 | 비대면 인증 시 요구사항 정의 최적화 | 처리 시간 40% 단축, 오류율 감소 |
| 통합 고객 정보 요구사항 정의 | 여러 시스템에 분산된 고객 정보를 통합 관리 | 조회 시간 20% 단축, 중복 데이터 감소 |
| API 요구사항 표준화 | 다양한 핀테크 서비스와의 연동 요구사항 표준화 | 처리 시간 15% 단축, 호환성 증대 |
| 클라우드 마이그레이션 요구사항 정립 | 레거시 시스템 이전 시 요구사항 최적화 | 전환 시간 30% 단축, 데이터 손실 방지 |
| 온라인 뱅킹 요구사항 분석 | 사용자 행동 패턴 분석을 통해 요구사항 정립 | UX 향상, 전환율 증가 |
| 보안 요구사항 최적화 | 다단계 인증 및 실시간 보안 점검 요구 정의 | 해킹 시도 25% 감소 |

## 아키텍처 설계

### 개요

* 금융 서비스에서 아키텍처 설계는 시스템의 구조적 일관성을 유지하고, 확장성을 극대화하기 위한 전략적 단계입니다. 계층적 구조와 인터페이스 정의를 통해 복잡한 업무 로직을 단순화합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 계층 구조 설계 | Presentation, Business, Data Layer로 분리 | 인터넷 뱅킹, 전자금융 거래 |
| 인터페이스 설계 | 내부 시스템 및 외부 API 연동 최적화 | 오픈 뱅킹, 핀테크 연계 |
| 데이터 흐름 최적화 | 복잡한 데이터 흐름을 단순화하고 최적화 | 실시간 거래 처리, 대량 이체 |
| MSA 구조 도입 | 마이크로서비스 기반 분리 설계 | 글로벌 송금, 대출 관리 시스템 |
| 비동기 통신 구조 | 이벤트 기반의 비동기 통신 최적화 | 카드 결제 승인, 예약 결제 시스템 |
| 통합 모니터링 설계 | 중앙 집중형 모니터링 체계 구축 | 거래 내역 모니터링, 오류 탐지 |
| 분산 처리 구조 | 대량 데이터 및 트랜잭션의 분산 처리 | 결제 처리 분산, 로드 밸런싱 |

### 구성도

+------------------------------------------------------+

| Application Architecture 설계 구조 |

|------------------------------------------------------|

| 1. 계층 구조 설계 |

| - Presentation Layer: UI 및 사용자 인터페이스 |

| - Business Layer: 비즈니스 로직 처리 |

| - Data Layer: 데이터 처리 및 저장 |

|------------------------------------------------------|

| 2. 인터페이스 설계 |

| - 내부 시스템 간 API 연동 최적화 |

| - 외부 핀테크 연동을 위한 표준 API 설계 |

|------------------------------------------------------|

| 3. 데이터 흐름 최적화 |

| - 실시간 데이터 전송 |

| - 트랜잭션 처리 시간 단축 |

|------------------------------------------------------|

| 4. MSA 구조 도입 |

| - 마이크로서비스 기반 모듈화 |

| - 확장성과 독립적 배포 가능 |

+------------------------------------------------------+

### 실사례 (설계 항목 예시 30개)

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| **계층 구조 설계** | Presentation, Business, Data Layer로 분리하여 관리 | 인터넷 뱅킹, 전자금융 거래 | ① UI Layer 분리② Service Layer 최적화③ DAO Layer 설계④ Controller-View 분리⑤ Session 관리 최적화⑥ Load Balancing 적용⑦ Cross-Cutting Concern 분리⑧ Business Logic 재사용성 보장⑨ 데이터 액세스 규격 정의⑩ 에러 핸들링 구조 설계 |
| **인터페이스 설계** | 내부 시스템 및 외부 API 연동 최적화 | 오픈 뱅킹, 핀테크 연계 | ① REST API 설계② SOAP API 설계③ JSON/XML 표준화④ API Rate Limiting⑤ API Key 관리⑥ OAuth2.0 인증⑦ 메시지 포맷 표준화⑧ API 버전 관리⑨ CORS 정책 설정⑩ API 게이트웨이 라우팅 |
| **데이터 흐름 최적화** | 복잡한 데이터 흐름을 단순화하고 최적화 | 실시간 거래 처리, 대량 이체 | ① 실시간 데이터 스트림 처리② 비동기 데이터 전송③ 큐 기반 처리 최적화④ 대량 데이터 배치 처리⑤ 분산 처리 구성⑥ 동시성 관리 최적화⑦ 트랜잭션 격리 수준 관리⑧ 데이터 유실 방지 메커니즘⑨ 데이터 일관성 검증⑩ 멀티스레드 최적화 |
| **MSA 구조 설계** | 마이크로서비스 기반 분리 설계 | 글로벌 송금, 대출 관리 시스템 | ① 마이크로서비스 단위 분리② 서비스 간 독립적 배포③ 데이터베이스 분리 관리④ 서비스 디스커버리⑤ API Gateway 통합⑥ 분산 캐시 처리⑦ 서비스 메시지 처리⑧ 무중단 배포 지원⑨ 서비스 간 트랜잭션 관리⑩ 장애 격리 구조 설계 |
| **비동기 통신 구조** | 이벤트 기반의 비동기 통신 최적화 | 카드 결제 승인, 예약 결제 시스템 | ① 메시지 큐 사용② 비동기 이벤트 처리③ Publish/Subscribe 패턴④ 이벤트 재처리 메커니즘⑤ 메시지 순서 보장⑥ 이벤트 누락 방지⑦ 비동기 에러 핸들링⑧ 메시지 TTL 설정⑨ 지연 처리 최적화⑩ 오프라인 큐 처리 |
| **통합 모니터링 설계** | 중앙 집중형 모니터링 체계 구축 | 거래 내역 모니터링, 오류 탐지 | ① 실시간 트랜잭션 모니터링② 에러 로그 추적③ 대시보드 시각화④ API 응답 시간 체크⑤ 보안 이벤트 탐지⑥ 시스템 리소스 사용량 모니터링⑦ 장애 탐지 및 알림⑧ 성능 병목 분석⑨ 실시간 로그 집계⑩ SLA 준수 모니터링 |
| **분산 처리 구조 설계** | 대량 데이터 및 트랜잭션의 분산 처리 | 결제 처리 분산, 로드 밸런싱 | ① Sharding 기반 데이터 분산② 분산 Lock 관리③ 글로벌 트랜잭션 분리④ 멀티 리전 처리 최적화⑤ 로드 밸런서 구성⑥ 세션 일관성 보장⑦ 분산 캐시 최적화⑧ Failover 지원⑨ 데이터 복제 최적화⑩ 이중화 처리 구조 |
| **API Gateway 설계** | 외부 시스템과의 안전한 연동 최적화 | 핀테크 연동, 외부 결제 API | ① 요청 인증 처리② Rate Limiting 설정③ Circuit Breaker 구성④ 로드 밸런싱 설정⑤ JWT 기반 인증⑥ 트래픽 분석 및 모니터링⑦ 요청 리트라이 처리⑧ API 라우팅 최적화⑨ 보안 정책 적용⑩ API 버전 관리 |
| **클라우드 네이티브 구조 설계** | 클라우드 기반 확장성 보장 | 마이크로서비스, 글로벌 확장 | ① 오토스케일링② 컨테이너 오케스트레이션③ 로드 밸런싱 설정④ 클라우드 네트워크 격리⑤ 멀티 리전 배포⑥ CI/CD 자동화 파이프라인⑦ 클라우드 모니터링⑧ 장애 복구 자동화⑨ 비용 최적화 관리⑩ 하이브리드 클라우드 지원 |

## 데이터 설계

### 개요

* 데이터 설계는 금융 거래의 무결성과 성능을 보장하기 위한 필수 단계입니다. 스키마 설계, 인덱스 최적화, 데이터 중복 방지 전략이 포함됩니다.
* 금융 시스템에서의 데이터 설계는 트랜잭션 처리의 신속성, 무결성, 보안성을 보장하기 위한 핵심 단계입니다.
* 특히 금융 데이터는 실시간 거래, 배치 처리, 대용량 이체 등이 빈번하므로 최적화된 스키마 설계와 데이터 흐름 관리가 필수적입니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 데이터 모델링 설계 | 논리적/물리적 데이터 모델 정의 및 최적화 | 고객 정보 DB, 거래 내역 DB |
| 데이터 정규화 | 중복 최소화 및 데이터 일관성 유지 | 고객 정보, 계좌 이력 관리 |
| 인덱스 최적화 | 대량 조회 시 성능 극대화 | 실시간 거래 조회, 대량 이체 처리 |
| 파티셔닝 전략 | 데이터 분산 저장 및 성능 향상 | 고객 데이터 파티셔닝, 로그 데이터 분리 |
| 데이터 암호화 설계 | 민감한 금융 정보를 암호화하여 보안 강화 | 계좌 번호, 카드 정보 보호 |
| CDC(Change Data Capture) | 실시간 데이터 변경 추적 및 반영 | 실시간 거래 처리, 상태 변경 추적 |
| 백업 및 복구 설계 | 장애 발생 시 데이터 손실 방지 | 금융 거래 이력 복구, 일일 정산 내역 복구 |
| 데이터 아카이빙 | 장기 보관 데이터를 안전하게 관리 | 거래 내역 보존, 감사 로그 보관 |
| 데이터 무결성 검증 | 트랜잭션 처리 중 데이터 손실 및 오류 방지 | 실시간 송금, 다중 계좌 이체 |
| ETL 프로세스 설계 | 대량 데이터의 변환, 로딩 최적화 | 일별 정산 배치, 고객 정보 동기화 |
| 항목 | **설명** | **적용 사례** |
| 데이터 스키마 설계 | 최적화된 데이터 모델 정의 | 고객 정보 DB, 거래 이력 DB |
| 인덱스 최적화 | 조회 속도를 높이기 위한 인덱스 설계 | 대량 데이터 조회, 통계 분석 |
| 데이터 중복 방지 | 중복 데이터 제거 및 일관성 유지 | 고객 정보 통합, 다중 계좌 관리 |
| 데이터 암호화 | 민감 정보의 보안 강화를 위한 암호화 적용 | 계좌 번호, 카드 정보 보호 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 데이터 설계 구조 |  |------------------------------------------------------|  | 1. 데이터 모델링 설계 |  | - 논리적/물리적 모델 정의 |  | - 중복 최소화 및 최적화 |  |------------------------------------------------------|  | 2. 인덱스 최적화 |  | - 빠른 조회 속도 |  | - 대량 거래 처리 최적화 |  |------------------------------------------------------|  | 3. 파티셔닝 전략 |  | - 데이터 분리 저장 및 관리 |  | - 성능 최적화 및 확장성 보장 |  |------------------------------------------------------|  | 4. 데이터 암호화 설계 |  | - AES, RSA 암호화 적용 |  | - 민감 정보 보호 |  |------------------------------------------------------|  | 5. CDC(Change Data Capture) |  | - 실시간 변경 데이터 추적 |  | - 트랜잭션 일관성 유지 |  |------------------------------------------------------|  | 6. 백업 및 복구 설계 |  | - 정기 백업 및 장애 복구 |  | - 데이터 손실 방지 |  |------------------------------------------------------|  | 7. 데이터 아카이빙 |  | - 장기 보관 데이터 관리 |  | - 감사 및 규제 대응 |  |------------------------------------------------------|  | 8. 데이터 무결성 검증 |  | - 트랜잭션 오류 방지 |  | - 이중 승인 및 검증 |  |------------------------------------------------------|  | 9. ETL 프로세스 설계 |  | - 데이터 변환 및 적재 최적화 |  | - 대용량 배치 처리 최적화 |  +------------------------------------------------------+ |

+------------------------------------------------------+

| Application Architecture 데이터 설계 구조 |

|------------------------------------------------------|

| 1. 데이터 스키마 설계 |

| - 최적화된 테이블 구조 정의 |

| - 고객 정보, 거래 이력 저장 |

|------------------------------------------------------|

| 2. 인덱스 최적화 |

| - 조회 속도 향상 |

| - 대량 데이터 빠른 탐색 |

|------------------------------------------------------|

| 3. 데이터 중복 방지 |

| - 데이터 일관성 보장 |

| - 중복 데이터 제거 및 통합 관리 |

|------------------------------------------------------|

| 4. 데이터 암호화 |

| - 민감 정보 보호를 위한 암호화 |

| - 계좌 정보, 카드 정보 암호화 처리 |

+------------------------------------------------------+

### 실사례

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| 데이터 모델링 설계 | 논리적/물리적 데이터 모델 정의 및 최적화 | 고객 정보 DB, 거래 내역 DB | ① 엔티티 정의② 속성 매핑③ 관계 설정④ 식별자 관리⑤ 정규화⑥ 비정규화 최적화⑦ 인덱스 설정⑧ 트리거 설정⑨ 외래키 제약 조건 설정⑩ 변경 추적 관리 |
| 데이터 정규화 | 중복 최소화 및 데이터 일관성 유지 | 고객 정보, 계좌 이력 관리 | ① 1NF, 2NF, 3NF 정규화② 중복 데이터 제거③ 테이블 분리 최적화④ 종속성 분석⑤ 참조 무결성 설정⑥ 역정규화 최적화⑦ PK/FK 설정⑧ 조인 최적화⑨ 컬럼 재사용 최적화⑩ 데이터 중복 방지 |
| 인덱스 최적화 | 대량 조회 시 성능 극대화 | 실시간 거래 조회, 대량 이체 처리 | ① B-Tree 인덱스 생성② 해시 인덱스 적용③ 복합 인덱스 설정④ 부분 인덱스 관리⑤ 클러스터드 인덱스 최적화⑥ 비정렬 검색 최적화⑦ 인덱스 리빌드 전략⑧ 중복 인덱스 탐지⑨ 히스토그램 관리⑩ 쿼리 분석 최적화 |
| 파티셔닝 전략 | 데이터 분산 저장 및 성능 향상 | 고객 데이터 파티셔닝, 로그 데이터 분리 | ① Range Partitioning② List Partitioning③ Hash Partitioning④ Composite Partitioning⑤ 파티션별 인덱스 최적화⑥ Partition Pruning⑦ 파티션 메타데이터 관리⑧ 파티션 증설 최적화⑨ 온라인 파티션 분리⑩ 파티션 백업 및 복구 |
| 데이터 암호화 설계 | 민감한 금융 정보를 암호화하여 보안 강화 | 계좌 번호, 카드 정보 보호 | ① AES 암호화 적용② RSA 암호화 적용③ 데이터 복호화 관리④ 키 관리 시스템 연동⑤ 전송 구간 암호화(TLS/SSL)⑥ 정적 데이터 암호화⑦ 필드 레벨 암호화⑧ 접근 통제 설정⑨ 보안 로그 추적⑩ 보안 인증 연동 |
| CDC(Change Data Capture) | 실시간 데이터 변경 추적 및 반영 | 실시간 거래 처리, 상태 변경 추적 | ① 데이터 변경 감지② 트리거 기반 변경 추적③ 실시간 스트림 처리④ 변경 이력 관리⑤ 변경 충돌 처리⑥ 분산 CDC 구성⑦ 트랜잭션 일관성 유지⑧ 다중 데이터 소스 동기화⑨ 이벤트 기반 처리⑩ 데이터 복제 최적화 |
| 백업 및 복구 설계 | 장애 발생 시 데이터 손실 방지 | 금융 거래 이력 복구, 일일 정산 내역 복구 | ① 정기 백업 스케줄링② 증분 백업 최적화③ 핫 백업 및 콜드 백업 설정④ 복구 시점 설정 (PITR)⑤ 복구 테스트 계획⑥ 백업 무결성 검증⑦ 장애 발생 시 복구 시나리오⑧ 오프사이트 백업 관리⑨ 데이터 스냅샷 저장⑩ 백업 이력 관리 |
| 데이터 아카이빙 | 장기 보관 데이터를 안전하게 관리 | 거래 내역 보존, 감사 로그 보관 | ① 장기 보관 규칙 설정② 데이터 압축 저장③ 암호화된 아카이빙④ 데이터 접근 제한 설정⑤ 규정 준수 기록 관리⑥ 검색 최적화 설정⑦ 읽기 전용 아카이브 설정⑧ 데이터 수명 주기 관리⑨ 실시간 조회 최적화⑩ 감사 이력 보관 |
| 데이터 무결성 검증 | 트랜잭션 처리 중 데이터 손실 및 오류 방지 | 실시간 송금, 다중 계좌 이체 | ① ACID 속성 보장② 참조 무결성 유지③ 중복 데이터 방지④ 동시성 제어⑤ 데이터 잠금 관리⑥ 유효성 검사 규칙 설정⑦ 데이터 일관성 검증⑧ 무결성 제약 조건 적용⑨ 복구 시 데이터 검증⑩ 에러 탐지 및 알림 |
| ETL 프로세스 설계 | 대량 데이터의 변환, 로딩 최적화 | 일별 정산 배치, 고객 정보 동기화 | ① 데이터 추출 최적화② 데이터 변환 규칙 설정③ 다중 소스 데이터 통합④ 스케줄링 최적화⑤ 오류 처리 및 재시도⑥ 중복 데이터 처리⑦ 배치 처리 최적화⑧ 로깅 및 모니터링⑨ 데이터 로딩 속도 최적화⑩ ETL 자동화 관리 |

## 인터페이스 정의

### 개요

* 금융 시스템 간의 원활한 데이터 송수신을 보장하기 위해 인터페이스를 표준화하고 최적화합니다.
* 금융권 시스템은 다양한 외부 서비스와의 연동이 빈번하기 때문에 명확한 인터페이스 정의가 필수적입니다.
* 금융 시스템에서의 인터페이스 정의는 내부 시스템 및 외부 연계 시스템 간의 원활한 데이터 통신을 보장하기 위한 구조적 설계입니다.
* REST API, SOAP API, 메시지 큐(MQ), 파일 연동 방식 등 다양한 인터페이스가 존재하며, 각각의 표준을 준수하여 개발됩니다.
* 특히 금융거래의 특성상 **보안, 일관성, 무결성**이 보장되어야 하며, **호환성**과 **확장성**이 고려된 설계가 필요합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| REST API 설계 | Representational State Transfer 방식의 API 표준 설계 | 모바일 뱅킹, 인터넷 뱅킹 |
| SOAP API 설계 | 금융기관 간 데이터 연동에 적합한 표준화된 API 설계 | 대외계 금융 시스템 연동 |
| 메시지 큐(MQ) 연동 | 비동기 통신을 위한 메시지 큐 시스템 연동 | 실시간 결제 승인, 대량 이체 |
| 파일 연동 설계 | 시스템 간 대용량 파일 데이터 교환 | 일별 정산 파일, 보고서 전송 |
| 실시간 인터페이스 설계 | 실시간 데이터 반영을 위한 연동 구조 | 카드 승인, 송금 처리 |
| 비동기 인터페이스 설계 | 비동기 처리 구조로 실시간 응답 보장 | 예약 결제, 이벤트 알림 |
| 동기 인터페이스 설계 | 요청 시 즉시 응답을 요구하는 구조 | 계좌 조회, 잔액 확인 |
| 인터페이스 보안 설계 | 전송 구간 암호화 및 인증 절차 보강 | OAuth2.0, JWT 인증 |
| 인터페이스 표준화 | 내부 및 외부 연동 시 표준 규격 통일 | JSON, XML 표준화 |
| API Gateway 설계 | 외부 시스템의 API 호출 최적화 | 핀테크 연동, 결제 게이트웨이 |
| 항목 | **설명** | **적용 사례** |
| API 표준화 | RESTful API, SOAP API 정의 | 외부 결제 시스템 연동, 보험 조회 |
| 데이터 포맷 정의 | JSON, XML 등 표준 포맷 적용 | 핀테크 간 연동, 국제 송금 |
| 인터페이스 보안 | OAuth2.0, JWT 인증 구조 설계 | 오픈 API, 클라우드 데이터 통신 |
| 상호연동 최적화 | 이기종 시스템 간 데이터 호환성 확보 | ERP 연동, 회계 시스템 통합 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 인터페이스 정의 구조 |  |------------------------------------------------------|  | 1. API 표준화 |  | - RESTful API 및 SOAP API 설계 |  | - 시스템 간 통신 최적화 |  |------------------------------------------------------|  | 2. 데이터 포맷 정의 |  | - JSON, XML 기반 통신 |  | - 핀테크 연동 및 국제 송금 지원 |  |------------------------------------------------------|  | 3. 인터페이스 보안 |  | - OAuth2.0, JWT 인증 구조 |  | - 보안성 강화된 데이터 전송 |  |------------------------------------------------------|  | 4. 상호연동 최적화 |  | - ERP, 외부 회계 시스템 간 연동 최적화 |  | - 데이터 호환성 및 안정성 확보 |  +------------------------------------------------------+ |
| +------------------------------------------------------+  | Application Architecture 인터페이스 구조 |  |------------------------------------------------------|  | 1. REST API 설계 |  | - 모바일 및 웹 인터페이스 최적화 |  | - 상태 기반 API 설계 |  |------------------------------------------------------|  | 2. SOAP API 설계 |  | - 금융기관 간 표준 메시지 포맷 설정 |  | - WSDL 기반 통신 최적화 |  |------------------------------------------------------|  | 3. 메시지 큐(MQ) 연동 |  | - 비동기 통신 구조 설계 |  | - MQ 처리 최적화 |  |------------------------------------------------------|  | 4. 파일 연동 설계 |  | - 대용량 데이터 전송 최적화 |  | - Batch 처리 구조 |  |------------------------------------------------------|  | 5. 실시간 인터페이스 설계 |  | - 실시간 데이터 반영 |  | - 트랜잭션 즉시 처리 |  |------------------------------------------------------|  | 6. 비동기 인터페이스 설계 |  | - 예약 결제, 이벤트 알림 처리 |  | - 메시지 유실 방지 구조 |  |------------------------------------------------------|  | 7. 동기 인터페이스 설계 |  | - 즉시 응답 요구 처리 |  | - 금융 조회 및 실시간 승인 |  |------------------------------------------------------|  | 8. 인터페이스 보안 설계 |  | - OAuth2.0, JWT 기반 인증 |  | - TLS/SSL 전송 보안 |  |------------------------------------------------------|  | 9. 인터페이스 표준화 |  | - JSON, XML 포맷 표준화 |  | - API 버전 관리 |  |------------------------------------------------------|  | 10. API Gateway 설계 |  | - 외부 시스템 연동 최적화 |  | - Rate Limiting 및 Circuit Breaker 적용 |  +------------------------------------------------------+ |

### 2.6.4 실사례 (설계 항목 예시 30개)

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| REST API 설계 | Representational State Transfer 방식의 표준 API 설계 | 모바일 뱅킹, 인터넷 뱅킹 | ① GET/POST/PUT/DELETE 지원② URI 설계 표준화③ HATEOAS 지원④ JSON/XML 포맷 지원⑤ Status Code 명시⑥ Payload 최적화⑦ Pagination 처리⑧ CORS 정책 적용⑨ API 버전 관리⑩ Rate Limiting 설정 |
| SOAP API 설계 | 금융기관 간 데이터 연동에 적합한 표준화된 API 설계 | 대외계 금융 시스템 연동 | ① WSDL 정의② XML Schema 검증③ SOAP Header 처리④ 보안 토큰 관리⑤ SSL/TLS 전송 보안⑥ WS-Security 적용⑦ 트랜잭션 롤백 처리⑧ 메시지 시퀀스 보장⑨ 멀티파트 메시지 처리⑩ API 로그 추적 |
| 메시지 큐(MQ) 연동 | 비동기 통신을 위한 메시지 큐 시스템 연동 | 실시간 결제 승인, 대량 이체 | ① 메시지 발행(Publish)② 메시지 구독(Subscribe)③ 메시지 순서 보장④ 재처리 로직 구현⑤ Dead Letter Queue 설정⑥ 메시지 TTL(Time To Live) 설정⑦ Topic 분리 관리⑧ 중복 메시지 방지⑨ 모니터링 및 로깅⑩ 오류 탐지 및 알림 |
| 파일 연동 설계 | 시스템 간 대용량 파일 데이터 교환 | 일별 정산 파일, 보고서 전송 | ① 파일 압축 및 암호화② 대량 파일 전송 최적화③ 파일 전송 중단 시 복구④ 전송 로그 기록⑤ SFTP/FTPS 연동⑥ 정합성 체크 및 검증⑦ 파일 포맷 변환 지원⑧ 배치 처리 지원⑨ 전송 완료 알림⑩ 스케줄링 전송 관리 |
| 실시간 인터페이스 설계 | 실시간 데이터 반영을 위한 연동 구조 | 카드 승인, 송금 처리 | ① 실시간 요청/응답 구조② 트랜잭션 동기화③ 상태 변경 실시간 반영④ 고가용성 클러스터링⑤ 빠른 응답 시간 보장⑥ 멀티스레드 처리⑦ 중복 요청 방지⑧ 오류 발생 시 롤백⑨ 데이터 무결성 유지⑩ 통합 로깅 및 추적 |
| 비동기 인터페이스 설계 | 비동기 처리 구조로 실시간 응답 보장 | 예약 결제, 이벤트 알림 | ① 이벤트 발행 및 소비② 메시지 유실 방지③ 트랜잭션 일관성 보장④ 처리 순서 보장⑤ 재처리 로직 구축⑥ 메시지 TTL 설정⑦ 비동기 에러 핸들링⑧ 메시지 중복 방지⑨ 메시지 큐 모니터링⑩ 보완 처리 (Dead Letter) |
| 동기 인터페이스 설계 | 요청 시 즉시 응답을 요구하는 구조 | 계좌 조회, 잔액 확인 | ① 요청-응답 시간 최적화② 트랜잭션 Lock 관리③ 중복 처리 방지④ 트랜잭션 커밋 최적화⑤ 다중 요청 처리 구조⑥ 응답 오류 처리⑦ 인터페이스 타임아웃 설정⑧ 상태 동기화⑨ 동시성 제어⑩ 데이터 무결성 검증 |
| 인터페이스 보안 설계 | 전송 구간 암호화 및 인증 절차 보강 | OAuth2.0, JWT 인증 | ① OAuth2.0 인증② JWT Token 발급 및 검증③ TLS/SSL 보안 통신④ 데이터 암호화(AES/RSA)⑤ CORS 정책 설정⑥ 권한 관리 및 통제⑦ API Rate Limiting⑧ 실시간 모니터링⑨ 접근 로그 추적⑩ 보안 감사 로그 |
| 인터페이스 표준화 | 내부 및 외부 연동 시 표준 규격 통일 | JSON, XML 표준화 | ① JSON Schema 정의② XML Schema 설계③ Swagger 정의서 작성④ OpenAPI 3.0 적용⑤ 메시지 포맷 표준화⑥ 필드 매핑 관리⑦ 인터페이스 명명 규칙⑧ 데이터 정합성 검증⑨ HTTP Status 표준화⑩ 인터페이스 버전 관리 |
| API Gateway 설계 | 외부 시스템의 API 호출 최적화 | 핀테크 연동, 결제 게이트웨이 | ① 인증 및 권한 관리② 로드 밸런싱 설정③ Circuit Breaker 적용④ Request Throttling⑤ Service Discovery⑥ 라우팅 최적화⑦ API 캐싱⑧ 모니터링 대시보드⑨ 실시간 트래픽 분석⑩ 장애 시 자동 페일오버 |

## 성능 최적화

### 개요

* 금융 거래의 처리 속도를 극대화하고, 트랜잭션 병목 현상을 제거하기 위한 최적화 작업을 수행합니다.
* 금융 시스템의 성능 최적화는 대량 트랜잭션 처리와 실시간 응답 시간을 보장하기 위한 중요한 설계 요소입니다.
* 특히 고가용성(High Availability)과 저지연(Low Latency) 환경을 유지하기 위해 **캐싱, 부하 분산, 비동기 처리, 인덱스 최적화**가 필수적입니다.
* 성능 최적화는 데이터 처리 속도 향상과 동시에 시스템의 안정성을 높이는 데 중점을 둡니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 캐싱 설계 | 데이터 조회 성능을 높이기 위해 자주 사용하는 데이터를 메모리에 캐싱 | 실시간 조회, 통계 조회 |
| 부하 분산 설계 | 트래픽을 여러 서버로 분산하여 처리 성능 향상 | 인터넷 뱅킹, 모바일 뱅킹 |
| 비동기 처리 최적화 | 비동기 이벤트 처리로 응답 시간 단축 | 카드 승인, 예약 결제 |
| 인덱스 최적화 | 대용량 데이터 조회 성능 극대화 | 거래 내역 조회, 계좌 이력 |
| 멀티스레드 처리 최적화 | 다중 스레드로 동시 처리 성능 극대화 | 실시간 송금, 다중 결제 승인 |
| 쿼리 최적화 | SQL 쿼리의 실행 시간 단축 | 거래 정산, 보고서 생성 |
| 데이터 압축 및 최적화 | 대량 데이터 전송 시 네트워크 사용량 최적화 | 금융 보고서 전송, 로그 저장 |
| 네트워크 최적화 | 통신 구간의 레이턴시 감소 | API Gateway, 모바일 트랜잭션 |
| 리소스 풀링 설계 | DB Connection Pool, Thread Pool 최적화 | 대량 이체 처리, 트랜잭션 동시 처리 |
| 로드 밸런싱 설계 | 트래픽 부하를 여러 서버에 분산 | 인터넷 뱅킹, 결제 시스템 |
| 항목 | **설명** | **적용 사례** |
| 트랜잭션 최적화 | 거래 처리 속도 개선 및 안정화 | 초당 수천 건의 거래 처리 |
| 캐싱 전략 | 자주 조회되는 데이터 캐싱 처리 | 잔액 조회, 환율 조회 |
| DB 파티셔닝 | 대량 데이터 처리를 위한 DB 분리 | 고객 정보 파티셔닝, 로그 데이터 관리 |
| 로드 밸런싱 | 서버 부하를 분산하여 성능 극대화 | 대규모 이벤트 처리, 금융 캠페인 |
|  |  |  |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 성능 최적화 구조 |  |------------------------------------------------------|  | 1. 트랜잭션 최적화 |  | - 실시간 트랜잭션 처리 |  | - 초당 수천 건의 처리 가능 |  |------------------------------------------------------|  | 2. 캐싱 전략 |  | - 자주 조회되는 데이터 캐싱 |  | - 빠른 응답 시간 제공 |  |------------------------------------------------------|  | 3. DB 파티셔닝 |  | - 데이터베이스 분리 저장 |  | - I/O 최적화 및 확장성 확보 |  |------------------------------------------------------|  | 4. 로드 밸런싱 |  | - 서버 간 부하 분산 |  | - 장애 대응 및 트래픽 관리 |  +------------------------------------------------------+ |

### 2.7.4 실사례 (설계 항목 예시 30개)

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| 캐싱 설계 | 데이터 조회 성능을 높이기 위해 자주 사용하는 데이터를 메모리에 캐싱 | 실시간 조회, 통계 조회 | ① Redis Cache 활용② Memcached 설정③ Local Cache 지원④ LRU 정책 적용⑤ TTL(Time To Live) 설정⑥ 분산 캐시 클러스터링⑦ 멀티 레벨 캐싱⑧ 캐시 무효화 전략⑨ Lazy Loading⑩ 캐시 히트율 분석 |
| 부하 분산 설계 | 트래픽을 여러 서버로 분산하여 처리 성능 향상 | 인터넷 뱅킹, 모바일 뱅킹 | ① Round Robin 분산② Least Connection 분산③ Hash 기반 라우팅④ Global Load Balancer 설정⑤ Failover 처리⑥ Health Check 모니터링⑦ Auto Scaling 설정⑧ Multi-Zone 분산⑨ 세션 유지 최적화⑩ 서버 간 상태 동기화 |
| 비동기 처리 최적화 | 비동기 이벤트 처리로 응답 시간 단축 | 카드 승인, 예약 결제 | ① 메시지 큐 적용② Future 패턴 사용③ Event-Driven 구조④ 비동기 로깅 처리⑤ 비동기 트랜잭션 커밋⑥ 메시지 재처리 로직⑦ 동시성 제어⑧ Latency 최적화⑨ Dead Letter Queue 설정⑩ 이벤트 순서 보장 |
| 인덱스 최적화 | 대용량 데이터 조회 성능 극대화 | 거래 내역 조회, 계좌 이력 | ① B-Tree 인덱스 생성② 해시 인덱스 설정③ Composite Index 설계④ Partial Index 적용⑤ 인덱스 리빌드 최적화⑥ 다중 필드 인덱스 생성⑦ Unique Index 설정⑧ 인덱스 통계 분석⑨ 인덱스 조정 스케줄링⑩ 쿼리 실행 최적화 |
| 멀티스레드 처리 최적화 | 다중 스레드로 동시 처리 성능 극대화 | 실시간 송금, 다중 결제 승인 | ① Thread Pool 설정② 동시성 제어 관리③ Lock-Free 처리④ Optimistic Locking⑤ Pessimistic Locking⑥ Thread Safe 구조 설계⑦ 병렬 처리 최적화⑧ 스레드 우선순위 관리⑨ 자원 경쟁 방지⑩ 스레드 상태 모니터링 |
| 쿼리 최적화 | SQL 쿼리의 실행 시간 단축 | 거래 정산, 보고서 생성 | ① 쿼리 계획 분석② 조인 최적화③ 서브쿼리 최소화④ Selective Column Fetch⑤ 실행 계획 캐싱⑥ Index Hint 적용⑦ 동적 쿼리 최적화⑧ In-memory 조회 최적화⑨ Bulk Update 최적화⑩ Query Rewrite 최적화 |
| 데이터 압축 및 최적화 | 대량 데이터 전송 시 네트워크 사용량 최적화 | 금융 보고서 전송, 로그 저장 | ① GZIP 압축 처리② Brotli 압축 설정③ 데이터 전송량 최적화④ 네트워크 대역폭 최적화⑤ 파일 전송 속도 개선⑥ 중복 데이터 제거⑦ 대용량 Blob 전송 최적화⑧ 실시간 압축 해제⑨ CDN 연동 최적화⑩ 파일 스토리지 최적화 |
| 네트워크 최적화 | 통신 구간의 레이턴시 감소 | API Gateway, 모바일 트랜잭션 | ① TCP/IP 최적화② HTTP Keep-Alive 설정③ 로컬 DNS 캐싱④ 네트워크 재전송 최적화⑤ CDN(Content Delivery Network) 사용⑥ WebSocket 최적화⑦ 네트워크 패킷 분석⑧ 통신 암호화 최적화⑨ QoS(Quality of Service) 관리⑩ 네트워크 대역폭 관리 |
| 리소스 풀링 설계 | DB Connection Pool, Thread Pool 최적화 | 대량 이체 처리, 트랜잭션 동시 처리 | ① Connection Pool 설정② Thread Pool 최적화③ Idle Timeout 설정④ Min/Max Pool Size 관리⑤ 재접속 로직 설정⑥ Connection Leak 탐지⑦ Pool Monitoring⑧ Prepared Statement 캐싱⑨ 자동 커밋 관리⑩ 비동기 풀링 처리 |
| 로드 밸런싱 설계 | 트래픽 부하를 여러 서버에 분산 | 인터넷 뱅킹, 결제 시스템 | ① Round Robin 분산② Least Connection 분산③ Consistent Hashing④ Global Load Balancing⑤ Failover 처리⑥ Multi-Region 배포⑦ Auto Scaling 설정⑧ Sticky Session 관리⑨ Traffic Routing 최적화⑩ DNS Load Balancing |

## 보안 설계

### 개요

* 금융 거래의 안정성과 데이터 보호를 보장하기 위한 보안 체계를 설계합니다. 특히 금융권에서는 개인정보 보호, 거래 무결성, 네트워크 보안이 필수적입니다.
* 이를 위해 다단계 인증, 데이터 암호화, 접근 제어, 보안 모니터링 체계를 구축합니다.
*  금융 시스템의 보안 설계는 고객의 자산 보호와 민감한 정보의 안전한 관리를 보장하는 필수 요소입니다.
*  접근 통제, 데이터 암호화, 전송 구간 보안, 인증 처리, 권한 관리 등 다층적인 보안 구조가 필요합니다.
*  금융 거래 시 발생할 수 있는 **해킹, 정보 유출, 데이터 변조**를 방지하기 위한 강력한 보안 체계가 필요합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 접근 제어 설계 | 사용자의 권한을 제어하고 불법 접근을 차단 | 관리자 페이지, 고객 정보 조회 |
| 데이터 암호화 설계 | 민감한 금융 정보를 암호화하여 보안 강화 | 계좌 정보, 송금 내역 |
| 전송 구간 보안 설계 | 데이터 전송 중 도청 및 변조 방지 | 인터넷 뱅킹, 모바일 결제 |
| 인증 처리 설계 | 사용자 인증 및 다단계 인증 처리 | OTP, MFA(Multi-Factor Authentication) |
| 권한 관리 설계 | 사용자 그룹별 접근 권한 설정 | 내부 직원 권한, 외부 사용자 권한 |
| 로그 및 감사 설계 | 금융 거래의 이력 추적 및 보안 감사 | 거래 이력 조회, 보안 감사 로그 |
| 보안 모니터링 설계 | 실시간 침입 탐지 및 보안 위협 감지 | 실시간 모니터링, IDS/IPS |
| 취약점 점검 및 대응 설계 | 보안 취약점 진단 및 대응 방안 수립 | SQL Injection, XSS 방지 |
| 보안 이벤트 관리 | 금융 사고 발생 시 신속한 대응 체계 | DDoS 대응, 해킹 탐지 |
| 데이터 무결성 보호 | 데이터 변경 및 변조 방지 | 송금 내역 보존, 거래 기록 보호 |
| 항목 | **설명** | **적용 사례** |
| 다단계 인증 (MFA) | OTP, 바이오 인증 등 다단계 인증 체계 구축 | 인터넷 뱅킹, 대출 신청 시 추가 인증 |
| 데이터 암호화 | AES, RSA 기반의 민감 데이터 암호화 | 고객 정보, 금융 거래 내역 |
| 접근 제어 (ACL) | 사용자 권한에 따른 접근 제어 관리 | 관리자, 고객, 협력사 간 접근 차별화 |
| 보안 모니터링 | 실시간 로그 분석 및 침입 탐지 | DDOS 공격 탐지, 비정상 거래 탐지 |
| 네트워크 보안 | 방화벽, IDS/IPS를 통한 네트워크 보호 | 내부망과 외부망 간의 통신 보안 |

### 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 보안 설계 구조 |  |------------------------------------------------------|  | 1. 다단계 인증 (MFA) |  | - OTP, 바이오 인증 적용 |  | - 로그인 및 고액 거래 시 추가 인증 |  |------------------------------------------------------|  | 2. 데이터 암호화 |  | - AES, RSA 암호화 적용 |  | - 금융 거래 및 고객 정보 보호 |  |------------------------------------------------------|  | 3. 접근 제어 (ACL) |  | - 사용자 권한에 따른 접근 관리 |  | - 관리자, 고객, 외부 파트너 권한 구분 |  |------------------------------------------------------|  | 4. 보안 모니터링 |  | - 실시간 로그 분석 |  | - 침입 탐지 및 비정상 거래 모니터링 |  |------------------------------------------------------|  | 5. 네트워크 보안 |  | - IDS/IPS, 방화벽 구성 |  | - 외부 위협 및 DDoS 방어 |  +------------------------------------------------------+ |
| +------------------------------------------------------+  | Application Architecture 보안 설계 구조 |  |------------------------------------------------------|  | 1. 접근 제어 설계 |  | - 사용자 권한 제어 및 인증 |  | - Role-Based Access Control (RBAC) |  |------------------------------------------------------|  | 2. 데이터 암호화 설계 |  | - AES, RSA 암호화 적용 |  | - 민감 정보 보호 |  |------------------------------------------------------|  | 3. 전송 구간 보안 설계 |  | - TLS/SSL 전송 보안 |  | - HTTPS 통신 최적화 |  |------------------------------------------------------|  | 4. 인증 처리 설계 |  | - OAuth2.0, JWT 인증 |  | - 다단계 인증 (MFA) |  |------------------------------------------------------|  | 5. 권한 관리 설계 |  | - 사용자 역할 기반 접근 관리 |  | - 세션 만료 및 토큰 만료 처리 |  |------------------------------------------------------|  | 6. 로그 및 감사 설계 |  | - 거래 이력 추적 |  | - 보안 감사 로그 |  |------------------------------------------------------|  | 7. 보안 모니터링 설계 |  | - 실시간 침입 탐지 시스템 |  | - 공격 패턴 탐지 및 대응 |  |------------------------------------------------------|  | 8. 취약점 점검 및 대응 설계 |  | - 정기적인 취약점 스캔 |  | - 취약점 리포트 및 패치 관리 |  |------------------------------------------------------|  | 9. 보안 이벤트 관리 |  | - 금융 사고 발생 시 실시간 대응 |  | - 이벤트 알림 및 복구 절차 |  |------------------------------------------------------|  | 10. 데이터 무결성 보호 |  | - 데이터 변경 및 변조 방지 |  | - 이력 추적 및 백업 관리 |  +------------------------------------------------------+ |

### 2.8.4 실사례 (설계 항목 예시 30개)

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| 접근 제어 설계 | 사용자의 권한을 제어하고 불법 접근을 차단 | 관리자 페이지, 고객 정보 조회 | ① Role-Based Access Control② 세션 관리③ 사용자 인증 정책④ 접근 로그 추적⑤ 비인가 접근 차단⑥ 다단계 승인 프로세스⑦ 권한 위임 관리⑧ 정책 기반 제어⑨ 실시간 접근 모니터링⑩ 접근 제한 알림 |
| 데이터 암호화 설계 | 민감한 금융 정보를 암호화하여 보안 강화 | 계좌 정보, 송금 내역 | ① AES 암호화② RSA 암호화③ TLS/SSL 통신 보안④ 키 관리 시스템 연동⑤ 데이터 복호화 관리⑥ 저장 데이터 암호화⑦ 필드 레벨 암호화⑧ 접근 통제 설정⑨ 보안 로그 추적⑩ 정책 기반 암호화 |
| 전송 구간 보안 설계 | 데이터 전송 중 도청 및 변조 방지 | 인터넷 뱅킹, 모바일 결제 | ① HTTPS 통신② TLS/SSL 인증서 관리③ 전송 중 암호화④ 무결성 체크⑤ 세션 보호⑥ 양방향 인증⑦ Man-in-the-Middle 공격 방지⑧ 데이터 유출 방지⑨ IPSec 연동⑩ VPN 연결 보안 |
| 인증 처리 설계 | 사용자 인증 및 다단계 인증 처리 | OTP, MFA | ① OTP 발급 및 인증② 생체 인증 처리③ 다중 디바이스 인증④ 인증 실패 시 계정 잠금⑤ 2단계 인증 절차⑥ 일회용 패스워드 생성⑦ 인증 로그 추적⑧ 임시 비밀번호 발급⑨ 분산 인증 처리⑩ 세션 하이재킹 방지 |
| 권한 관리 설계 | 사용자 그룹별 접근 권한 설정 | 내부 직원 권한, 외부 사용자 권한 | ① 사용자 그룹 정의② Role-Based Access Control③ 계층적 권한 설정④ 권한 상속 구조⑤ 권한 승인 프로세스⑥ 정책 기반 접근 통제⑦ 권한 변경 시 실시간 반영⑧ 비인가 접근 로그 추적⑨ 권한 요청 워크플로우⑩ 권한 만료 및 철회 |
| 로그 및 감사 설계 | 금융 거래의 이력 추적 및 보안 감사 | 거래 이력 조회, 보안 감사 로그 | ① 트랜잭션 이력 저장② 보안 이벤트 기록③ 접속 이력 관리④ 감사 로그 생성⑤ 데이터 변경 이력 추적⑥ 사용자 활동 모니터링⑦ 보안 위반 탐지⑧ 시계열 로그 분석⑨ 로그 백업 및 보관⑩ 규정 준수 감사 |
| 보안 모니터링 설계 | 실시간 침입 탐지 및 보안 위협 감지 | 실시간 모니터링, IDS/IPS | ① 실시간 이벤트 모니터링② IDS/IPS 연동③ DDoS 공격 탐지④ 패턴 기반 침입 탐지⑤ Zero-Day 공격 탐지⑥ 데이터 유출 경고⑦ 실시간 알림 시스템⑧ 트래픽 분석⑨ 정책 위반 탐지⑩ 침입 시 자동 차단 |
| 취약점 점검 및 대응 설계 | 보안 취약점 진단 및 대응 방안 수립 | SQL Injection, XSS 방지 | ① SQL Injection 차단② XSS(Cross-Site Scripting) 방어③ CSRF(Cross-Site Request Forgery) 방어④ 보안 패치 관리⑤ 취약점 탐지 스캐닝⑥ OWASP Top 10 대응⑦ 버전 업데이트 관리⑧ 보안 설정 감사⑨ Penetration Test 계획⑩ 보안 리포트 생성 |
| 보안 이벤트 관리 | 금융 사고 발생 시 신속한 대응 체계 | DDoS 대응, 해킹 탐지 | ① 보안 이벤트 탐지② 실시간 알림 발송③ 자동 차단 및 복구④ 실시간 로깅⑤ 패턴 분석 및 탐지⑥ 정책 기반 탐지⑦ 공격 시도 차단⑧ 비정상 트래픽 분석⑨ DDoS 대응 계획⑩ 장애 복구 계획 |
| 데이터 무결성 보호 | 데이터 변경 및 변조 방지 | 송금 내역 보존, 거래 기록 보호 | ① ACID 속성 보장② 데이터 변경 이력 관리③ 데이터 무결성 검증④ 트랜잭션 롤백⑤ 동시성 제어⑥ 다중 접근 제어⑦ 데이터 정합성 체크⑧ 트랜잭션 Lock 관리⑨ 중복 방지 처리⑩ 데이터 감사 및 추적 |

## 테스트 및 검증

### 개요

* 설계된 아키텍처가 요구사항을 만족하고, 금융 시스템의 안정성과 무결성을 보장하기 위해 테스트 계획을 수립하고 검증을 수행합니다.
* 주요 금융 서비스에 대한 부하 테스트, 침투 테스트, 회귀 테스트를 포함합니다.
*  금융 시스템의 테스트 및 검증은 서비스의 안정성과 무결성을 보장하기 위한 필수 단계입니다.
*  주요 기능의 정상 동작 확인, 성능 최적화, 장애 복구 능력 등을 평가하고 검증합니다.
*  테스트는 **기능 테스트, 성능 테스트, 보안 테스트, 통합 테스트, 회귀 테스트**로 나뉘며, 각 테스트가 성공적으로 완료되어야 서비스가 배포됩니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 기능 테스트 설계 | 주요 기능의 정상 동작 여부를 검증 | 송금 처리, 계좌 이체, 잔액 조회 |
| 성능 테스트 설계 | 고부하 상황에서의 처리 성능 검증 | 대량 송금, 실시간 거래 |
| 보안 테스트 설계 | 금융 데이터의 안전성과 취약점 분석 | 개인정보 보호, 암호화 검증 |
| 통합 테스트 설계 | 여러 모듈 간 연계 동작 검증 | 결제 시스템 연동, 내부 시스템 연계 |
| 회귀 테스트 설계 | 변경 사항이 기존 기능에 영향이 없는지 검증 | 버그 수정 후 재검증 |
| 장애 복구 테스트 설계 | 시스템 장애 발생 시 복구 능력 평가 | DB 장애 복구, 네트워크 복구 |
| 사용자 시나리오 테스트 설계 | 실제 사용자의 시나리오를 반영한 테스트 | 사용자 거래, 로그인 및 이체 |
| API 테스트 설계 | API의 기능, 보안, 성능을 검증 | OpenAPI, REST API |
| 데이터 일관성 테스트 설계 | 트랜잭션 처리 중 데이터 무결성 검증 | 다중 거래 처리, 동시성 관리 |
| 배포 테스트 설계 | 서비스 배포 시 장애 없이 정상 동작 검증 | 무중단 배포, Canary 배포 |
| 항목 | **설명** | **적용 사례** |
| 부하 테스트 | 최대 거래량에 대한 시스템 처리 성능 검증 | 대량 이체, 정산 프로세스 |
| 침투 테스트 | 보안 취약점을 탐색하고 보완 | 웹 애플리케이션, API Gateway |
| 회귀 테스트 | 업데이트 후 기존 기능의 정상 작동 여부 확인 | 정기 업데이트, 보안 패치 |
| 통합 테스트 | 각 모듈 간의 통신과 데이터 연동 검증 | 결제 시스템 ↔ 은행 시스템 연동 |
| 사용자 수용 테스트 (UAT) | 최종 사용자 환경에서 요구사항 충족 여부 확인 | 모바일 뱅킹 앱, 인터넷 뱅킹 시스템 |

### 도형 기반 아키텍처 구성도

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 테스트 설계 구조 |  |------------------------------------------------------|  | 1. 기능 테스트 설계 |  | - 주요 기능 정상 동작 확인 |  | - 송금, 조회, 결제 처리 검증 |  |------------------------------------------------------|  | 2. 성능 테스트 설계 |  | - 고부하 상황에서의 처리 성능 검증 |  | - TPS, Latency, Throughput 측정 |  |------------------------------------------------------|  | 3. 보안 테스트 설계 |  | - 금융 데이터의 안전성 검증 |  | - 암호화, 접근 제어, SQL Injection 방지 |  |------------------------------------------------------|  | 4. 통합 테스트 설계 |  | - 여러 모듈 간 상호작용 검증 |  | - 결제 시스템 연동, 내부 시스템 통신 |  |------------------------------------------------------|  | 5. 회귀 테스트 설계 |  | - 수정된 기능이 기존 기능에 미치는 영향 분석 |  | - 이전 버전과의 호환성 검증 |  |------------------------------------------------------|  | 6. 장애 복구 테스트 설계 |  | - 장애 발생 시 복구 능력 확인 |  | - DB 복구, 네트워크 장애 대응 |  |------------------------------------------------------|  | 7. 사용자 시나리오 테스트 설계 |  | - 사용자 경험을 기반으로 실제 거래 시나리오 테스트 |  | - 로그인, 송금, 결제 프로세스 검증 |  |------------------------------------------------------|  | 8. API 테스트 설계 |  | - API의 기능, 보안, 성능을 검증 |  | - REST API, SOAP API 검증 |  |------------------------------------------------------|  | 9. 데이터 일관성 테스트 설계 |  | - 다중 거래 처리 시 데이터 무결성 검증 |  | - 동시성 관리, 트랜잭션 정합성 확인 |  |------------------------------------------------------|  | 10. 배포 테스트 설계 |  | - 배포 시 무중단 서비스 유지 |  | - Canary 배포, Blue-Green 배포 |  +------------------------------------------------------+ |
| +------------------------------------------------------+  | Application Architecture 테스트 및 검증 구조 |  |------------------------------------------------------|  | 1. 부하 테스트 |  | - 대량 거래 시 성능 검증 |  | - 거래 실패율 및 응답 시간 측정 |  |------------------------------------------------------|  | 2. 침투 테스트 |  | - 보안 취약점 탐색 및 방어 |  | - 웹 애플리케이션 및 API 보안 점검 |  |------------------------------------------------------|  | 3. 회귀 테스트 |  | - 시스템 업데이트 후 정상 작동 여부 확인 |  | - 기능 변화 시 영향도 분석 |  |------------------------------------------------------|  | 4. 통합 테스트 |  | - 시스템 간 데이터 연동 테스트 |  | - 실시간 연계 구조 검증 |  |------------------------------------------------------|  | 5. 사용자 수용 테스트 (UAT) |  | - 실제 사용자 환경에서 요구사항 충족 여부 확인 |  | - 모바일 뱅킹, 인터넷 뱅킹 실 사용자 테스트 |  +------------------------------------------------------+ |

### 실사례

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 설계 항목 | 설명 | | 적용 사례 | 세부 기능 |
| 기능 테스트 설계 | 주요 기능의 정상 동작 여부를 검증 | | 송금 처리, 계좌 이체 | ① 사용자 시나리오 기반 테스트② 화면 전환 검증③ 입력 유효성 체크④ 오류 메시지 처리⑤ 예외 상황 처리⑥ 비동기 처리 검증⑦ 거래 완료 알림⑧ 계좌 간 이체 검증⑨ 실시간 잔액 확인⑩ 트랜잭션 롤백 테스트 |
| 성능 테스트 설계 | 고부하 상황에서의 처리 성능 검증 | | 대량 송금, 실시간 거래 | ① TPS 측정② Latency 측정③ Throughput 분석④ 메모리 사용량 최적화⑤ CPU 사용률 측정⑥ 네트워크 지연 시간 분석⑦ 부하 시 장애 탐지⑧ Auto Scaling 테스트⑨ 스토리지 사용량 최적화⑩ 트랜잭션 처리 속도 |
| 보안 테스트 설계 | 금융 데이터의 안전성과 취약점 분석 | | 개인정보 보호, 암호화 검증 | ① SQL Injection 방지② XSS(Cross-Site Scripting) 방지③ CSRF(Cross-Site Request Forgery) 방지④ 인증 처리 검증⑤ 데이터 전송 암호화⑥ 권한 관리 검증⑦ 세션 하이재킹 방지⑧ 접근 제어 정책 테스트⑨ 보안 로깅⑩ 네트워크 스니핑 방지 |
| 통합 테스트 설계 | 여러 모듈 간 연계 동작 검증 | | 결제 시스템 연동, 내부 시스템 연계 | ① 서비스 간 데이터 연동 테스트② API 통신 검증③ 메시지 큐 통합 테스트④ 트랜잭션 연계 처리 검증⑤ 장애 발생 시 롤백 확인⑥ 외부 연계 시스템 테스트⑦ 동기/비동기 인터페이스 테스트⑧ 통신 타임아웃 검증⑨ 데이터 일관성 유지 확인⑩ 인터페이스 보안 검증 |
| 회귀 테스트 설계 | 변경 사항이 기존 기능에 영향이 없는지 검증 | | 버그 수정 후 재검증 | ① 수정된 코드의 재검증② 기존 기능 정상 동작 확인③ 트랜잭션 처리 오류 검출④ UI 변경 사항 반영 확인⑤ 성능 저하 여부 검사⑥ 데이터 유실 확인⑦ 기존 테스트 케이스 반복 실행⑧ 통합 모듈 테스트⑨ 업무 로직 변화 영향 분석⑩ 배포 후 최종 점검 |
| 장애 복구 테스트 설계 | 시스템 장애 발생 시 복구 능력 평가 | | DB 장애 복구, 네트워크 복구 | ① 장애 탐지 시 알림② Failover 처리③ 백업 데이터 복원④ 이중화 구성 복구⑤ 네트워크 연결 복구⑥ DB 복구 시 트랜잭션 정합성 확인⑦ 데이터 무결성 검증⑧ 복구 후 로깅 검증⑨ 복구 시간 측정⑩ 자동화된 복구 프로세스 |
| 사용자 시나리오 테스트 설계 | 실제 사용자의 시나리오를 반영한 테스트 | | 사용자 거래, 로그인 및 이체 | ① 로그인 시나리오 검증② 송금 및 이체 절차 테스트③ 사용자 권한에 따른 접근 제한④ 거래 내역 조회 시나리오⑤ 다중 계좌 송금 테스트⑥ 오류 상황 시 복구 확인⑦ 사용자 알림 검증⑧ 인증 오류 처리⑨ 중복 거래 방지⑩ 로그아웃 처리 검증 |
| API 테스트 설계 | API의 기능, 보안, 성능을 검증 | | OpenAPI, REST API | ① API 엔드포인트 테스트② 요청/응답 시간 측정③ 보안 토큰 검증④ 입력 파라미터 유효성 체크⑤ 비정상 요청 처리⑥ 멱등성(Idempotency) 검증⑦ 상태 코드 검증⑧ CORS 정책 테스트⑨ Rate Limiting 적용 검증⑩ 응답 데이터 포맷 확인 |
| 데이터 일관성 테스트 설계 | 트랜잭션 처리 중 데이터 무결성 검증 | | 다중 거래 처리, 동시성 관리 | ① 데이터 정합성 체크② 동시성 제어 검증③ 트랜잭션 커밋 검증④ Lock 처리 확인⑤ Dirty Read 방지⑥ Non-Repeatable Read 방지⑦ Phantom Read 방지⑧ 데이터 중복 방지⑨ 장애 시 롤백 테스트⑩ 복구 후 데이터 정합성 확인 |
| 배포 테스트 설계 | 서비스 배포 시 장애 없이 정상 동작 검증 | | 무중단 배포, Canary 배포 | ① Blue-Green Deployment② Canary Release 검증③ Zero Downtime 배포 확인④ 트래픽 스위칭 검증⑤ 롤백 시나리오 검증⑥ 배포 후 서비스 상태 점검⑦ 구성 변경 반영 검증⑧ 버전 호환성 체크⑨ Scale Out 테스트⑩ 모니터링 및 로깅 |
| 사례명 | | **주요 내용** | | **주요 개선 효과** |
| 다단계 인증 적용 | | 금융 거래 시 OTP와 바이오 인증을 도입하여 보안 강화 | | 거래 보안성 40% 향상, 인증 실패율 감소 |
| API Gateway 보안 최적화 | | API 통신 시 OAuth2.0 인증 구조 도입 | | 데이터 유출 30% 감소, 접근 보안성 향상 |
| 대량 거래 부하 테스트 | | 10만 건 이상의 거래를 동시 처리 시 검증 수행 | | 거래 속도 20% 향상, 오류율 감소 |
| 침투 테스트 개선 | | 외부 공격 탐지와 방어 체계 구축 | | 해킹 시도 차단율 50% 증가 |
| 회귀 테스트 자동화 | | 정기적 업데이트 후 기존 기능의 안정성 확인 | | 장애 발생률 15% 감소, 유지보수 비용 절감 |

### 마무리말

테스트 및 검증은 금융 시스템의 안정성과 신뢰성을 확보하기 위한 필수적인 단계입니다. 특히 금융 거래는 실시간성과 정확성이 요구되기 때문에 **기능 테스트, 성능 테스트, 보안 테스트, 통합 테스트, 회귀 테스트** 등이 모두 철저하게 수행되어야 합니다.  
각각의 테스트 항목이 실제 서비스에 반영될 때, 예상치 못한 문제를 최소화하고 \*\*무결성(Integrity)\*\*과 \*\*일관성(Consistency)\*\*을 보장할 수 있어야 하며, 장애 상황에서도 빠른 복구가 가능하도록 준비되어야 합니다.

이를 통해 서비스의 가용성을 높이고, 고객 신뢰도를 유지할 수 있습니다. 특히 API 연동, 데이터 무결성, 배포 시 무중단 처리 등의 영역에서 반복적인 테스트를 통해 오류를 사전에 발견하고 대응하는 것이 중요합니다.

### 시사점

 **안정적 서비스 제공**

* 금융 시스템의 특성상 거래 오류나 보안 취약점은 곧바로 고객 피해로 이어지므로, 철저한 테스트를 통해 신뢰성을 확보해야 한다.

 **테스트 자동화의 필요성**

* 기능 테스트, 성능 테스트, 회귀 테스트는 자동화된 CI/CD 파이프라인에 포함되어 반복적인 검증이 필요하다. 이를 통해 배포 주기를 단축하고, 신속한 오류 탐지가 가능해진다.

 **보안 테스트의 중요성**

* 보안 사고가 금융 거래에 미치는 영향은 치명적이므로, SQL Injection, XSS, CSRF 같은 취약점에 대한 철저한 방어 체계가 필요하다.

 **통합 테스트를 통한 모듈 간 연계 보장**

* 여러 모듈 간 상호작용이 정상적으로 작동하는지에 대한 통합 테스트는 필수적이다. 특히 외부 금융기관 API 연동 시, 데이터 정합성을 보장할 수 있어야 한다.

 **장애 복구 능력의 강화**

* 예상치 못한 장애 발생 시 빠르게 복구할 수 있는 능력을 키워야 한다. 이를 위해 **Failover 구조, 이중화 설계, 데이터 백업 정책**을 테스트 환경에서 지속적으로 검증해야 한다.

 **사용자 시나리오 기반 테스트의 필요성**

* 실제 사용자 행동을 기반으로 한 시나리오 테스트를 통해 예상치 못한 오류를 사전에 발견하고 조치할 수 있어야 한다.

 **데이터 일관성 검증의 철저함**

* 거래 도중 발생할 수 있는 데이터 불일치를 막기 위해 데이터 무결성 테스트가 필수적이다. 트랜잭션이 중단되거나 Rollback이 발생할 경우, 일관성을 유지할 수 있도록 설계되어야 한다.

|  |  |
| --- | --- |
| 시사점 | 설명 |
| 보안 강화 | 다단계 인증과 데이터 암호화를 통해 금융 거래의 안전성 확보 |
| 시스템 무결성 보장 | 테스트와 검증을 통한 데이터 오류 방지와 성능 최적화 |
| 실시간 보안 모니터링 | 침입 탐지와 실시간 로그 분석을 통해 빠른 대응 가능 |

# **애플리케이션 아키텍처 설계**

## **개요**

* 금융 시스템의 애플리케이션 아키텍처 설계는 안정적인 거래 처리와 확장성을 보장하기 위한 핵심 설계입니다.
* 클라이언트 요청부터 데이터베이스 처리까지의 **계층적 구조**를 최적화하고, **MSA 구조**, **비동기 처리**를 도입하여 고성능을 유지합니다.
* **서비스 간 통신, 트랜잭션 관리, 로드 밸런싱, 장애 복구** 등이 효과적으로 설계되어야 금융 거래의 안정성과 무결성을 보장할 수 있습니다.

## **주요 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 계층형 아키텍처 설계 | Presentation, Business, Data Layer 구조화 | 인터넷 뱅킹, 모바일 뱅킹 |
| MSA (Microservice Architecture) 설계 | 독립적인 서비스 구조로 확장성 극대화 | 글로벌 결제 시스템, 대출 관리 |
| 비동기 처리 설계 | 트랜잭션 처리 속도 최적화 | 실시간 결제 승인, 예약 처리 |
| 트랜잭션 관리 설계 | ACID 원칙 준수와 동시성 처리 보장 | 실시간 이체, 대량 거래 처리 |
| 서비스 디스커버리 설계 | 마이크로서비스 간 동적 라우팅 및 탐색 | API Gateway, 서비스 메시 |
| API Gateway 설계 | 외부 서비스와의 통신 최적화 | 핀테크 연동, 결제 시스템 |
| 이벤트 소싱(Event Sourcing) 설계 | 상태 변경 이력을 저장하여 데이터 정합성 유지 | 거래 내역 추적, 정산 시스템 |
| CQRS(Command Query Responsibility Segregation) 설계 | 명령과 조회의 처리 경로를 분리 | 대량 조회 최적화, 대기 시간 감소 |
| 로드 밸런싱 설계 | 트래픽을 여러 서버로 분산하여 처리 성능 향상 | 인터넷 뱅킹, 결제 시스템 |
| 장애 복구 및 이중화 설계 | 시스템 장애 시 빠른 복구를 보장 | 금융 트랜잭션 이중화, 데이터 복구 |

## **구성도**

plaintext

복사편집

+------------------------------------------------------+

| Application Architecture 아키텍처 구조 |

|------------------------------------------------------|

| 1. 계층형 아키텍처 설계 |

| - Presentation Layer: 사용자 인터페이스 |

| - Business Layer: 비즈니스 로직 처리 |

| - Data Layer: 데이터 처리 및 저장 |

|------------------------------------------------------|

| 2. MSA (Microservice Architecture) 설계 |

| - 독립 서비스 배포 |

| - 확장성 및 장애 격리 보장 |

|------------------------------------------------------|

| 3. 비동기 처리 설계 |

| - 이벤트 기반 메시지 큐 처리 |

| - 실시간 트랜잭션 반영 |

|------------------------------------------------------|

| 4. 트랜잭션 관리 설계 |

| - ACID 원칙 준수 |

| - 동시성 처리 보장 |

|------------------------------------------------------|

| 5. 서비스 디스커버리 설계 |

| - 마이크로서비스 동적 탐색 |

| - 로드 밸런싱 최적화 |

|------------------------------------------------------|

| 6. API Gateway 설계 |

| - 외부 시스템과의 통신 최적화 |

| - Rate Limiting 및 보안 설정 |

|------------------------------------------------------|

| 7. 이벤트 소싱 설계 |

| - 트랜잭션 상태 변경 내역 저장 |

| - 이벤트 재처리 지원 |

|------------------------------------------------------|

| 8. CQRS 설계 |

| - Command와 Query의 경로 분리 |

| - 대량 데이터 처리 최적화 |

|------------------------------------------------------|

| 9. 로드 밸런싱 설계 |

| - 트래픽 분산 처리 |

| - Failover 및 확장 지원 |

|------------------------------------------------------|

| 10. 장애 복구 및 이중화 설계 |

| - 이중화된 시스템 복구 |

| - 실시간 데이터 복원 |

+------------------------------------------------------+

## **상세 주요 내용 설계**

### **계층형 아키텍처 설계**

#### **개요**

* 계층형 아키텍처는 Presentation, Business, Data Layer로 구성되어 각각의 역할을 명확하게 분리합니다.
* Presentation Layer는 UI를 관리하고, Business Layer는 비즈니스 로직을 처리하며, Data Layer는 데이터의 영속성을 보장합니다.
* 이 구조는 유지보수성과 확장성에 매우 유리하며, 역할이 명확히 분리되어 높은 응집도와 낮은 결합도를 유지할 수 있습니다.

#### **주요 설계 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| MVC 패턴 설계 | Controller, Service, DAO의 역할 구분 | ① Controller는 HTTP 요청 처리 ② Service는 비즈니스 로직 처리 ③ DAO는 DB 연동 ④ Data Transfer Object(DTO) 사용 ⑤ View와 Model의 분리 ⑥ Validation 로직 처리 ⑦ 요청/응답 매핑 최적화 ⑧ RESTful API 설계 ⑨ JSON/XML 매핑 처리 ⑩ 비즈니스 로직 재사용성 보장 |
| 트랜잭션 관리 | 비즈니스 로직 단위의 트랜잭션 처리 | ① 커밋 및 롤백 처리 ② 동시성 제어 (Optimistic/Pessimistic Locking) ③ Isolation Level 설정 ④ Distributed Transaction 지원 ⑤ 트랜잭션 전파 설정 (Propagation Required, Supports 등) ⑥ 트랜잭션 Timeout 관리 ⑦ ACID 속성 보장 ⑧ 이벤트 기반 트랜잭션 처리 ⑨ 트랜잭션 모니터링 ⑩ 분산 환경에서의 트랜잭션 관리 |
| 서비스 모듈화 | 서비스 간 독립적 처리 | ① 비즈니스 로직의 모듈화 ② 인터페이스 추상화 ③ 서비스 간 종속성 최소화 ④ 모듈 단위 테스트 ⑤ 독립 배포 가능 ⑥ 서비스 간 메시지 처리 ⑦ 통합 테스트 케이스 작성 ⑧ REST API 문서화 ⑨ 응답 시간 최적화 ⑩ 확장성을 고려한 서비스 설계 |
| 예외 처리 | 비정상 동작 시 오류 관리 | ① Custom Exception 정의 ② Global Exception Handler 설정 ③ 에러 메시지 로깅 ④ Validation 예외 처리 ⑤ 비즈니스 로직 예외 처리 ⑥ DB 예외 처리 ⑦ 네트워크 예외 처리 ⑧ 장애 알림 및 대응 ⑨ 재시도 로직 설정 ⑩ REST API 에러 응답 표준화 |
| 계층 간 통신 | Presentation → Business → Data Layer | ① Data Transfer Object(DTO) 사용 ② API Response 구조 정의 ③ Controller와 Service 간 데이터 전달 최적화 ④ 인터페이스 기반 의존성 주입 ⑤ 비동기 요청 처리 ⑥ SOAP 및 REST 통신 지원 ⑦ 메시지 직렬화 및 역직렬화 ⑧ HTTP Method 최적화 ⑨ 단일 책임 원칙 준수 ⑩ 로깅 및 트래킹 |

### **MSA (Microservice Architecture) 설계**

#### **개요**

* 마이크로서비스 아키텍처는 독립적인 서비스 단위로 구성되어 개별적으로 배포되고 관리됩니다.
* 서비스 간 결합도가 낮아 장애 격리와 확장성이 뛰어나며, 개별적으로 스케일링이 가능합니다.
* CI/CD 파이프라인을 통해 무중단 배포가 가능하고, API Gateway를 통해 외부 접근을 제어합니다.

#### **주요 설계 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 서비스 분리 | 각 도메인을 개별 서비스로 분리 | ① 도메인 기반 서비스 설계 ② 독립적 배포 가능 ③ 서비스 간 결합도 최소화 ④ 데이터베이스 분리 (DaaS) ⑤ RESTful API 제공 ⑥ 비동기 메시지 처리 ⑦ 서비스 간 인증 및 보안 ⑧ 독립 테스트 및 배포 ⑨ 클라우드 네이티브 설계 ⑩ 무중단 배포 전략 |
| API Gateway | 서비스 간 접근 제어 및 요청 처리 | ① 인증 및 권한 관리 ② 라우팅 설정 ③ Rate Limiting 설정 ④ CORS 정책 관리 ⑤ 로깅 및 모니터링 ⑥ API 버전 관리 ⑦ Circuit Breaker 적용 ⑧ 트래픽 분산 처리 ⑨ 캐싱 설정 ⑩ 보안 정책 적용 (OAuth2, JWT) |
| Circuit Breaker | 장애 전파 방지 | ① 장애 시 빠른 차단 ② Timeout 설정 ③ Fallback 처리 ④ 서비스 복구 시 자동 재연결 ⑤ 장애 탐지 로깅 ⑥ 임계값 설정 ⑦ 장애 분석 리포팅 ⑧ 동적 라우팅 처리 ⑨ 서비스 헬스 체크 ⑩ Circuit 상태 모니터링 |
| Configuration 관리 | 환경 설정 중앙 집중화 | ① Spring Cloud Config ② Consul, Etcd 연동 ③ 프로파일별 설정 관리 ④ 배포 환경에 따른 Config 관리 ⑤ 동적 설정 변경 지원 ⑥ 중앙 집중 로깅 ⑦ Config 변경 시 알림 ⑧ Secure Vault 연동 ⑨ 버전 관리 ⑩ 모니터링 및 추적 |
| 무중단 배포 | 서비스 중단 없는 배포 | ① Blue-Green Deployment ② Canary Release 관리 ③ Zero Downtime 배포 ④ Rolling Update 설정 ⑤ Health Check 통과 시 배포 ⑥ 배포 실패 시 자동 롤백 ⑦ 배포 속도 최적화 ⑧ 무중단 스케일링 ⑨ 트래픽 라우팅 최적화 ⑩ A/B Testing 지원 |
| 메시지 큐 사용 | 비동기 데이터 처리 | ① Kafka, RabbitMQ 활용 ② 이벤트 기반 메시징 ③ 비동기 처리 최적화 ④ Pub/Sub 모델 구현 ⑤ 메시지 중복 방지 ⑥ 메시지 TTL 설정 ⑦ 메시지 재처리 로직 ⑧ QoS 설정 ⑨ Dead Letter Queue 관리 ⑩ 메시지 순서 보장 |

### **비동기 처리 설계**

#### **개요**

* 비동기 처리 구조는 대량 트랜잭션을 빠르게 처리하기 위한 최적화된 설계입니다.
* 메시지 큐를 활용하여 이벤트 기반으로 데이터를 처리하며, 요청에 대한 응답을 기다리지 않고 비동기로 처리하여 **성능 향상**과 **응답 시간 단축**을 목표로 합니다.
* 주요 기술로는 **Kafka**, **RabbitMQ**, **AWS SQS** 등이 있으며, 장애 발생 시 메시지 손실을 최소화하기 위해 **Dead Letter Queue(DLQ)** 구조를 사용합니다.

#### **주요 설계 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 메시지 큐 설계 | 대량 데이터 비동기 처리 | ① Kafka, RabbitMQ 사용 ② Topic & Queue 설정 ③ Partition 관리 ④ Consumer Group 설정 ⑤ Producer와 Consumer의 비동기 처리 ⑥ 메시지 중복 방지 ⑦ 메시지 순서 보장 ⑧ 메시지 TTL(Time to Live) 설정 ⑨ 메시지 전달 보장(At least once, Exactly once) ⑩ Dead Letter Queue 설정 |
| 비동기 트랜잭션 관리 | 비동기 방식으로 트랜잭션 처리 | ① Saga 패턴 적용 ② 이벤트 소싱 기반 처리 ③ 데이터 일관성 보장 ④ 트랜잭션 격리 수준 설정 ⑤ 보상 트랜잭션 처리(Compensation Transaction) ⑥ 분산 트랜잭션 처리 ⑦ 이벤트 발행 및 구독 ⑧ 롤백 시나리오 설정 ⑨ 비동기 커밋 처리 ⑩ 모니터링 및 로깅 |
| 이벤트 드리븐 아키텍처 | 이벤트 기반의 비동기 통신 | ① Publish/Subscribe 모델 ② 이벤트 버스(Event Bus) 사용 ③ 이벤트 핸들러 구성 ④ Event Router 설정 ⑤ 장애 발생 시 재처리 ⑥ 멀티 캐스팅 처리 ⑦ 메시지 브로커와 통합 ⑧ Event Store 구성 ⑨ 비동기 로깅 ⑩ 실시간 알림 처리 |
| 지연 큐(Delay Queue) 설계 | 특정 시간 이후에 처리되는 메시지 | ① 메시지 전송 지연 설정 ② 처리 시간 스케줄링 ③ 우선순위 메시지 처리 ④ Redis Delay Queue 사용 ⑤ AWS SQS Delay 적용 ⑥ 타임아웃 설정 ⑦ 큐 상태 모니터링 ⑧ 실패 시 재처리 ⑨ 메시지 만료 관리 ⑩ DLQ에 이관 |
| QoS (Quality of Service) 설정 | 메시지 전송 품질 보장 | ① 메시지 전달 보장(At most once, At least once) ② 중복 수신 방지 ③ 메시지 순서 유지 ④ 네트워크 손실 복구 ⑤ 재전송 정책 설정 ⑥ 메시지 처리 속도 제어 ⑦ 서비스 품질 측정 ⑧ 트래픽 스로틀링 ⑨ Load Balancing 설정 ⑩ SLA(Service Level Agreement) 관리 |

### **트랜잭션 관리 설계**

#### **개요**

* 트랜잭션 관리 설계는 금융 거래의 무결성을 보장하고, 동시성 처리 시 데이터의 일관성을 유지하기 위한 중요한 설계입니다.
* ACID 원칙을 준수하여 모든 거래가 정확하게 처리되며, 실패 시 Rollback이 보장되어야 합니다.
* **분산 트랜잭션**, **2PC (Two-Phase Commit)**, **Saga Pattern** 등이 적용됩니다.

#### **주요 설계 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| ACID 원칙 준수 | 트랜잭션의 원자성, 일관성, 격리성, 지속성을 보장 | ① 원자성(Atomicity): All or Nothing ② 일관성(Consistency): 정합성 유지 ③ 격리성(Isolation): 트랜잭션 간 영향 방지 ④ 지속성(Durability): 커밋된 데이터 영구 보장 ⑤ Isolation Level 설정 (Read Committed, Serializable 등) ⑥ 커밋 롤백 처리 ⑦ 동시성 처리 보장 ⑧ 트랜잭션 Lock 관리 ⑨ Snapshot Isolation 적용 ⑩ 트랜잭션 로그 추적 |
| 분산 트랜잭션 관리 | 다중 서비스 간 트랜잭션 관리 | ① 2PC (Two-Phase Commit) 적용 ② XA 트랜잭션 지원 ③ JTA(Java Transaction API) 사용 ④ 글로벌 트랜잭션 관리 ⑤ 상태 동기화 처리 ⑥ 중간 실패 시 보상 처리 ⑦ 롤백 일관성 유지 ⑧ 트랜잭션 격리 수준 설정 ⑨ 다중 DB 트랜잭션 처리 ⑩ 장애 발생 시 자동 복구 |
| Saga Pattern 적용 | 비동기 트랜잭션 복구 전략 | ① 보상 트랜잭션 설정 ② 이벤트 기반 트랜잭션 처리 ③ 장애 시 Rollback 흐름 구성 ④ 서비스 간 트랜잭션 상태 동기화 ⑤ Event Choreography 패턴 ⑥ Event Orchestration 패턴 ⑦ 중단 시 재처리 로직 ⑧ 이벤트 로그 보관 ⑨ 트랜잭션 상태 관리 ⑩ 비동기 보상 처리 |

### **서비스 디스커버리 설계**

#### **개요**

* 서비스 디스커버리는 MSA(Microservice Architecture) 환경에서 각 서비스의 위치를 자동으로 탐색하고 연결을 유지하는 구조입니다.
* 서비스 등록(Register)과 탐색(Discovery)을 통해 동적으로 라우팅을 설정하며, 확장성과 장애 복구에 강점이 있습니다.
* 주요 기술로는 **Spring Cloud Eureka**, **Consul**, **Zookeeper** 등이 있으며, 로드 밸런싱과 함께 활용됩니다.

#### **주요 설계 내용**

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 서비스 레지스트리 | 서비스의 위치 정보를 등록하고 탐색 | ① Spring Cloud Eureka 설정 ② Consul 연동 ③ Zookeeper 클러스터 구성 ④ Service Instance 등록 ⑤ Heartbeat 체크 ⑥ 동적 IP 업데이트 ⑦ 서비스 상태 모니터링 ⑧ Health Check 주기 관리 ⑨ 서비스 종료 시 자동 해제 ⑩ 멀티 클러스터 연동 |
| 서비스 탐색 | 클라이언트가 동적으로 서비스 위치를 탐색 | ① Eureka Client 설정 ② Consul Client 설정 ③ DNS 기반 탐색 ④ API Gateway와 연계 ⑤ Circuit Breaker 적용 ⑥ 장애 발생 시 대체 서비스 탐색 ⑦ 글로벌 서비스 탐색 ⑧ 지역별 로드 밸런싱 ⑨ 라우팅 정책 설정 ⑩ Latency 기반 탐색 최적화 |
| 로드 밸런싱 | 트래픽을 여러 서비스 인스턴스로 분산 | ① Round Robin 방식 ② Least Connection 방식 ③ IP Hash 방식 ④ Weighted 방식 ⑤ Service Discovery와 연동 ⑥ Spring Cloud LoadBalancer 설정 ⑦ Ribbon, Netflix Zuul 연동 ⑧ Session Sticky 설정 ⑨ 장애 탐지 시 Failover 처리 ⑩ Auto Scaling 연동 |
| Health Check | 서비스의 상태를 주기적으로 체크 | ① Health Endpoint 설정 ② Spring Actuator 연동 ③ Custom Health Indicator 설정 ④ 서비스 생존 여부 확인 ⑤ Eureka Heartbeat 모니터링 ⑥ Consul Health Check 통합 ⑦ Circuit Breaker와 연동 ⑧ 서비스 복구 시 알림 발송 ⑨ 장애 탐지 시 라우팅 변경 ⑩ 상태 모니터링 대시보드 |
| 서비스 동적 확장 | 서비스 상태에 따른 동적 확장 | ① Auto Scaling 설정 ② Spring Cloud Gateway와 연동 ③ 서비스 증가 시 동적 등록 ④ 서비스 축소 시 자동 해제 ⑤ 트래픽 부하 분산 최적화 ⑥ Kubernetes Horizontal Pod Autoscaler 사용 ⑦ 무중단 스케일링 ⑧ Multi-AZ (Availability Zone) 지원 ⑨ Scaling Event 로그 관리 ⑩ Capacity Planning 최적화 |

### ****API Gateway 설계****

#### ****3.4.6.1 개요****

* API Gateway는 클라이언트와 여러 마이크로서비스 간의 통신을 중앙에서 관리하는 진입점 역할을 합니다.
* 인증(Authentication), 권한 관리(Authorization), 요청 라우팅(Routing), 모니터링(Monitoring)을 통합적으로 처리합니다.
* 주요 기술로는 **Spring Cloud Gateway**, **Kong**, **AWS API Gateway** 등이 있으며, **Rate Limiting**과 **Circuit Breaker**를 통해 보호합니다.

#### 주요 설계 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 인증 및 권한 관리 | API 접근에 대한 인증과 권한 부여 | ① OAuth2.0 인증② JWT(Json Web Token) 사용③ API Key 관리④ 역할 기반 접근 제어(RBAC)⑤ Spring Security 연동⑥ SSO(Single Sign-On) 설정⑦ 접근 토큰 발급⑧ 만료 시간 관리⑨ 토큰 리프레시 정책⑩ 다중 인증(MFA) 설정 |
| 라우팅 설정 | 클라이언트 요청을 올바른 서비스로 전달 | ① URL Path 기반 라우팅② HTTP Method 기반 라우팅③ 동적 라우팅 설정④ Eureka 서비스 탐색 연동⑤ Request Path Rewrite⑥ Failover 라우팅⑦ 로드 밸런싱 설정⑧ Service Discovery 연동⑨ 우선순위 기반 라우팅⑩ 상태 모니터링 |
| Rate Limiting | 과도한 요청으로부터 보호 | ① IP 기반 Rate Limiting② User ID 기반 Throttling③ Burst Limit 설정④ Time Window 관리⑤ 동적 요청 제한 설정⑥ 지역별 Rate Limit 설정⑦ API Key 별 Rate 설정⑧ QoS(Quality of Service) 적용⑨ 오버로드 방지 로깅⑩ Circuit Breaker 연동 |
| 로깅 및 모니터링 | API 요청과 응답을 추적하고 모니터링 | ① 요청 및 응답 로깅② 에러 발생 시 로그 기록③ 요청 시간 추적④ HTTP Status Code 분석⑤ API 사용량 통계⑥ Spring Actuator 연동⑦ 트랜잭션 추적(Distributed Tracing)⑧ 중앙 집중식 로그 관리⑨ 대시보드 시각화⑩ 로그 분석 및 경고 발송 |
| 캐싱 최적화 | 반복되는 요청에 대한 응답 시간 단축 | ① In-Memory Cache 설정② Redis, Memcached 활용③ GET 요청 캐싱④ Expiration Time 설정⑤ 지역별 캐싱 정책⑥ 멀티 리전 캐싱⑦ 캐시 무효화 전략⑧ Lazy Loading 적용⑨ 캐시 미스(Cache Miss) 처리⑩ 캐시 적중률 최적화 |

### ****이벤트 소싱(Event Sourcing) 설계****

#### ****개요****

* 이벤트 소싱(Event Sourcing)은 시스템의 상태를 데이터베이스에 현재 상태로 저장하는 대신 **발생한 모든 이벤트의 이력**으로 저장하는 구조입니다.
* 이를 통해 데이터의 모든 변경 이력을 추적할 수 있으며, 복구 시 모든 이벤트를 순차적으로 실행하면 **정확한 상태를 복원**할 수 있습니다.
* 이벤트는 **Event Store**에 저장되며, \*\*CQRS(Command Query Responsibility Segregation)\*\*와 결합하여 대량 조회 최적화에 유리합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 이벤트 로그 저장 | 모든 상태 변경을 이벤트 로그로 저장 | ① 이벤트 발생 시 Event Store에 기록 ② 이벤트 ID 생성 및 저장 ③ 발생 순서 보장 ④ 이벤트 타입 구분 ⑤ 이벤트 메타데이터 저장 ⑥ 이력 추적 가능 ⑦ 이벤트 삭제 방지 ⑧ 타임스탬프 기록 ⑨ 이벤트 스키마 관리 ⑩ 보안 및 암호화 처리 |
| 이벤트 재생 | 이벤트 로그를 순차적으로 재생하여 상태 복원 | ① 최신 상태 계산 ② 특정 시점으로 복구 ③ 이벤트 순서 보장 ④ 장애 발생 시 Rollback ⑤ 트랜잭션 일관성 유지 ⑥ 시간 여행(Time Travel) 기능 ⑦ Partial Replay 지원 ⑧ 이벤트 Conflict 해결 ⑨ 분산 환경에서 동기화 ⑩ 실패 시 재처리 |
| Event Store 구성 | 이벤트 로그를 효율적으로 관리하기 위한 저장소 | ① Kafka, Event Store 사용 ② Write-Ahead Log 관리 ③ Append-Only 구조 사용 ④ 이벤트 스냅샷 지원 ⑤ 이벤트 만료 처리 ⑥ 분산 Event Store 구성 ⑦ 다중 파티션 관리 ⑧ 복제본 관리(Replication) ⑨ 이벤트 압축 처리 ⑩ 성능 최적화 |
| CQRS 연동 | 이벤트 기반 조회 최적화 | ① Command와 Query 분리 ② Read Model 최적화 ③ 별도의 조회 DB 사용 ④ Query 성능 최적화 ⑤ 다중 리전 조회 지원 ⑥ 비동기 처리 ⑦ 실시간 데이터 반영 ⑧ Projection 생성 ⑨ 이벤트 기반 조회 캐싱 ⑩ 데이터 정합성 보장 |
| 장애 복구 지원 | 이벤트 이력 기반으로 빠르게 복구 | ① 이벤트 리플레이로 상태 복원 ② Event Store 백업 ③ Snapshot 사용 최적화 ④ 이벤트 누락 시 재처리 ⑤ 장애 탐지 시 재생성 ⑥ 데이터 정합성 유지 ⑦ 트랜잭션 로그 분석 ⑧ 분산 환경에서 복구 시 동기화 ⑨ 이벤트 Conflict 해결 ⑩ 장애 대응 시나리오 적용 |

### ****CQRS (Command Query Responsibility Segregation) 설계****

#### ****개요****

* CQRS(Command Query Responsibility Segregation)는 \*\*명령(Command)\*\*과 \*\*조회(Query)\*\*를 분리하여 설계하는 구조입니다.
* 명령은 상태 변경을 담당하고 조회는 데이터 검색에 최적화된 구조를 사용합니다.
* 이 접근 방식은 **읽기와 쓰기 모델의 독립적 최적화**가 가능하며, 이벤트 소싱과 결합 시 높은 성능을 발휘합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 명령 처리(Command Handling) | 상태 변경을 수행하는 명령 처리 | ① Create, Update, Delete 처리 ② 비동기 이벤트 발생 ③ Command Handler 구성 ④ 상태 변경 이벤트 발행 ⑤ 이벤트 순서 보장 ⑥ Lock-Free 처리 ⑦ 비정상 Command Rollback ⑧ 트랜잭션 관리 ⑨ Command Queue 관리 ⑩ 장애 발생 시 재처리 |
| 조회 처리(Query Handling) | 조회 성능을 최적화한 설계 | ① 별도 Read Model 사용 ② Projection 생성 ③ 인덱스 최적화 ④ Read Cache 설정 ⑤ Data Mart와 통합 ⑥ Full Text Search 지원 ⑦ 동시 조회 처리 최적화 ⑧ Pagination 처리 ⑨ 분산 조회 최적화 ⑩ 캐싱 정책 최적화 |
| Eventual Consistency | 읽기와 쓰기 사이의 일관성 보장 | ① 이벤트 전파를 통한 최종 일관성 보장 ② Read Model의 지연 반영 처리 ③ 데이터 Conflict 해결 ④ 이벤트 버스 통신 ⑤ 분산 트랜잭션 최적화 ⑥ 스냅샷 기반 복원 ⑦ Lock-Free Read 처리 ⑧ 비동기 동기화 ⑨ 장애 발생 시 재동기화 ⑩ 데이터 정합성 체크 |
| CQRS 데이터 분리 | 명령과 조회의 데이터베이스 분리 | ① Command DB와 Query DB 구분 ② 별도의 Schema 관리 ③ Read-Only DB 구성 ④ 복제본(Replica) 활용 ⑤ 캐싱 레이어 적용 ⑥ 실시간 동기화 설정 ⑦ DB Sharding 설정 ⑧ 분산 환경에서의 데이터 동기화 ⑨ 장애 시 Failover 지원 ⑩ 확장성 최적화 |
| 확장성 및 성능 최적화 | 대규모 트래픽 처리 시 확장성 보장 | ① 이벤트 기반 Scaling ② Auto Scaling 설정 ③ Partitioning 최적화 ④ 멀티 리전 데이터 복제 ⑤ Latency 최적화 ⑥ 트래픽 분산 처리 ⑦ 실시간 상태 동기화 ⑧ Event Store의 분산 처리 ⑨ Gateway 최적화 ⑩ Service Discovery 연동 |

### ****로드 밸런싱 설계****

#### ****개요****

* 로드 밸런싱은 다수의 서버로 네트워크 트래픽을 분산하여 **시스템 성능과 가용성을 최적화**하는 구조입니다.
* 요청이 특정 서버에 집중되는 것을 방지하고, 장애 발생 시 자동으로 다른 서버로 트래픽을 분배하여 **서비스의 연속성**을 보장합니다.
* 주요 기술로는 **Nginx**, **HAProxy**, **AWS ELB**, **Kubernetes Ingress** 등이 있으며, **Round Robin, Least Connection, IP Hash**와 같은 알고리즘을 활용합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 로드 밸런싱 알고리즘 | 트래픽 분산을 위한 최적화된 알고리즘 사용 | ① Round Robin 방식 ② Least Connection 방식 ③ IP Hash 기반 라우팅 ④ Weighted Round Robin 설정 ⑤ Least Response Time 적용 ⑥ Random Load Balancing ⑦ Sticky Session 처리 ⑧ Geo-based Load Balancing ⑨ Health Check 기반 라우팅 ⑩ Fault Tolerant 분산 처리 |
| Health Check | 서비스 인스턴스의 상태를 주기적으로 검사 | ① HTTP Health Check 설정 ② TCP Health Check 설정 ③ HTTPS Health Check 설정 ④ Heartbeat Monitor 구성 ⑤ Graceful Shutdown 관리 ⑥ 장애 발생 시 자동 Failover ⑦ 정상 복구 시 자동 등록 ⑧ Health 상태 대시보드 연동 ⑨ 복구 시 트래픽 재분배 ⑩ 모니터링 알림 설정 |
| 글로벌 로드 밸런싱 | 글로벌 서비스 배포 시 트래픽 분산 | ① Multi-Region 로드 밸런싱 ② Latency 기반 라우팅 ③ Geo-DNS 라우팅 설정 ④ AWS Global Accelerator 연동 ⑤ GCP Cloud Load Balancing 설정 ⑥ Azure Traffic Manager 사용 ⑦ 지역 간 Failover 처리 ⑧ Disaster Recovery 연동 ⑨ Multi-Region Failback 설정 ⑩ Cross-Region Replication 최적화 |
| 세션 스티키(Session Sticky) | 사용자 세션이 동일 서버에 유지되도록 설정 | ① Session Affinity 설정 ② Cookie 기반 세션 유지 ③ IP Hash 기반 스티키 세션 ④ HAProxy Stick Table 사용 ⑤ Spring Session 클러스터링 ⑥ Redis 세션 저장소 설정 ⑦ 세션 만료 시 Graceful Handling ⑧ Cluster 간 세션 동기화 ⑨ 사용자 세션 복구 ⑩ 무중단 세션 유지 |
| Auto Scaling 연동 | 트래픽 증가에 따른 동적 확장 | ① Kubernetes Horizontal Pod Autoscaler 설정 ② AWS Auto Scaling 설정 ③ GCP Autoscaler 설정 ④ Threshold 기반 Scale Out ⑤ Load Balancing 연동 ⑥ 스케일링 시 무중단 처리 ⑦ Scale-In 정책 설정 ⑧ 리소스 최적화 ⑨ 비용 최적화 관리 ⑩ 장애 시 자동 복구 |

### ****장애 복구 및 이중화 설계****

#### ****개요****

* 장애 복구 및 이중화 설계는 **시스템 중단 없이 연속적인 서비스 제공**을 목표로 합니다.
* 장애 발생 시 신속하게 감지하고, 자동으로 다른 노드로 전환(Failover)하여 서비스를 지속합니다.
* 이중화 구조로는 **Active-Active**, **Active-Passive** 방식이 있으며, 주요 기술로는 **Zookeeper**, **Consul**, **Kubernetes HA**가 활용됩니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 이중화 구조 설계 | Active-Active, Active-Passive 구조로 고가용성 보장 | ① Active-Active 클러스터 구성 ② Active-Passive 설정 ③ Failover 시 자동 전환 ④ 데이터 동기화 관리 ⑤ 장애 탐지 시 Traffic Switch ⑥ 데이터 Replication 설정 ⑦ Heartbeat Monitor 구성 ⑧ Disaster Recovery 전략 ⑨ 멀티 리전 이중화 ⑩ Split-Brain 방지 |
| 장애 탐지 및 대응 | 장애 발생 시 빠른 감지와 대응 | ① 장애 탐지 모니터링 설정 ② 실시간 알림 발송 ③ 장애 유형 분류 ④ 자동 롤백 처리 ⑤ 로그 분석 및 원인 추적 ⑥ Self-Healing 설정 ⑦ 장애 발생 시 Auto Scaling ⑧ 자동화된 서비스 복구 ⑨ Recovery Point Objective(RPO) 설정 ⑩ Recovery Time Objective(RTO) 설정 |
| 데이터 백업 및 복구 | 데이터 손실 방지와 빠른 복구 | ① Full Backup 설정 ② Incremental Backup 구성 ③ Differential Backup 관리 ④ Disaster Recovery Site 설정 ⑤ Offsite Backup 저장 ⑥ 암호화된 백업 처리 ⑦ 백업 데이터 무결성 검증 ⑧ 복구 테스트 자동화 ⑨ 백업 주기 설정 ⑩ 클라우드 백업 최적화 |
| 스냅샷 및 복제 | 장애 시 빠른 상태 복원 | ① 데이터베이스 스냅샷 생성 ② Volume Snapshot 설정 ③ Kubernetes PV Snapshot 활용 ④ 시점 복구 지원 ⑤ Multi-Region Snapshot 연동 ⑥ Snapshot Consistency 보장 ⑦ 주기적 Snapshot 백업 ⑧ 장애 시 데이터 복구 ⑨ 암호화된 스냅샷 생성 ⑩ Auto-Snapshot 설정 |
| DR(Disaster Recovery) 설계 | 재해 발생 시 서비스 연속성 보장 | ① DR Site 구축 ② Multi-AZ 배포 ③ RPO/RTO 최적화 ④ 데이터 동기화 보장 ⑤ Cloud Region 간 Active-Passive 설정 ⑥ Region Failover 처리 ⑦ DR 연습 테스트 시나리오 ⑧ 데이터 복구 검증 ⑨ DNS Failover 설정 ⑩ 전환 시 Latency 최적화 |

## **실사례 (설계 항목 예시 30개)**

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| 계층형 아키텍처 설계 | Presentation, Business, Data Layer 구조화 | 인터넷 뱅킹, 모바일 뱅킹 | ① Controller-Service-DAO 구조② MVC 패턴 적용③ 계층 간 종속성 관리④ 트랜잭션 전파 설정⑤ 세션 관리 최적화⑥ 비즈니스 로직 캡슐화⑦ DAO에서 직접 SQL 처리⑧ 인터페이스 통합 관리⑨ 비즈니스 모듈화⑩ 예외 처리 관리 |
| MSA (Microservice Architecture) 설계 | 독립적인 서비스 구조로 확장성 극대화 | 글로벌 결제 시스템, 대출 관리 | ① 서비스 분리 및 배포② 서비스 간 메시지 큐 연동③ 독립적인 데이터베이스 관리④ 장애 격리 처리⑤ API Gateway 설정⑥ 서비스 간 데이터 동기화⑦ Configuration 관리⑧ 무중단 배포 지원⑨ Circuit Breaker 적용⑩ 서비스 디스커버리 |
| 비동기 처리 설계 | 트랜잭션 처리 속도 최적화 | 실시간 결제 승인, 예약 처리 | ① 메시지 큐 (Kafka, RabbitMQ) 구성② Async Task 처리③ Event-Driven 처리④ 트랜잭션 분리⑤ 대기 시간 최소화⑥ 순서 보장 처리⑦ 메시지 재처리⑧ Latency 최적화⑨ 비동기 로깅⑩ 비동기 에러 핸들링 |
| 트랜잭션 관리 설계 | ACID 원칙 준수와 동시성 처리 보장 | 실시간 이체, 대량 거래 처리 | ① 커밋 및 롤백 처리② 동시성 제어 (Optimistic/Pessimistic Locking)③ Isolation Level 설정④ Distributed Transaction 지원⑤ 트랜잭션 전파 설정 (Propagation Required, Supports 등)⑥ 트랜잭션 Timeout 관리⑦ ACID 속성 보장⑧ 이벤트 기반 트랜잭션 처리⑨ 트랜잭션 모니터링⑩ 분산 환경에서의 트랜잭션 관리 |
| 서비스 디스커버리 설계 | 마이크로서비스 간 동적 라우팅 및 탐색 | API Gateway, 서비스 메시 | ① Eureka, Consul 사용② Service Registry 등록③ Heartbeat 체크④ 동적 IP 업데이트⑤ DNS 기반 탐색⑥ 로드 밸런싱과 연동⑦ 장애 시 Failover 처리⑧ 멀티 리전 탐색⑨ Health Check 설정⑩ 서비스 상태 모니터링 |
| API Gateway 설계 | 외부 서비스와의 통신 최적화 | 핀테크 연동, 결제 시스템 | ① 인증 및 권한 관리 (OAuth2.0, JWT)② 라우팅 설정③ Rate Limiting④ 로깅 및 모니터링⑤ API 버전 관리⑥ Circuit Breaker 설정⑦ 캐싱 설정⑧ CORS 정책 관리⑨ 트래픽 분산 처리⑩ 보안 정책 적용 |
| 이벤트 소싱 설계 | 상태 변경 이력을 저장하여 데이터 정합성 유지 | 거래 내역 추적, 정산 시스템 | ① 이벤트 발생 시 Event Store 기록② 트랜잭션 변경 내역 관리③ 이벤트 재생으로 상태 복원④ 특정 시점으로 복구 가능⑤ Event Store의 장애 복구⑥ CQRS 연동⑦ 장애 발생 시 Rollback 처리⑧ 이벤트 Conflict 관리⑨ Snapshot 생성⑩ Event Handler 최적화 |
| CQRS 설계 | 명령과 조회의 처리 경로를 분리 | 대량 조회 최적화, 대기 시간 감소 | ① Command와 Query의 분리② 별도의 Read Model 사용③ 인덱스 최적화④ 이벤트 전파를 통한 일관성 유지⑤ Full Text Search 최적화⑥ 동시 조회 처리 최적화⑦ 캐싱 정책 설정⑧ CQRS 데이터 분리⑨ 이벤트 기반 캐싱⑩ 조회 성능 최적화 |
| 로드 밸런싱 설계 | 트래픽을 여러 서버로 분산하여 처리 성능 향상 | 인터넷 뱅킹, 결제 시스템 | ① Round Robin 방식② Least Connection 방식③ IP Hash 기반 라우팅④ Weighted Round Robin 설정⑤ Geo-based Load Balancing⑥ Health Check 설정⑦ Session Sticky 처리⑧ Auto Scaling 연동⑨ 글로벌 로드 밸런싱⑩ 장애 시 Failover 처리 |
| 장애 복구 및 이중화 설계 | 시스템 장애 시 빠른 복구를 보장 | 금융 트랜잭션 이중화, 데이터 복구 | ① Active-Active, Active-Passive 구성② Heartbeat 모니터링③ 장애 발생 시 자동 전환④ 데이터 복제 및 동기화⑤ DR(Disaster Recovery) 구축⑥ Multi-AZ 이중화⑦ 자동 롤백 처리⑧ 장애 탐지 및 알림⑨ 스냅샷 복구⑩ 멀티 리전 장애 대응 |

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| **계층형 아키텍처 설계** | Presentation, Business, Data Layer 구조화 | 인터넷 뱅킹, 모바일 뱅킹 | ① Controller-Service-DAO 구조 ② MVC 패턴 적용 ③ 계층 간 종속성 관리 ④ 트랜잭션 전파 설정 ⑤ 세션 관리 최적화 ⑥ 비즈니스 로직 캡슐화 ⑦ DAO에서 직접 SQL 처리 ⑧ 인터페이스 통합 관리 ⑨ 비즈니스 모듈화 ⑩ 예외 처리 관리 |
| **MSA (Microservice Architecture) 설계** | 독립적인 서비스 구조로 확장성 극대화 | 글로벌 결제 시스템, 대출 관리 | ① 서비스 분리 및 배포 ② 서비스 간 메시지 큐 연동 ③ 독립적인 데이터베이스 관리 ④ 장애 격리 처리 ⑤ API Gateway 설정 ⑥ 서비스 간 데이터 동기화 ⑦ Configuration 관리 ⑧ 무중단 배포 지원 ⑨ Circuit Breaker 적용 ⑩ 서비스 디스커버리 |

## **마무리말**

애플리케이션 아키텍처 설계는 금융 시스템의 안정성과 확장성을 보장하기 위한 핵심적인 역할을 수행합니다.  
특히, 금융 시스템은 대규모 트랜잭션 처리와 보안성이 요구되기 때문에 **MSA 구조, 비동기 처리, 트랜잭션 관리, 로드 밸런싱, 장애 복구 및 이중화 설계**가 철저하게 이루어져야 합니다.

각 설계 항목에서 다음과 같은 핵심적인 목표를 달성해야 합니다:

1. **안정성 확보**
   * 계층형 아키텍처와 트랜잭션 관리 설계를 통해 데이터의 정합성을 보장하고, 장애 발생 시 신속한 복구가 가능해야 한다.
2. **확장성과 유연성 극대화**
   * MSA 구조를 통해 서비스 간 결합도를 최소화하고, 독립적인 확장 및 배포가 가능하도록 설계해야 한다.
3. **실시간 처리 최적화**
   * 비동기 처리와 이벤트 소싱 구조를 통해 실시간 데이터 반영이 가능하며, 거래 지연 없이 빠르게 처리될 수 있어야 한다.
4. **장애 복구와 이중화**
   * 로드 밸런싱을 통해 서비스 부하를 분산시키고, 이중화 설계로 장애 시에도 빠른 복구가 가능하도록 해야 한다.
5. **보안성과 무결성 보장**
   * API Gateway와 서비스 디스커버리를 통해 외부 접근을 통제하고, 모든 데이터 전송에 대한 보안을 강화해야 한다.

## ****시사점****

* **금융 시스템의 특성에 최적화된 설계 구조**
* 금융 시스템은 수많은 거래가 동시에 발생하는 환경이므로, MSA 구조와 비동기 처리 설계가 필수적이다.
* 특히, 트랜잭션 처리 속도와 무결성 보장은 고객 신뢰도에 직결되므로 이를 최적화할 수 있는 구조가 필요하다.
* **글로벌 서비스 확장 및 장애 대응력**
* 로드 밸런싱과 글로벌 디스커버리 구조를 통해 **Multi-Region** 서비스가 가능하며, 지리적 장애 발생 시에도 빠르게 대응할 수 있다.
* AWS Global Accelerator, Azure Traffic Manager 등을 활용한 글로벌 확장은 금융 서비스의 가용성을 극대화한다.
* **이벤트 소싱과 CQRS를 통한 무결성 보장**
* 이벤트 소싱(Event Sourcing)과 CQRS(Command Query Responsibility Segregation) 구조는 모든 데이터 변경을 이벤트로 기록하여 완전한 추적이 가능하다.
* 장애 발생 시에도 모든 이벤트를 재생하여 정확한 복구가 가능하므로 데이터 무결성을 보장할 수 있다.
* **서비스 확장성과 유지보수의 용이성**
* MSA 구조는 개별 서비스 단위로 개발 및 배포가 가능하므로, 특정 서비스의 확장이나 수정이 전체 시스템에 영향을 주지 않는다.
* 이를 통해 유지보수 비용이 절감되며, CI/CD 파이프라인에 통합하면 배포 시간도 단축된다.
* **클라우드 네이티브 환경과의 최적화**
* Kubernetes 기반의 오토스케일링, 로드 밸런싱, 이중화 설계를 통해 클라우드 환경에 최적화된 확장성과 복구 능력을 갖추게 된다.
* 멀티 클러스터와 분산 데이터베이스를 통해 장애 발생 시에도 서비스 중단 없이 대응할 수 있다.

# ****4. 애플리케이션 아키텍처 구현 전략****

## ****개요****

* 애플리케이션 아키텍처 구현 전략은 금융 시스템의 확장성과 안정성을 극대화하기 위한 **설계된 아키텍처를 실제 환경에 반영**하는 단계입니다.
* 주요 목표는 설계된 계층 구조, MSA(Microservice Architecture), 비동기 처리, 트랜잭션 관리, 로드 밸런싱, 장애 복구를 최적화하여 실제 시스템에 안정적으로 적용하는 것입니다.
* 이 구현 전략을 통해 **자동화된 배포, 무중단 서비스, 보안 최적화**를 달성합니다.

## ****주요 내용 요약****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| API Gateway 구현 | 클라이언트 요청을 적절한 서비스로 라우팅 | 핀테크 연동, 대규모 금융 트랜잭션 |
| MSA 구현 | 서비스 단위의 독립적 배포 및 확장 | 실시간 결제 시스템, 대출 관리 |
| 비동기 메시지 처리 | 이벤트 기반 처리로 트랜잭션 최적화 | 예약 처리, 알림 서비스 |
| 트랜잭션 관리 구현 | 금융 거래의 무결성과 일관성 보장 | 실시간 송금, 온라인 결제 |
| 서비스 디스커버리 구성 | 동적 탐색 및 로드 밸런싱 | 금융 API Gateway, 외부 연동 |
| 로드 밸런서 설정 | 다중 서버에 트래픽 분산 | 인터넷 뱅킹, 모바일 결제 |
| 장애 복구 및 이중화 구현 | 시스템 중단 없이 복구 및 유지 | 금융 거래 내역 보존, DB 복구 |
| 모니터링 및 로깅 설정 | 실시간 상태 추적 및 분석 | 거래 처리 시간 분석, 장애 탐지 |
| CI/CD 파이프라인 구성 | 자동화된 빌드, 테스트, 배포 | 금융 시스템 릴리즈 관리 |
| 보안 정책 적용 | 금융 정보의 안전한 전송 및 저장 | 암호화 처리, 인증 관리 |

## ****구성도****

|  |
| --- |
| +------------------------------------------------------+  | Application Architecture 구현 전략 구조 |  |------------------------------------------------------|  | 1. API Gateway 구현 |  | - 클라이언트 요청 처리 및 라우팅 |  | - 보안 정책 적용 및 인증 |  |------------------------------------------------------|  | 2. MSA (Microservice Architecture) 구현 |  | - 서비스 단위 독립 배포 |  | - 무중단 확장 가능 |  |------------------------------------------------------|  | 3. 비동기 메시지 처리 |  | - 이벤트 기반 메시지 큐 사용 |  | - 실시간 데이터 처리 최적화 |  |------------------------------------------------------|  | 4. 트랜잭션 관리 구현 |  | - 데이터 정합성 보장 |  | - ACID 원칙 준수 |  |------------------------------------------------------|  | 5. 서비스 디스커버리 구성 |  | - Eureka, Consul 기반 동적 탐색 |  | - 로드 밸런싱 연동 |  |------------------------------------------------------|  | 6. 로드 밸런서 설정 |  | - Nginx, HAProxy 사용 |  | - 트래픽 자동 분산 |  |------------------------------------------------------|  | 7. 장애 복구 및 이중화 구현 |  | - Active-Active, Active-Passive 이중화 |  | - 무중단 서비스 유지 |  |------------------------------------------------------|  | 8. 모니터링 및 로깅 설정 |  | - Spring Actuator, Prometheus, ELK 연동 |  | - 실시간 상태 체크 |  |------------------------------------------------------|  | 9. CI/CD 파이프라인 구성 |  | - Jenkins, GitLab CI/CD, ArgoCD |  | - 무중단 배포 및 테스트 자동화 |  |------------------------------------------------------|  | 10. 보안 정책 적용 |  | - OAuth2.0, JWT, TLS/SSL 적용 |  | - 데이터 암호화 및 인증 관리 |  +------------------------------------------------------+ |

## ****상세 주요 내용 설계****

### ****API Gateway 구현****

#### ****개요****

* API Gateway는 클라이언트 요청을 올바른 서비스로 라우팅하며, 인증(Authentication)과 권한 관리(Authorization), Rate Limiting을 수행합니다.
* 이를 통해 외부에서의 접근을 통제하고, 내부 서비스로의 안전한 통신을 보장합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 인증 및 권한 관리 | API 접근에 대한 인증과 권한 부여 | ① OAuth2.0 인증 ② JWT(Json Web Token) 사용 ③ API Key 관리 ④ 역할 기반 접근 제어(RBAC) ⑤ Spring Security 연동 ⑥ SSO(Single Sign-On) 설정 ⑦ 접근 토큰 발급 ⑧ 만료 시간 관리 ⑨ 토큰 리프레시 정책 ⑩ 다중 인증(MFA) 설정 |
| 라우팅 설정 | 클라이언트 요청을 올바른 서비스로 전달 | ① URL Path 기반 라우팅 ② HTTP Method 기반 라우팅 ③ 동적 라우팅 설정 ④ Eureka 서비스 탐색 연동 ⑤ Request Path Rewrite ⑥ Failover 라우팅 ⑦ 로드 밸런싱 설정 ⑧ Service Discovery 연동 ⑨ 우선순위 기반 라우팅 ⑩ 상태 모니터링 |
| Rate Limiting | 과도한 요청으로부터 보호 | ① IP 기반 Rate Limiting ② User ID 기반 Throttling ③ Burst Limit 설정 ④ Time Window 관리 ⑤ 동적 요청 제한 설정 ⑥ 지역별 Rate Limit 설정 ⑦ API Key 별 Rate 설정 ⑧ QoS(Quality of Service) 적용 ⑨ 오버로드 방지 로깅 ⑩ Circuit Breaker 연동 |
| 로깅 및 모니터링 | API 요청과 응답을 추적하고 모니터링 | ① 요청 및 응답 로깅 ② 에러 발생 시 로그 기록 ③ 요청 시간 추적 ④ HTTP Status Code 분석 ⑤ API 사용량 통계 ⑥ Spring Actuator 연동 ⑦ 트랜잭션 추적(Distributed Tracing) ⑧ 중앙 집중식 로그 관리 ⑨ 대시보드 시각화 ⑩ 로그 분석 및 경고 발송 |
| 캐싱 최적화 | 반복되는 요청에 대한 응답 시간 단축 | ① In-Memory Cache 설정 ② Redis, Memcached 활용 ③ GET 요청 캐싱 ④ Expiration Time 설정 ⑤ 지역별 캐싱 정책 ⑥ 멀티 리전 캐싱 ⑦ 캐시 무효화 전략 ⑧ Lazy Loading 적용 ⑨ 캐시 미스(Cache Miss) 처리 ⑩ 캐시 적중률 최적화 |

### ****MSA (Microservice Architecture) 구현****

#### ****개요****

* MSA(Microservice Architecture) 구현은 금융 시스템의 확장성과 독립성을 극대화하기 위한 핵심 전략입니다.
* 각 도메인별로 서비스가 분리되며, 독립적으로 배포 및 확장이 가능합니다.
* 서비스 간 결합도를 최소화하고, 메시지 큐와 이벤트 기반 처리로 **비동기 통신**을 지원합니다.
* 주요 기술로는 **Spring Cloud**, **Docker**, **Kubernetes**, **Spring Boot**가 활용됩니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 서비스 분리 | 각 도메인을 개별 서비스로 분리하여 독립적 배포 | ① 도메인 기반 설계 (DDD: Domain-Driven Design) ② 독립적 서비스 배포 ③ 서비스 간 결합도 최소화 ④ 데이터베이스 분리 (DaaS: Database as a Service) ⑤ RESTful API 제공 ⑥ 비동기 메시지 처리 ⑦ 서비스 간 인증 및 보안 관리 ⑧ 독립 테스트 및 배포 ⑨ 클라우드 네이티브 설계 ⑩ 무중단 배포 전략 (Canary Release, Blue-Green Deployment) |
| API Gateway 통합 | 외부 요청을 단일 진입점으로 관리 | ① 인증 및 권한 관리 ② 라우팅 설정 ③ Circuit Breaker 패턴 적용 ④ 서비스 디스커버리 연동 ⑤ API 버전 관리 ⑥ Rate Limiting 설정 ⑦ SSL/TLS 암호화 통신 ⑧ 로깅 및 모니터링 ⑨ 캐싱 설정 ⑩ CORS 정책 관리 |
| 메시지 큐 통신 | 비동기 데이터 처리 및 이벤트 전달 | ① Kafka, RabbitMQ 사용 ② Topic & Queue 설정 ③ Publisher/Subscriber 모델 ④ 이벤트 재처리 로직 ⑤ 메시지 중복 방지 ⑥ 메시지 순서 보장 ⑦ DLQ(Dead Letter Queue) 구성 ⑧ 메시지 TTL(Time to Live) 설정 ⑨ 장애 시 재시도 정책 ⑩ 소비자(Consumer) 동적 확장 |
| 서비스 디스커버리 | 동적 라우팅 및 로드 밸런싱 | ① Eureka, Consul 사용 ② Service Registry 등록 ③ Heartbeat 체크 ④ 동적 IP 업데이트 ⑤ DNS 기반 탐색 ⑥ 로드 밸런싱과 연동 ⑦ 장애 시 Failover 처리 ⑧ 멀티 리전 탐색 ⑨ Health Check 설정 ⑩ 서비스 상태 모니터링 |
| 무중단 배포 | 서비스 중단 없이 업데이트 및 확장 | ① Blue-Green Deployment ② Canary Release 관리 ③ Zero Downtime 배포 ④ Rolling Update 설정 ⑤ Health Check 통과 시 배포 ⑥ 배포 실패 시 자동 롤백 ⑦ 배포 속도 최적화 ⑧ 무중단 스케일링 ⑨ 트래픽 라우팅 최적화 ⑩ A/B Testing 지원 |
| 데이터베이스 분리 | 서비스별 독립적인 DB 사용 | ① 각 서비스 별 DB Schema 관리 ② Cross-Service Join 방지 ③ Polyglot Persistence 적용 ④ DB Sharding 설정 ⑤ Multi-Tenant 구조 지원 ⑥ 데이터 정합성 관리 ⑦ 분산 트랜잭션 관리 ⑧ DB Migration 자동화 ⑨ DR(Disaster Recovery) 계획 수립 ⑩ Backup & Restore 최적화 |
| 컨테이너화 | 독립된 실행 환경 제공 | ① Docker 이미지 생성 ② Kubernetes 클러스터 배포 ③ Pod 관리 최적화 ④ Helm Chart 사용 ⑤ 환경 변수 기반 설정 ⑥ Multi-Stage Dockerfile 적용 ⑦ 네트워크 격리 보장 ⑧ Resource Limit 설정 ⑨ Logging 및 Monitoring 설정 ⑩ Canary 및 Blue-Green 배포 지원 |
| 분산 트랜잭션 관리 | 서비스 간 데이터 정합성 유지 | ① 2PC (Two-Phase Commit) 적용 ② Saga Pattern 사용 ③ 보상 트랜잭션 처리 ④ 데이터 일관성 유지 ⑤ 중간 실패 시 Rollback 처리 ⑥ 이벤트 기반 트랜잭션 관리 ⑦ 트랜잭션 모니터링 ⑧ 중복 처리 방지 ⑨ 분산 환경의 ACID 보장 ⑩ 이벤트 소싱 연계 |
| 장애 격리 및 복구 | 개별 서비스 장애가 전체에 영향 없도록 처리 | ① Circuit Breaker 설정 ② Bulkhead Pattern 적용 ③ 장애 시 빠른 Failover ④ 장애 탐지 시 알림 발송 ⑤ 장애 영역 격리 ⑥ 장애 복구 후 자동 재시작 ⑦ 상태 체크 및 재시도 로직 ⑧ 장애 내역 로깅 ⑨ 모니터링 및 트래킹 ⑩ 셀프 힐링(Self-Healing) 적용 |

### ****비동기 메시지 처리****

#### ****개요****

* 비동기 메시지 처리 구조는 대량 트랜잭션을 빠르게 처리하기 위한 최적화된 설계입니다.
* 요청에 대한 응답을 기다리지 않고 비동기로 처리하여 **성능 향상**과 **응답 시간 단축**을 목표로 합니다.
* 주요 기술로는 **Kafka**, **RabbitMQ**, **AWS SQS** 등이 활용되며, 장애 발생 시 메시지 손실을 최소화하기 위해 **Dead Letter Queue(DLQ)** 구조를 사용합니다.
* 또한 **Event-Driven Architecture**를 통해 MSA 간 비동기 이벤트 전달을 최적화합니다.

#### ****4.4.3.2 주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 메시지 큐 설계 | 대량 데이터 비동기 처리 | ① Kafka, RabbitMQ 사용 ② Topic & Queue 설정 ③ Partition 관리 ④ Consumer Group 설정 ⑤ Producer와 Consumer의 비동기 처리 ⑥ 메시지 중복 방지 ⑦ 메시지 순서 보장 ⑧ 메시지 TTL(Time to Live) 설정 ⑨ 메시지 전달 보장(At least once, Exactly once) ⑩ Dead Letter Queue 설정 |
| 이벤트 드리븐 아키텍처 | 이벤트 기반의 비동기 통신 | ① Publish/Subscribe 모델 ② 이벤트 버스(Event Bus) 사용 ③ 이벤트 핸들러 구성 ④ Event Router 설정 ⑤ 장애 발생 시 재처리 ⑥ 멀티 캐스팅 처리 ⑦ 메시지 브로커와 통합 ⑧ Event Store 구성 ⑨ 비동기 로깅 ⑩ 실시간 알림 처리 |
| 지연 큐(Delay Queue) 설계 | 특정 시간 이후에 처리되는 메시지 | ① 메시지 전송 지연 설정 ② 처리 시간 스케줄링 ③ 우선순위 메시지 처리 ④ Redis Delay Queue 사용 ⑤ AWS SQS Delay 적용 ⑥ 타임아웃 설정 ⑦ 큐 상태 모니터링 ⑧ 실패 시 재처리 ⑨ 메시지 만료 관리 ⑩ DLQ에 이관 |
| QoS (Quality of Service) 설정 | 메시지 전송 품질 보장 | ① 메시지 전달 보장(At most once, At least once) ② 중복 수신 방지 ③ 메시지 순서 유지 ④ 네트워크 손실 복구 ⑤ 재전송 정책 설정 ⑥ 메시지 처리 속도 제어 ⑦ 서비스 품질 측정 ⑧ 트래픽 스로틀링 ⑨ Load Balancing 설정 ⑩ SLA(Service Level Agreement) 관리 |
| 멱등성(Idempotence) 처리 | 중복 메시지에 대한 안전한 처리 | ① 메시지 ID 추적 ② 트랜잭션 처리 시 중복 방지 ③ 이벤트 발생 순서 보장 ④ 멱등 처리 설정 ⑤ API Gateway와 연동 ⑥ Kafka Idempotence 설정 ⑦ Redis Lock 활용 ⑧ 상태 변경 시 중복 방지 ⑨ Consistency 보장 ⑩ 메시지 일관성 유지 |
| 메시지 보안 | 전송 중 데이터 보호 | ① TLS/SSL 암호화 설정 ② JWT 토큰 검증 ③ 메시지 서명 검증 ④ RabbitMQ Access Control 설정 ⑤ Kafka ACL(Access Control List) 적용 ⑥ 암호화 전송(HTTPS, AMQP Secure) ⑦ 송신자 인증 처리 ⑧ 전송 시 무결성 검증 ⑨ 메시지 무단 접근 방지 ⑩ 데이터 유출 방지 |
| 메시지 복구 및 재처리 | 장애 발생 시 손실 메시지 복구 | ① Dead Letter Queue(DLQ) 구성 ② 실패 시 재처리 정책 설정 ③ 복구 시 트랜잭션 재처리 ④ 메시지 오프셋 추적 ⑤ Kafka Rewind 설정 ⑥ RabbitMQ Requeue 처리 ⑦ 메시지 재처리 횟수 관리 ⑧ 모니터링 및 로그 기록 ⑨ 비정상 메시지 격리 ⑩ 복구 후 상태 점검 |
| 메시지 압축 및 최적화 | 전송 효율성 향상 | ① Gzip, Snappy 압축 사용 ② 메시지 Payload 최적화 ③ 중복 데이터 제거 ④ 프로토콜 버퍼 사용 ⑤ 메시지 크기 제한 설정 ⑥ 메시지 패킷 최적화 ⑦ 압축 시 CPU 사용량 최적화 ⑧ 전송 속도 모니터링 ⑨ 메시지 압축 후 CRC 체크 ⑩ 압축률 분석 및 조정 |
| 트래픽 모니터링 | 메시지 전송 상태 추적 | ① 메시지 송수신 모니터링 ② 실패율 분석 ③ Latency 측정 ④ Throughput 분석 ⑤ Consumer Lag 분석 ⑥ 메시지 지연 시간 추적 ⑦ 메시지 전달 상태 대시보드 ⑧ 알림 설정 (Slack, Email) ⑨ 로그 분석 및 시각화 ⑩ 장애 발생 시 알림 발송 |

### ****트랜잭션 관리 구현****

#### ****개요****

* 트랜잭션 관리 구현은 금융 시스템의 **데이터 정합성**과 **일관성**을 보장하기 위한 핵심 설계입니다.
* ACID(Atomicity, Consistency, Isolation, Durability) 원칙을 준수하여 **모든 거래가 정확하게 처리**되며,  
  장애 시 **Rollback**이 가능하도록 설정됩니다.
* 특히, **분산 트랜잭션** 환경에서는 \*\*2PC (Two-Phase Commit)\*\*와 **Saga Pattern**을 활용하여 서비스 간 일관성을 유지합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| ACID 원칙 준수 | 트랜잭션의 원자성, 일관성, 격리성, 지속성을 보장 | ① 원자성(Atomicity): All or Nothing ② 일관성(Consistency): 정합성 유지 ③ 격리성(Isolation): 트랜잭션 간 영향 방지 ④ 지속성(Durability): 커밋된 데이터 영구 보장 ⑤ Isolation Level 설정 (Read Committed, Serializable 등) ⑥ 커밋 롤백 처리 ⑦ 동시성 처리 보장 ⑧ 트랜잭션 Lock 관리 ⑨ Snapshot Isolation 적용 ⑩ 트랜잭션 로그 추적 |
| 트랜잭션 전파 설정 | 트랜잭션이 다른 트랜잭션에 미치는 영향을 제어 | ① Propagation.REQUIRED 설정 ② Propagation.REQUIRES\_NEW 설정 ③ Propagation.NESTED 설정 ④ Propagation.MANDATORY 설정 ⑤ Nested Transaction 관리 ⑥ 트랜잭션 Isolation Level 설정 ⑦ 다중 DB 트랜잭션 전파 ⑧ 중첩 트랜잭션 처리 ⑨ 부모-자식 트랜잭션 동기화 ⑩ 트랜잭션 에러 핸들링 |
| 분산 트랜잭션 관리 | 여러 서비스 간 트랜잭션 일관성 유지 | ① 2PC (Two-Phase Commit) 적용 ② XA 트랜잭션 지원 ③ JTA(Java Transaction API) 사용 ④ 글로벌 트랜잭션 관리 ⑤ 상태 동기화 처리 ⑥ 중간 실패 시 보상 처리 ⑦ 롤백 일관성 유지 ⑧ 트랜잭션 격리 수준 설정 ⑨ 다중 DB 트랜잭션 처리 ⑩ 장애 발생 시 자동 복구 |
| Saga Pattern 적용 | 비동기 트랜잭션 복구 전략 | ① 보상 트랜잭션 설정 ② 이벤트 기반 트랜잭션 처리 ③ 장애 시 Rollback 흐름 구성 ④ 서비스 간 트랜잭션 상태 동기화 ⑤ Event Choreography 패턴 ⑥ Event Orchestration 패턴 ⑦ 중단 시 재처리 로직 ⑧ 이벤트 로그 보관 ⑨ 트랜잭션 상태 관리 ⑩ 비동기 보상 처리 |
| 동시성 제어 | 동시에 실행되는 트랜잭션 간 충돌 방지 | ① Optimistic Lock 설정 ② Pessimistic Lock 설정 ③ Read Lock과 Write Lock 구분 ④ Row-level Lock 설정 ⑤ MVCC (Multi-Version Concurrency Control) ⑥ Lock Timeout 설정 ⑦ Deadlock 방지 ⑧ Lock Escalation 관리 ⑨ 분산 환경에서 Lock 동기화 ⑩ 트랜잭션 Conflict 해결 |
| 트랜잭션 복구 | 장애 발생 시 트랜잭션 복구 처리 | ① 장애 발생 시 Rollback ② DB 로그 복구 ③ WAL (Write-Ahead Logging) 사용 ④ 트랜잭션 로그 추적 ⑤ 복구 시점 관리 (Point-in-Time Recovery) ⑥ Failover 시 일관성 유지 ⑦ 로그 분석 및 재처리 ⑧ 자동화된 복구 시나리오 ⑨ 트랜잭션 ID 추적 ⑩ 복구 후 상태 검증 |
| 트랜잭션 모니터링 | 실시간 트랜잭션 상태 모니터링 | ① 트랜잭션 실행 시간 추적 ② 커밋/롤백 비율 분석 ③ 실패율 측정 ④ Latency 모니터링 ⑤ Throughput 분석 ⑥ 장애 탐지 및 알림 ⑦ 중단된 트랜잭션 추적 ⑧ 트랜잭션 누락 탐지 ⑨ 이벤트 흐름 추적 ⑩ 실시간 대시보드 시각화 |
| 트랜잭션 최적화 | 대량 트랜잭션 처리 최적화 | ① Batch 처리 설정 ② Connection Pool 관리 ③ Lazy Loading 적용 ④ SQL 쿼리 최적화 ⑤ Fetch Size 조정 ⑥ 네트워크 최적화 ⑦ Prepared Statement 사용 ⑧ Connection Timeout 설정 ⑨ 트랜잭션 크기 제한 ⑩ 커밋 빈도 최적화 |
| 트랜잭션 격리 수준 설정 | 데이터의 일관성과 동시성 처리 최적화 | ① Read Uncommitted ② Read Committed ③ Repeatable Read ④ Serializable ⑤ Phantom Read 방지 ⑥ Dirty Read 방지 ⑦ Non-Repeatable Read 방지 ⑧ Isolation Level 튜닝 ⑨ DBMS별 Isolation 지원 ⑩ 트랜잭션 Conflict 방지 |

### ****서비스 디스커버리 구성****

#### ****개요****

* 서비스 디스커버리(Service Discovery)는 MSA(Microservice Architecture) 환경에서 각 서비스의 위치를 자동으로 탐색하고 연결을 유지하는 구조입니다.
* 서비스가 동적으로 생성되거나 삭제되더라도 클라이언트가 변경 없이 접근할 수 있도록 합니다.
* 주요 기술로는 **Spring Cloud Eureka**, **Consul**, **Zookeeper** 등이 있으며, **로드 밸런싱**과 함께 안정적인 서비스 탐색을 지원합니다.
* 또한, 장애 발생 시 **Failover**를 자동으로 처리하여 안정성을 높입니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 서비스 레지스트리 | 서비스의 위치 정보를 등록하고 탐색 | ① Spring Cloud Eureka 설정 ② Consul 연동 ③ Zookeeper 클러스터 구성 ④ Service Instance 등록 ⑤ Heartbeat 체크 ⑥ 동적 IP 업데이트 ⑦ 서비스 상태 모니터링 ⑧ Health Check 주기 관리 ⑨ 서비스 종료 시 자동 해제 ⑩ 멀티 클러스터 연동 |
| 서비스 탐색 | 클라이언트가 동적으로 서비스 위치를 탐색 | ① Eureka Client 설정 ② Consul Client 설정 ③ DNS 기반 탐색 ④ API Gateway와 연계 ⑤ Circuit Breaker 적용 ⑥ 장애 발생 시 대체 서비스 탐색 ⑦ 글로벌 서비스 탐색 ⑧ 지역별 로드 밸런싱 ⑨ 라우팅 정책 설정 ⑩ Latency 기반 탐색 최적화 |
| 로드 밸런싱 | 트래픽을 여러 서비스 인스턴스로 분산 | ① Round Robin 방식 ② Least Connection 방식 ③ IP Hash 방식 ④ Weighted 방식 ⑤ Service Discovery와 연동 ⑥ Spring Cloud LoadBalancer 설정 ⑦ Ribbon, Netflix Zuul 연동 ⑧ Session Sticky 설정 ⑨ 장애 탐지 시 Failover 처리 ⑩ Auto Scaling 연동 |
| Health Check | 서비스의 상태를 주기적으로 체크 | ① Health Endpoint 설정 ② Spring Actuator 연동 ③ Custom Health Indicator 설정 ④ 서비스 생존 여부 확인 ⑤ Eureka Heartbeat 모니터링 ⑥ Consul Health Check 통합 ⑦ Circuit Breaker와 연동 ⑧ 서비스 복구 시 알림 발송 ⑨ 장애 탐지 시 라우팅 변경 ⑩ 상태 모니터링 대시보드 |
| 서비스 동적 확장 | 서비스 상태에 따른 동적 확장 | ① Auto Scaling 설정 ② Spring Cloud Gateway와 연동 ③ 서비스 증가 시 동적 등록 ④ 서비스 축소 시 자동 해제 ⑤ 트래픽 부하 분산 최적화 ⑥ Kubernetes Horizontal Pod Autoscaler 사용 ⑦ 무중단 스케일링 ⑧ Multi-AZ (Availability Zone) 지원 ⑨ Scaling Event 로그 관리 ⑩ Capacity Planning 최적화 |
| 서비스 자동 등록 및 해제 | 서비스 인스턴스의 자동화 관리 | ① Spring Boot Actuator 사용 ② Eureka Self-Preservation 설정 ③ Consul TTL(Time To Live) 관리 ④ Zookeeper Ephemeral Node 사용 ⑤ Heartbeat 미응답 시 자동 해제 ⑥ 상태 변화 시 즉시 반영 ⑦ 서비스 재시작 시 자동 등록 ⑧ 장애 시 자동 해제 ⑨ Cluster 간 서비스 등록 ⑩ Cloud 환경에서 자동 동기화 |
| 장애 감지 및 복구 | 장애 발생 시 자동 탐지 및 복구 | ① Circuit Breaker 설정 ② Health Check Failure 시 알림 ③ Auto Healing 설정 ④ 서비스 비정상 종료 감지 ⑤ Heartbeat Miss 시 복구 시도 ⑥ Load Balancer와 연계한 Failover ⑦ API Gateway 차단 설정 ⑧ 문제 발생 시 Route 변경 ⑨ Recovery Time Objective (RTO) 관리 ⑩ Recovery Point Objective (RPO) 관리 |
| 서비스 라우팅 최적화 | 최적의 서비스 탐색 경로 설정 | ① Spring Cloud Gateway와 연동 ② Dynamic Routing 설정 ③ Eureka와 Zuul 연계 ④ DNS 기반 로드 밸런싱 ⑤ 지역별 최적 경로 탐색 ⑥ Multi-Region Routing 설정 ⑦ Circuit Breaker로 장애 시 자동 변경 ⑧ Request Path Mapping ⑨ Weight 기반 라우팅 ⑩ 실시간 Latency 최적화 |
| 서비스 상태 모니터링 | 서비스 상태의 실시간 모니터링 | ① Prometheus와 Grafana 연동 ② 서비스 지표 수집 (CPU, Memory, Latency) ③ 트래픽 분석 및 리포팅 ④ 로그 수집 및 시각화 ⑤ Health Check 상태 확인 ⑥ 장애 발생 시 알림 발송 ⑦ 메트릭 기반 스케일링 ⑧ 트랜잭션 상태 추적 ⑨ 글로벌 서비스 모니터링 ⑩ 장애 이력 분석 |
| 서비스 정책 관리 | 서비스 배포 및 운영 정책 설정 | ① Canary Deployment 정책 설정 ② Blue-Green Deployment 전략 ③ Circuit Breaker Threshold 설정 ④ API Rate Limiting 정책 ⑤ Region Locking 설정 ⑥ Disaster Recovery 정책 수립 ⑦ 서비스 복구 시나리오 작성 ⑧ 요청 우선순위 설정 ⑨ 트래픽 관리 정책 ⑩ Multi-Region Failover 전략 |

### ****로드 밸런서 설정****

#### ****개요****

* 로드 밸런서 설정은 다수의 서버에 트래픽을 분산하여 **시스템 성능과 가용성**을 최적화하는 구조입니다.
* 요청이 특정 서버에 집중되는 것을 방지하고, 장애 발생 시 자동으로 다른 서버로 트래픽을 전환(Failover)하여 **서비스의 연속성**을 보장합니다.
* 주요 기술로는 **Nginx**, **HAProxy**, **AWS ELB**, **Kubernetes Ingress** 등이 있으며, **Round Robin, Least Connection, IP Hash**와 같은 알고리즘을 활용합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 로드 밸런싱 알고리즘 | 트래픽 분산을 위한 최적화된 알고리즘 사용 | ① Round Robin 방식 ② Least Connection 방식 ③ IP Hash 기반 라우팅 ④ Weighted Round Robin 설정 ⑤ Least Response Time 적용 ⑥ Random Load Balancing ⑦ Sticky Session 처리 ⑧ Geo-based Load Balancing ⑨ Health Check 기반 라우팅 ⑩ Fault Tolerant 분산 처리 |
| Health Check | 서비스 인스턴스의 상태를 주기적으로 검사 | ① HTTP Health Check 설정 ② TCP Health Check 설정 ③ HTTPS Health Check 설정 ④ Heartbeat Monitor 구성 ⑤ Graceful Shutdown 관리 ⑥ 장애 발생 시 자동 Failover ⑦ 정상 복구 시 자동 등록 ⑧ Health 상태 대시보드 연동 ⑨ 복구 시 트래픽 재분배 ⑩ 모니터링 알림 설정 |
| 글로벌 로드 밸런싱 | 글로벌 서비스 배포 시 트래픽 분산 | ① Multi-Region 로드 밸런싱 ② Latency 기반 라우팅 ③ Geo-DNS 라우팅 설정 ④ AWS Global Accelerator 연동 ⑤ GCP Cloud Load Balancing 설정 ⑥ Azure Traffic Manager 사용 ⑦ 지역 간 Failover 처리 ⑧ Disaster Recovery 연동 ⑨ Multi-Region Failback 설정 ⑩ Cross-Region Replication 최적화 |
| 세션 스티키(Session Sticky) | 사용자 세션이 동일 서버에 유지되도록 설정 | ① Session Affinity 설정 ② Cookie 기반 세션 유지 ③ IP Hash 기반 스티키 세션 ④ HAProxy Stick Table 사용 ⑤ Spring Session 클러스터링 ⑥ Redis 세션 저장소 설정 ⑦ 세션 만료 시 Graceful Handling ⑧ Cluster 간 세션 동기화 ⑨ 사용자 세션 복구 ⑩ 무중단 세션 유지 |
| Auto Scaling 연동 | 트래픽 증가에 따른 동적 확장 | ① Kubernetes Horizontal Pod Autoscaler 설정 ② AWS Auto Scaling 설정 ③ GCP Autoscaler 설정 ④ Threshold 기반 Scale Out ⑤ Load Balancing 연동 ⑥ 스케일링 시 무중단 처리 ⑦ Scale-In 정책 설정 ⑧ 리소스 최적화 ⑨ 비용 최적화 관리 ⑩ 장애 시 자동 복구 |
| SSL/TLS 설정 | 보안 통신을 위한 암호화 설정 | ① HTTPS 통신 설정 ② SSL/TLS 인증서 적용 ③ SSL Offloading 처리 ④ Self-Signed Certificate 생성 ⑤ Let's Encrypt 연동 ⑥ TLS 1.2, 1.3 지원 ⑦ Forward Secrecy 설정 ⑧ HSTS(HTTP Strict Transport Security) 설정 ⑨ 인증서 갱신 자동화 ⑩ 중간자 공격(MITM) 방지 |
| DDoS 방어 설정 | 대량의 비정상 트래픽에 대한 방어 | ① AWS Shield 설정 ② Cloudflare DDoS Protection ③ IP Blacklist 관리 ④ Rate Limiting 설정 ⑤ UDP Flooding 방지 ⑥ SYN Flooding 차단 ⑦ Application Layer 방어 ⑧ WAF(Web Application Firewall) 적용 ⑨ Slowloris Attack 방어 ⑩ Traffic Scrubbing |
| 캐싱 최적화 | 로드 밸런서 레벨에서 캐싱 처리 | ① In-Memory Cache 설정 ② Redis, Memcached 연동 ③ GET 요청 캐싱 ④ Expiration Time 설정 ⑤ 지역별 캐싱 정책 ⑥ 멀티 리전 캐싱 ⑦ 캐시 무효화 전략 ⑧ Lazy Loading 적용 ⑨ 캐시 미스(Cache Miss) 처리 ⑩ 캐시 적중률 최적화 |
| 모니터링 및 로그 관리 | 로드 밸런서 상태 및 트래픽 분석 | ① Prometheus와 Grafana 연동 ② ELK Stack (Elasticsearch, Logstash, Kibana) 설정 ③ 실시간 트래픽 분석 ④ 로그 수집 및 보관 ⑤ 응답 시간 분석 ⑥ 에러 발생 시 알림 전송 ⑦ Latency 측정 ⑧ 분산 로그 관리 ⑨ Health 상태 시각화 ⑩ 로그 롤링 및 백업 |
| 무중단 배포 지원 | 서비스 중단 없이 확장 및 배포 | ① Blue-Green Deployment ② Canary Release 설정 ③ Zero Downtime 배포 ④ Rolling Update 구성 ⑤ Health Check 후 트래픽 전환 ⑥ 배포 실패 시 롤백 ⑦ 무중단 배포 최적화 ⑧ 트래픽 라우팅 변경 ⑨ 실시간 상태 감시 ⑩ 멀티 리전 배포 최적화 |

### ****장애 복구 및 이중화 구현****

#### ****개요****

* 장애 복구 및 이중화 설계는 **시스템 중단 없이 연속적인 서비스 제공**을 목표로 합니다.
* 장애 발생 시 신속하게 감지하고, 자동으로 다른 노드로 전환(Failover)하여 서비스를 지속합니다.
* 이중화 구조로는 **Active-Active**, **Active-Passive** 방식이 있으며, 주요 기술로는 **Zookeeper**, **Consul**, **Kubernetes HA**가 활용됩니다.
* 데이터의 무결성을 보장하면서 빠르게 복구되도록 **DR(Disaster Recovery)** 및 **Replication**을 구성합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 이중화 구조 설계 | Active-Active, Active-Passive 구조로 고가용성 보장 | ① Active-Active 클러스터 구성 ② Active-Passive 설정 ③ Failover 시 자동 전환 ④ 데이터 동기화 관리 ⑤ 장애 탐지 시 Traffic Switch ⑥ 데이터 Replication 설정 ⑦ Heartbeat Monitor 구성 ⑧ Disaster Recovery 전략 ⑨ 멀티 리전 이중화 ⑩ Split-Brain 방지 |
| 장애 탐지 및 대응 | 장애 발생 시 빠른 감지와 대응 | ① 장애 탐지 모니터링 설정 ② 실시간 알림 발송 ③ 장애 유형 분류 ④ 자동 롤백 처리 ⑤ 로그 분석 및 원인 추적 ⑥ Self-Healing 설정 ⑦ 장애 발생 시 Auto Scaling ⑧ 자동화된 서비스 복구 ⑨ Recovery Point Objective(RPO) 설정 ⑩ Recovery Time Objective(RTO) 설정 |
| 데이터 백업 및 복구 | 데이터 손실 방지와 빠른 복구 | ① Full Backup 설정 ② Incremental Backup 구성 ③ Differential Backup 관리 ④ Disaster Recovery Site 설정 ⑤ Offsite Backup 저장 ⑥ 암호화된 백업 처리 ⑦ 백업 데이터 무결성 검증 ⑧ 복구 테스트 자동화 ⑨ 백업 주기 설정 ⑩ 클라우드 백업 최적화 |
| 스냅샷 및 복제 | 장애 시 빠른 상태 복원 | ① 데이터베이스 스냅샷 생성 ② Volume Snapshot 설정 ③ Kubernetes PV Snapshot 활용 ④ 시점 복구 지원 ⑤ Multi-Region Snapshot 연동 ⑥ Snapshot Consistency 보장 ⑦ 주기적 Snapshot 백업 ⑧ 장애 시 데이터 복구 ⑨ 암호화된 스냅샷 생성 ⑩ Auto-Snapshot 설정 |
| DR(Disaster Recovery) 설계 | 재해 발생 시 서비스 연속성 보장 | ① DR Site 구축 ② Multi-AZ 배포 ③ RPO/RTO 최적화 ④ 데이터 동기화 보장 ⑤ Cloud Region 간 Active-Passive 설정 ⑥ Region Failover 처리 ⑦ DR 연습 테스트 시나리오 ⑧ 데이터 복구 검증 ⑨ DNS Failover 설정 ⑩ 전환 시 Latency 최적화 |
| 클러스터링 및 페일오버 | 클러스터 내 장애 시 자동 전환 | ① Zookeeper, Consul 설정 ② Master-Slave 전환 ③ Cluster State Sync 처리 ④ Heartbeat 감지 ⑤ Split Brain 방지 ⑥ Leader Election 설정 ⑦ Kubernetes HA 설정 ⑧ 장애 시 자동 롤백 ⑨ Failback 정책 설정 ⑩ Service Mesh 연동 |
| Self-Healing 설정 | 장애 발생 시 자동 복구 처리 | ① Kubernetes Auto Healing ② Instance Recovery 설정 ③ AWS Auto Healing 설정 ④ 프로세스 재시작 정책 ⑤ Health Check 실패 시 재시작 ⑥ 로그 분석을 통한 원인 추적 ⑦ 장애 복구 후 상태 검증 ⑧ 자동화된 복구 시나리오 실행 ⑨ 상태 복구 시 알림 발송 ⑩ 트랜잭션 보전 처리 |
| 네트워크 이중화 | 네트워크 장애 시 대응 및 복구 | ① Multi-AZ 설정 ② Dual Path 구성 ③ BGP Failover 설정 ④ VRRP (Virtual Router Redundancy Protocol) 사용 ⑤ VPC Peering 설정 ⑥ Gateway 이중화 ⑦ Load Balancer 다중화 ⑧ CloudFront Region Failover ⑨ DNS Failover 설정 ⑩ 네트워크 경로 최적화 |
| 로그 분석 및 장애 리포팅 | 장애 발생 시 원인 분석 및 추적 | ① ELK Stack 설정 ② Fluentd를 통한 로그 수집 ③ 장애 탐지 시 알림 전송 ④ 로그 시각화 대시보드 ⑤ 장애 유형 분석 리포트 ⑥ 트랜잭션 흐름 추적 ⑦ 서비스 복구 시점 기록 ⑧ 원인 추적 이력 보관 ⑨ 장애 탐지 자동화 ⑩ AI 기반 Root Cause Analysis |
| 멀티 리전 이중화 | 지역 간 재해 발생 시 자동 전환 | ① Multi-Region 배포 설정 ② AWS Global Accelerator 사용 ③ GCP Cloud Load Balancer 연동 ④ Azure Traffic Manager 설정 ⑤ Cross-Region Replication ⑥ Latency 기반 Failover ⑦ Disaster Recovery Region 지정 ⑧ Multi-Region DNS Failover ⑨ 멀티 리전 간 동기화 ⑩ 복구 시 Latency 최적화 |

### ****모니터링 및 로깅 설정****

#### ****개요****

* 모니터링 및 로깅 설정은 시스템의 상태를 실시간으로 감지하고 문제를 추적할 수 있도록 하는 중요한 관리 요소입니다.
* 주요 컴포넌트로는 **Prometheus**, **Grafana**, **ELK Stack(Elasticsearch, Logstash, Kibana)**, **Fluentd**, **Zipkin** 등이 활용됩니다.
* **분산 트랜잭션 추적, 실시간 알림, 대시보드 시각화**를 통해 빠른 장애 감지와 복구를 지원합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 시스템 모니터링 | 애플리케이션 상태 및 성능 추적 | ① Prometheus로 메트릭 수집 ② Grafana 대시보드 시각화 ③ CPU, Memory 사용량 모니터링 ④ 네트워크 트래픽 추적 ⑤ Disk I/O 분석 ⑥ Latency 측정 ⑦ Throughput 분석 ⑧ Auto Scaling 트리거 설정 ⑨ 장애 발생 시 알림 발송 ⑩ 장애 이력 보관 |
| 애플리케이션 로그 수집 | 서비스 실행 중 발생하는 로그 수집 및 분석 | ① Fluentd를 통한 로그 수집 ② ELK Stack 설정 ③ JSON 형식으로 표준화 ④ 로그의 중앙 집중 관리 ⑤ 로그 레벨 설정 (INFO, ERROR, WARN) ⑥ 로그 회전 설정 (Log Rotation) ⑦ 실시간 로그 분석 ⑧ 서비스 간 로그 연동 ⑨ 로깅 정책 설정 ⑩ 장애 시점 추적 |
| 분산 트랜잭션 추적 | 서비스 간 트랜잭션 흐름 분석 | ① Zipkin, Jaeger 설정 ② Trace ID 생성 및 전파 ③ Distributed Tracing 설정 ④ 서비스 간 통신 시 Latency 분석 ⑤ 트랜잭션 흐름 시각화 ⑥ 장애 구간 탐지 ⑦ 트랜잭션 순서 보장 ⑧ 이벤트 간 상관 관계 분석 ⑨ 경로 최적화 추적 ⑩ 실시간 경고 설정 |
| 대시보드 시각화 | 실시간 데이터 상태 시각화 | ① Grafana 대시보드 구성 ② Kibana를 통한 로그 시각화 ③ 실시간 트래픽 분석 ④ 장애 발생 시 그래프 표현 ⑤ CPU, Memory 사용량 시각화 ⑥ 트랜잭션 처리량 표시 ⑦ 응답 시간 분석 ⑧ 사용자 세션 모니터링 ⑨ 이벤트 히스토리 추적 ⑩ SLA(Single Level Agreement) 모니터링 |
| 알림 및 경고 설정 | 장애 발생 시 실시간 알림 전송 | ① Slack, Email, SMS 연동 ② Prometheus AlertManager 설정 ③ 서비스 중단 시 경고 발송 ④ 응답 시간 지연 시 알림 ⑤ 트랜잭션 오류 탐지 시 알림 ⑥ 로드 밸런서 장애 시 경고 ⑦ 로그 에러 발생 시 통지 ⑧ Disk 용량 임계점 초과 시 알림 ⑨ DB Connection Fail 시 경고 ⑩ 모니터링 지표 초과 시 자동 통지 |
| 이벤트 추적 | 서비스 실행 중 발생한 이벤트 기록 | ① 이벤트 타임라인 추적 ② 이벤트 발생 시간 기록 ③ 이벤트 상태 모니터링 ④ 이벤트 리플레이 기능 ⑤ 이벤트 순서 보장 ⑥ 이벤트 손실 감지 ⑦ 이벤트 버스(Event Bus) 연동 ⑧ 이벤트 간 상관 관계 분석 ⑨ 비정상 이벤트 탐지 ⑩ 이벤트 전파 시 성능 최적화 |
| 성능 최적화 모니터링 | 시스템 성능을 극대화하기 위한 최적화 | ① Garbage Collection(GC) 모니터링 ② CPU 스레드 사용량 분석 ③ JVM Heap Memory 추적 ④ 네트워크 Latency 최적화 ⑤ DB Connection Pool 모니터링 ⑥ 캐싱 처리 분석 ⑦ I/O 성능 측정 ⑧ TPS(Transactions Per Second) 분석 ⑨ 쿼리 최적화 모니터링 ⑩ API 응답 시간 분석 |
| 보안 로그 모니터링 | 보안 이벤트 추적 및 위협 탐지 | ① Spring Security Audit 설정 ② 로그인 실패 추적 ③ 접근 권한 변경 감지 ④ IP 블랙리스트 탐지 ⑤ 비정상 API 호출 모니터링 ⑥ XSS, SQL Injection 탐지 ⑦ 사용자 권한 변경 추적 ⑧ VPN Access Log 분석 ⑨ JWT 토큰 만료 관리 ⑩ 클라이언트 IP 변경 추적 |
| SLA 및 SLO 모니터링 | 서비스 수준 협약 준수 여부 모니터링 | ① SLA(Single Level Agreement) 추적 ② SLO(Service Level Objective) 모니터링 ③ 가용성 분석 ④ 응답 시간 목표 설정 ⑤ 장애 대응 시간 추적 ⑥ 성능 목표 달성률 분석 ⑦ API 응답 시간 모니터링 ⑧ 트랜잭션 성공률 분석 ⑨ 데이터 처리율 확인 ⑩ 목표 대비 지표 분석 |
| 로그 보관 및 관리 | 로그의 장기 보관 및 관리 최적화 | ① Elasticsearch를 통한 장기 보관 ② S3 연동 백업 ③ 주기적인 로그 압축 및 보관 ④ 이력 데이터 분석 ⑤ 장애 발생 시점의 로그 보존 ⑥ 접근 로그 별도 관리 ⑦ GDPR 및 개인정보 보호 준수 ⑧ Multi-Region 로그 복제 ⑨ Log Retention Policy 설정 ⑩ 접근 권한 관리 |

### ****CI/CD 파이프라인 구성****

#### ****개요****

* CI/CD(Continuous Integration / Continuous Deployment) 파이프라인은 **지속적인 통합과 배포 자동화**를 통해 개발된 애플리케이션을 신속하고 안정적으로 배포할 수 있도록 하는 전략입니다.
* 주요 기술로는 **Jenkins**, **GitLab CI/CD**, **ArgoCD**, **Tekton**, **Spinnaker** 등이 있으며, **자동화된 테스트, 빌드, 배포**가 가능합니다.
* CI/CD를 통해 **무중단 배포, 롤백, 멀티 리전 배포, Canary Release, Blue-Green Deployment** 등이 가능하며, 배포 과정에서 발생할 수 있는 오류를 최소화합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| Continuous Integration (CI) | 코드 변경 사항을 지속적으로 통합하여 신속한 빌드 및 테스트 수행 | ① Jenkins Pipeline 설정 ② GitLab CI/CD 구성 ③ 소스 코드 자동 빌드 ④ 코드 정적 분석(SonarQube) ⑤ Unit Test 자동화 ⑥ Integration Test 자동화 ⑦ Code Coverage 측정 ⑧ Pull Request 자동 테스트 ⑨ Dependency Scan ⑩ 멀티 모듈 빌드 지원 |
| Continuous Delivery (CD) | 통합된 코드를 자동으로 배포할 수 있는 상태로 유지 | ① ArgoCD를 통한 자동 배포 ② Tekton Pipeline 설정 ③ Canary Release 설정 ④ Blue-Green Deployment 구성 ⑤ Zero Downtime 배포 ⑥ Docker Container Registry 연동 ⑦ Helm Chart 기반 배포 ⑧ Kubernetes Rollout 설정 ⑨ ConfigMap & Secret 동기화 ⑩ 실패 시 자동 Rollback |
| 배포 자동화 | 배포 프로세스를 코드화하여 반복적인 배포 자동화 | ① Infrastructure as Code (IaC) 설정 ② Terraform을 통한 환경 구성 ③ Ansible, Chef, Puppet 사용 ④ 배포 스크립트 관리 ⑤ 스테이징 환경과 프로덕션 환경 분리 ⑥ Jenkinsfile 구성 ⑦ Argo Rollouts 사용 ⑧ 무중단 배포 최적화 ⑨ 다중 클러스터 배포 ⑩ 클라우드 네이티브 배포 |
| 테스트 자동화 | 빌드된 코드를 자동으로 테스트하여 품질 보장 | ① Unit Test 자동화 ② Integration Test 자동화 ③ End-to-End(E2E) Test 설정 ④ Regression Test 관리 ⑤ Performance Test 구성 ⑥ API Gateway Test 연동 ⑦ Selenium을 활용한 UI Test ⑧ Test Coverage 분석 ⑨ Mock 서버 연동 ⑩ 장애 시나리오 테스트 |
| 모니터링 및 피드백 | 배포 이후 시스템 상태를 실시간 모니터링하고 피드백 수집 | ① Prometheus & Grafana 대시보드 구성 ② ELK Stack 연동 ③ 실시간 배포 로그 수집 ④ 배포 실패 시 알림 발송 ⑤ Latency 측정 ⑥ 트랜잭션 분석 ⑦ 배포 성능 모니터링 ⑧ Metrics 분석 및 리포트 생성 ⑨ 슬랙(Slack) 연동 알림 ⑩ API 응답 시간 추적 |
| 롤백 전략 | 배포 실패 시 신속한 복구를 위한 자동 롤백 처리 | ① ArgoCD 롤백 설정 ② Spinnaker Pipeline Rollback ③ Helm Rollback 처리 ④ Jenkins Pipeline에서 자동 Rollback ⑤ Kubernetes Rollout Undo ⑥ Canary Release 중단 및 복구 ⑦ GitOps 방식으로 버전 관리 ⑧ 배포 실패 시 이전 버전 복원 ⑨ 장애 탐지 시 즉시 Rollback ⑩ 롤백 후 테스트 자동 실행 |
| 멀티 리전 배포 | 전 세계 여러 리전에 동시에 배포 | ① AWS Global Accelerator 설정 ② GCP Multi-Region 설정 ③ Azure Traffic Manager 사용 ④ Multi-Region Kubernetes 클러스터 구성 ⑤ Cross-Region Load Balancing ⑥ Geo-Distributed Deployment ⑦ CDN(Content Delivery Network) 연동 ⑧ 복제본(Replica) 배포 ⑨ Region Failover 구성 ⑩ Latency 최적화 배포 |
| 보안 및 접근 제어 | CI/CD 파이프라인 내 보안 강화 | ① Git Secret 관리 ② Vault 연동으로 민감 정보 보호 ③ Role-Based Access Control (RBAC) 설정 ④ OAuth2.0 인증 연동 ⑤ API Key 관리 ⑥ Artifact Signing 설정 ⑦ IP Whitelisting 설정 ⑧ Docker Image Scan ⑨ Container Vulnerability Scan ⑩ AWS IAM Role 기반 인증 |
| 아티팩트 관리 | 빌드된 바이너리 파일 및 Docker Image 관리 | ① Nexus, JFrog Artifactory 사용 ② Docker Registry 설정 ③ Maven Repository 연동 ④ Gradle Artifact 배포 ⑤ 이미지 버전 관리 ⑥ Rollback 시 Artifact 재활용 ⑦ 다중 버전 호환 관리 ⑧ 사이닝(Signing) 및 검증 ⑨ 오픈소스 종속성 관리 ⑩ 빌드 속도 최적화 |
| GitOps 구현 | Git을 Single Source of Truth로 관리 | ① ArgoCD GitOps 설정 ② FluxCD 연동 ③ Pull Request 기반 배포 ④ GitHub Actions 통합 ⑤ GitLab CI와 연동 ⑥ Git Branch 관리 최적화 ⑦ 상태 불일치 시 자동 조정 ⑧ 배포 환경 상태 동기화 ⑨ GitOps 이벤트 트리거 설정 ⑩ 멀티 클러스터 관리 |

### ****보안 정책 적용****

#### ****개요****

* 금융 시스템의 보안 정책은 민감한 정보의 유출을 방지하고, **데이터 무결성**과 **사용자 인증**을 보장하기 위한 필수적인 설계 요소입니다.
* 주요 기술로는 **OAuth2.0**, **JWT(Json Web Token)**, **TLS/SSL 암호화**, **WAF(Web Application Firewall)** 등이 있으며, **전송 구간 암호화, 접근 제어, 로깅 및 감사**를 포함합니다.
* 특히, **OWASP Top 10**을 기준으로 XSS, SQL Injection, CSRF 등 보안 취약점을 차단하고, 실시간 보안 모니터링을 통해 위협을 빠르게 탐지합니다.

#### ****주요 설계 내용****

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 세부 기능 |
| 인증 및 권한 관리 | 사용자 인증과 권한 부여를 통해 접근 통제 | ① OAuth2.0 인증 설정 ② JWT(Json Web Token) 발급 ③ Multi-Factor Authentication(MFA) 지원 ④ RBAC(Role-Based Access Control) 구성 ⑤ Spring Security 통합 ⑥ SSO(Single Sign-On) 설정 ⑦ Token Expiration 설정 ⑧ Refresh Token 관리 ⑨ IP Whitelisting 설정 ⑩ 사용자 세션 추적 |
| 전송 구간 암호화 | 네트워크 통신 중 데이터 보호 | ① TLS/SSL 암호화 설정 ② HTTPS 통신 강제화 ③ Forward Secrecy 적용 ④ HSTS(HTTP Strict Transport Security) 설정 ⑤ Self-Signed Certificate 발급 ⑥ Mutual TLS 인증 ⑦ SSL Offloading 처리 ⑧ 중간자 공격 방지(MITM) ⑨ Perfect Forward Secrecy(PFS) 설정 ⑩ 인증서 자동 갱신 |
| 데이터 보안 및 암호화 | 민감한 데이터를 안전하게 보호 | ① AES-256, RSA 암호화 적용 ② 필드 레벨 암호화(Field-Level Encryption) ③ 데이터베이스 암호화(TDE, Transparent Data Encryption) ④ 개인정보 마스킹 설정 ⑤ 암호화 키 관리(Key Management Service, KMS) ⑥ Rest API 데이터 암호화 ⑦ 암호화된 백업 처리 ⑧ 데이터 무결성 검증 ⑨ 해시 알고리즘(SHA-256) 사용 ⑩ 접근 제어 정책 설정 |
| 방화벽 및 네트워크 보안 | 외부 공격으로부터 시스템 보호 | ① WAF(Web Application Firewall) 설정 ② IP Blacklist 및 Whitelist 관리 ③ DDoS Protection 설정 ④ VPN Gateway 구성 ⑤ Bastion Host 관리 ⑥ IDS/IPS 설정 ⑦ VPC 네트워크 격리 ⑧ Port Scanning 탐지 ⑨ 네트워크 트래픽 모니터링 ⑩ Zero Trust Architecture 구현 |
| API 보안 | 외부 API 호출 시 보안 정책 적용 | ① OAuth2.0 인증 토큰 설정 ② API Rate Limiting 적용 ③ CORS(Cross-Origin Resource Sharing) 설정 ④ API Gateway 보안 정책 구성 ⑤ JWT Signature 검증 ⑥ XSS 및 CSRF 방어 ⑦ API Key 발급 및 관리 ⑧ OpenAPI Spec 기반 보안 검증 ⑨ 요청 IP 추적 ⑩ API 사용 이력 로깅 |
| 감사 및 로깅 | 보안 이벤트의 추적과 감사 로그 관리 | ① Spring Security Audit 설정 ② 로그인 및 로그아웃 이력 추적 ③ DB 변경 사항 로깅 ④ 권한 변경 이력 보관 ⑤ 접근 시도 실패 기록 ⑥ IP 변경 탐지 및 알림 ⑦ 데이터 조회 이력 저장 ⑧ Audit Trail 관리 ⑨ GDPR 및 정보 보호 규정 준수 ⑩ 실시간 로그 분석 |
| 취약점 관리 | 보안 취약점을 탐지하고 대응 | ① OWASP Top 10 대응 정책 ② SQL Injection 탐지 및 차단 ③ XSS(Cross-Site Scripting) 방지 ④ CSRF(Cross-Site Request Forgery) 방어 ⑤ Insecure Deserialization 탐지 ⑥ Directory Traversal 방어 ⑦ Security Misconfiguration 탐지 ⑧ 불필요한 포트 차단 ⑨ Outdated Component 탐지 ⑩ 보안 패치 자동화 |
| 접근 통제 정책 | 시스템 자원에 대한 접근 제한 | ① RBAC(Role-Based Access Control) 설정 ② ABAC(Attribute-Based Access Control) 적용 ③ IP Filtering 설정 ④ Geo-Fencing 설정 ⑤ Time-based Access Control ⑥ VPN Access Control 설정 ⑦ Kubernetes Network Policy 설정 ⑧ AWS Security Group 관리 ⑨ Private Subnet 격리 ⑩ 다중 인증 요구 |
| 보안 모니터링 및 대응 | 실시간 보안 위협 탐지 및 대응 | ① SIEM(Security Information and Event Management) 연동 ② Prometheus Alert 설정 ③ Grafana 보안 대시보드 구성 ④ ELK Stack으로 실시간 로그 분석 ⑤ 비정상 트래픽 탐지 ⑥ 악성 로그인 탐지 ⑦ SSH Brute Force 방지 ⑧ AWS GuardDuty 연동 ⑨ 실시간 알림(Slack, Email) ⑩ 자동화된 보안 대응 |
| 데이터 유출 방지(DLP) | 외부로의 데이터 유출 방지 | ① DLP 정책 설정 ② 데이터 전송 시 암호화 ③ 이메일 첨부파일 암호화 ④ USB 및 외장 장치 차단 ⑤ 클라우드 스토리지 접근 통제 ⑥ 화면 캡처 방지 설정 ⑦ 데이터 사용 이력 추적 ⑧ 민감 정보 접근 모니터링 ⑨ 파일 공유 정책 관리 ⑩ 네트워크 필터링 설정 |

## ****실사례 (설계 항목 예시 30개)****

요구사항 분석 시 구체적으로 설계해야 하는 항목들을 **30개 이상** 정리하였으며,  
각 항목에는 **적용 사례**와 **필수 세부 기능 10개 이상**을 함께 표기하였습니다.  
**4.4 상세 주요 내용 설계**의 항목 순서에 맞추어 설계 항목을 작성합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 설계 항목 | 설명 | 적용 사례 | 세부 기능 |
| **API Gateway 구현** | 클라이언트 요청을 단일 진입점에서 관리하며, 서비스로 라우팅 | 금융권 Open API 플랫폼, 모바일 뱅킹 | ① OAuth2.0 인증 ② JWT 토큰 검증 ③ 라우팅 설정 ④ CORS 정책 관리 ⑤ Rate Limiting ⑥ 로깅 및 모니터링 ⑦ Circuit Breaker 설정 ⑧ API 버전 관리 ⑨ SSL/TLS 암호화 ⑩ IP Filtering |
| **MSA (Microservice Architecture) 구현** | 독립된 서비스 단위로 개발 및 배포 | 대규모 결제 시스템, 보험 청구 시스템 | ① 서비스 간 비동기 메시지 처리 ② 서비스 디스커버리 연동 ③ 독립적 배포와 확장 ④ 데이터베이스 분리 (DaaS) ⑤ Spring Boot 기반 Microservice ⑥ Kubernetes 클러스터링 ⑦ 무중단 배포 설정 ⑧ API Gateway 통합 ⑨ Eureka & Consul 연동 ⑩ Circuit Breaker 적용 |
| **비동기 메시지 처리** | 트랜잭션 처리 속도 최적화 | 실시간 결제 승인, 예약 처리 | ① 메시지 큐 (Kafka, RabbitMQ) 구성 ② Async Task 처리 ③ Event-Driven 처리 ④ 트랜잭션 분리 ⑤ 대기 시간 최소화 ⑥ 순서 보장 처리 ⑦ 메시지 재처리 ⑧ Latency 최적화 ⑨ 비동기 로깅 ⑩ 비동기 에러 핸들링 |
| **트랜잭션 관리 구현** | 금융 거래의 무결성과 일관성 보장 | 실시간 송금, 대량 이체, 금융 결제 시스템 | ① ACID 속성 보장 ② Two-Phase Commit(2PC) 적용 ③ Saga Pattern 관리 ④ Isolation Level 설정 ⑤ XA 트랜잭션 처리 ⑥ 트랜잭션 격리성 보장 ⑦ 동시성 제어 (Pessimistic/Optimistic Locking) ⑧ 트랜잭션 모니터링 ⑨ 장애 시 Rollback ⑩ 복구 시점 관리 |
| **서비스 디스커버리 구성** | 마이크로서비스 간 동적 라우팅 및 탐색 | 금융 API Gateway, 외부 연동 | ① Eureka, Consul 사용 ② Service Registry 등록 ③ Heartbeat 체크 ④ 동적 IP 업데이트 ⑤ DNS 기반 탐색 ⑥ 로드 밸런싱과 연동 ⑦ 장애 시 Failover 처리 ⑧ 멀티 리전 탐색 ⑨ Health Check 설정 ⑩ 서비스 상태 모니터링 |
| **로드 밸런서 설정** | 트래픽을 여러 서버로 분산하여 처리 성능 향상 | 인터넷 뱅킹, 결제 시스템 | ① Round Robin 방식 ② Least Connection 방식 ③ IP Hash 기반 라우팅 ④ Weighted Round Robin 설정 ⑤ Geo-based Load Balancing ⑥ Health Check 설정 ⑦ Session Sticky 처리 ⑧ Auto Scaling 연동 ⑨ 글로벌 로드 밸런싱 ⑩ 장애 시 Failover 처리 |
| **장애 복구 및 이중화 구현** | 시스템 장애 시 빠른 복구를 보장 | 금융 트랜잭션 이중화, 데이터 복구 | ① Active-Active, Active-Passive 구성 ② Heartbeat 모니터링 ③ 장애 발생 시 자동 전환 ④ 데이터 복제 및 동기화 ⑤ DR(Disaster Recovery) 구축 ⑥ Multi-AZ 이중화 ⑦ 자동 롤백 처리 ⑧ 장애 탐지 및 알림 ⑨ 스냅샷 복구 ⑩ 멀티 리전 장애 대응 |
| **모니터링 및 로깅 설정** | 실시간 상태 추적 및 이력 관리 | 결제 이력 모니터링, 대시보드 시각화 | ① Prometheus와 Grafana 설정 ② ELK Stack 연동 ③ Fluentd 로그 수집 ④ Trace ID 기반 분산 추적 ⑤ 실시간 로그 분석 ⑥ Latency 모니터링 ⑦ 장애 발생 시 알림 ⑧ CPU, Memory 사용량 추적 ⑨ 트랜잭션 처리 시간 분석 ⑩ 사용자 요청 분석 |
| **CI/CD 파이프라인 구성** | 지속적인 통합 및 배포 자동화 | 글로벌 결제 시스템, 보험 관리 시스템 | ① Jenkins Pipeline 설정 ② GitLab CI/CD 연동 ③ ArgoCD 자동 배포 ④ Canary Release 설정 ⑤ Blue-Green Deployment 구성 ⑥ Zero Downtime 배포 ⑦ Helm Chart 배포 관리 ⑧ Kubernetes Rollout 설정 ⑨ Multi-Cluster 배포 ⑩ GitOps 관리 |
| **보안 정책 적용** | 금융 정보 보호 및 전송 구간 암호화 | 온라인 뱅킹, 금융 API 보안 | ① OAuth2.0 인증 설정 ② JWT 기반 인증 관리 ③ Spring Security 연동 ④ TLS/SSL 암호화 설정 ⑤ 데이터베이스 암호화(TDE) ⑥ WAF 설정 ⑦ XSS 및 SQL Injection 방어 ⑧ IP Whitelisting 설정 ⑨ Access Control Policy ⑩ Audit Logging |

## ****마무리****

애플리케이션 아키텍처 구현 전략은 금융 시스템의 **안정성, 확장성, 보안성**을 극대화하기 위한 핵심 설계입니다.  
특히, 금융 시스템은 대규모 트랜잭션 처리와 높은 수준의 보안이 요구되기 때문에 다음과 같은 요소들이 철저하게 구현되어야 합니다:

* + **안정성 확보**
* MSA(Microservice Architecture) 구조와 API Gateway를 활용하여 서비스 간 결합도를 최소화하고,  
  독립적인 배포와 장애 시 격리된 복구가 가능하도록 설계되었습니다.
* 트랜잭션 관리와 장애 복구 이중화 구조를 통해 서비스 중단 없이 무결성을 보장합니다.
  + **확장성과 유연성 극대화**
* 서비스 간 비동기 메시지 처리와 서비스 디스커버리로 \*\*자동 확장(Auto Scaling)\*\*이 가능하며,  
  글로벌 서비스 확장을 위한 **멀티 리전(Multi-Region) 배포**도 지원합니다.
* 로드 밸런서를 활용한 트래픽 최적화는 서비스 성능을 극대화합니다.
  + **보안성 강화**
* OAuth2.0, JWT 인증, RBAC(Role-Based Access Control)을 통해 금융 데이터를 보호하며,  
  데이터 전송 구간은 TLS/SSL로 암호화되었습니다.
* 방화벽 및 API 보안 정책을 통해 악의적인 공격으로부터 보호하며, 실시간 보안 모니터링이 연동되어 빠르게 탐지하고 대응합니다.
  + **무중단 배포 및 자동화**
* CI/CD 파이프라인을 통해 개발부터 배포까지 자동화하여 빠른 릴리즈와 무중단 업데이트가 가능합니다.
* Canary Release, Blue-Green Deployment, Rolling Update를 통해 단계적 배포와 문제 발생 시 즉시 롤백이 가능합니다.
  + **실시간 모니터링 및 복구**
* Prometheus, Grafana, ELK Stack을 활용하여 시스템 상태를 실시간으로 추적하고,  
  장애 발생 시 빠르게 탐지하고 복구합니다.
* Self-Healing 설정으로 장애 탐지 시 자동으로 재시작하거나 복구하는 구조가 구축되었습니다.

## ****4.7 시사점****

**1. 금융 시스템의 최적화된 MSA 구조 필요성**

* 금융 시스템의 대규모 트랜잭션을 처리하기 위해서는 **MSA 구조**가 필수적입니다.
* MSA는 서비스 간 결합도를 최소화하여 개별 서비스의 배포와 확장을 빠르게 진행할 수 있습니다.
* 또한 장애가 발생하더라도 특정 서비스만 격리되기 때문에 전체 시스템의 안정성을 유지할 수 있습니다.

**2. 글로벌 확장성과 장애 대응력의 중요성**

* 글로벌 서비스 확장을 고려한다면, **Multi-Region** 배포 전략과 **로드 밸런싱 최적화**가 필수입니다.
* AWS Global Accelerator, Azure Traffic Manager와 같은 글로벌 라우팅 기술을 적용하면  
  지역 간 Latency를 최적화하고 장애 시 빠르게 Failover가 가능합니다.

**3. 실시간 모니터링 및 복구 체계 구축**

* 금융 트랜잭션의 경우 **실시간 추적**이 가능해야 하며, **Distributed Tracing**을 통해 장애 구간을 빠르게 탐색할 수 있어야 합니다.
* Prometheus와 Grafana 대시보드를 통해 실시간 상태를 시각화하고, 장애 발생 시 빠른 복구를 지원해야 합니다.
* Self-Healing 구조를 적용하면 장애 발생 시 즉시 복구가 가능하여 고객에게 끊김 없는 서비스 경험을 제공합니다.

**4. 데이터 보안과 접근 제어의 중요성**

* 금융 데이터를 다루기 때문에 데이터 전송 시 **TLS/SSL** 암호화가 필수적이며, **RBAC**를 통한 접근 제어가 필요합니다.
* 또한, API 보안을 강화하고, WAF(Web Application Firewall)를 설정하여 **OWASP Top 10** 공격을 방어해야 합니다.
* 실시간 Audit Trail을 통해 접근 기록을 관리하고, 보안 이벤트를 추적해야만 신뢰성을 보장할 수 있습니다.

**5. CI/CD 파이프라인을 통한 무중단 배포**

* 금융 서비스는 중단 없이 배포할 수 있는 무중단 배포 전략이 필수입니다.
* CI/CD 파이프라인을 통해 **자동화된 테스트, 빌드, 배포**가 가능하며, 오류 발생 시 즉시 Rollback이 가능합니다.
* GitOps 방식을 적용하면 Git Repository가 단일 진실 원천(Single Source of Truth) 역할을 하며, 일관된 배포가 보장됩니다.

# ****5. 솔루션 조합 및 기술 선정****

## ****5.1 기술 영역별 후보 솔루션****

### ****5.1.1 개요****

* 기술 영역별로 최적의 후보 솔루션을 선정하기 위해 시장에서 검증된 기술과 최신 트렌드를 반영하였습니다.
* 클라우드 플랫폼, 애플리케이션 프레임워크, API 게이트웨이, 메시지 브로커, 데이터베이스, 보안 솔루션, 모니터링 툴 등으로 구분하였으며,  
  각 기술에 대해 **상용 솔루션**과 **오픈 소스 솔루션**을 구분하여 정리하였습니다.

### ****5.1.2 주요 기술 영역 및 후보 솔루션 목록****

|  |  |  |  |
| --- | --- | --- | --- |
| 기술 영역 | 주요 상용 솔루션 | 주요 오픈 소스 솔루션 | 설명 |
| **클라우드 플랫폼** | AWS, Microsoft Azure, Google Cloud Platform | OpenStack, Cloud Foundry | 클라우드 네이티브 환경을 지원하며 글로벌 리전 배포 가능 |
| **애플리케이션 프레임워크** | Spring Boot, .NET Core, IBM WebSphere | Spring, Quarkus, Micronaut | MSA 구조 및 RESTful API 개발에 최적화된 프레임워크 |
| **API 게이트웨이** | Apigee, AWS API Gateway, Kong Enterprise | Kong, Tyk, Spring Cloud Gateway | API 요청을 관리하고, 인증 및 라우팅을 처리 |
| **메시지 브로커** | IBM MQ, RabbitMQ Enterprise, Confluent Kafka | RabbitMQ, Apache Kafka, NATS | 비동기 메시지 처리 및 서비스 간 통신 최적화 |
| **데이터베이스** | Oracle DB, Microsoft SQL Server, IBM Db2 | MySQL, PostgreSQL, MongoDB, Cassandra | 트랜잭션 처리와 대량 데이터 저장 및 조회 |
| **검색 엔진** | Elastic Cloud, IBM Watson Discovery | Elasticsearch, Apache Solr | 대량의 데이터에 대한 빠른 검색과 분석 |
| **컨테이너 오케스트레이션** | AWS ECS, Google Kubernetes Engine, OpenShift | Kubernetes, Docker Swarm, Nomad | MSA 서비스의 배포, 확장, 복구를 자동화 |
| **서비스 디스커버리** | AWS App Mesh, Consul Enterprise | Consul, Eureka, Zookeeper | MSA 간 동적 서비스 탐색과 라우팅 처리 |
| **모니터링 툴** | Dynatrace, Datadog, New Relic | Prometheus, Grafana, ELK Stack | 실시간 상태 모니터링과 대시보드 시각화 |
| **보안 솔루션** | Palo Alto Prisma, AWS Shield, Azure Sentinel | Wazuh, Vault, Keycloak | API 보호, 트랜잭션 암호화, 접근 제어 |
| **CI/CD 도구** | Jenkins Enterprise, GitLab CI, Bamboo | Jenkins, GitLab CI, Tekton | 자동화된 빌드, 테스트, 배포 |
| **로깅 및 추적** | Splunk, Sumo Logic | ELK Stack (Elasticsearch, Logstash, Kibana), Fluentd | 트랜잭션 추적 및 실시간 로그 분석 |
| **캐싱 솔루션** | Redis Enterprise, Hazelcast | Redis, Memcached | 조회 속도 향상을 위한 인메모리 캐싱 |
| **로드 밸런서** | F5 Big-IP, Citrix NetScaler | Nginx, HAProxy, Traefik | 다수의 서비스 인스턴스에 트래픽 분산 |
| **파일 스토리지** | AWS S3, Google Cloud Storage | MinIO, Ceph | 파일 및 객체 저장소 |

### **5.1.3 분석 요약**

* 클라우드 플랫폼은 **AWS, Azure, GCP**가 글로벌 인프라 확장에 최적화되어 있으며, 대체 기술로 **OpenStack**이 활용 가능합니다.
* 애플리케이션 프레임워크는 금융권 트랜잭션에 강한 **Spring Boot**가 가장 적합하며, 경량화된 **Micronaut**도 대안이 될 수 있습니다.
* 메시지 브로커는 실시간 처리가 중요한 금융권 특성상 **Apache Kafka**가 최적이며, 고신뢰성을 요구할 경우 **IBM MQ**가 유효합니다.
* API 게이트웨이는 상용 솔루션으로 **Apigee**, 오픈 소스에서는 **Spring Cloud Gateway**가 MSA 연동에 적합합니다.
* 데이터베이스는 **Oracle DB**가 안정적이나, **PostgreSQL**도 강력한 트랜잭션 처리를 지원합니다.
* 컨테이너 오케스트레이션은 **Kubernetes**가 업계 표준으로 자리잡고 있으며, 클라우드 네이티브 구현 시 필수 요소입니다.

## ****솔루션 비교 및 평가 기준****

### ****5.2.1 개요****

* 금융권 시스템에서 요구하는 주요 특성인 **고가용성, 무중단 처리, 대량 트랜잭션 처리, 보안성**을 기반으로 각 기술 솔루션을 평가합니다.
* 기술별 **성능, 확장성, 안정성, 보안성, 운영 관리**를 항목으로 설정하였으며, 상용 솔루션과 오픈 소스 솔루션을 구분하여 비교합니다.

### ****5.2.2 주요 솔루션 평가표****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 기술 영역 | 솔루션 | 성능 | 확장성 | 안정성 | 보안성 | 운영 관리 | 종합 평가 |
| **클라우드 플랫폼** | AWS | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | 최상 |
|  | Microsoft Azure | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | Google Cloud Platform | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★★ | ★★★★☆ | 최상 |
|  | OpenStack | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **애플리케이션 프레임워크** | Spring Boot | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★★ | 최상 |
|  | .NET Core | ★★★★★ | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
|  | Micronaut | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **API 게이트웨이** | Apigee | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | AWS API Gateway | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | Kong | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **메시지 브로커** | Apache Kafka | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | 최상 |
|  | RabbitMQ | ★★★★☆ | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
|  | NATS | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **데이터베이스** | Oracle DB | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | Microsoft SQL Server | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
|  | PostgreSQL | ★★★★☆ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
|  | MongoDB | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **컨테이너 오케스트레이션** | Kubernetes | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | OpenShift | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
| **모니터링 툴** | Prometheus & Grafana | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | ★★★★★ | 최상 |
|  | Dynatrace | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
| **보안 솔루션** | AWS Shield | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | Palo Alto Prisma | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | 최상 |
|  | Wazuh | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |
| **CI/CD 도구** | Jenkins | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★★ | 최상 |
|  | GitLab CI | ★★★★★ | ★★★★★ | ★★★★★ | ★★★★☆ | ★★★★☆ | 상 |
|  | Tekton | ★★★★☆ | ★★★★★ | ★★★★☆ | ★★★★☆ | ★★★★☆ | 상 |

### ****5.2.3 평가 기준****

|  |  |  |
| --- | --- | --- |
| 평가 기준 | 설명 | 비율(%) |
| **성능** | 대량의 트랜잭션 처리 속도와 응답 시간 최적화 | 30% |
| **확장성** | 트래픽 증가 시 확장 가능한 구조 | 20% |
| **안정성** | 장애 발생 시 자동 복구 및 데이터 일관성 보장 | 20% |
| **보안성** | 민감 데이터 보호 및 외부 공격 방지 | 20% |
| **운영 관리** | 관리 용이성 및 배포 자동화 | 10% |

### ****5.2.4 종합 평가 요약****

* 금융권 특성상 **성능**과 **안정성**이 가장 중요하며, 그다음으로 **확장성**이 중요하게 평가됩니다.
* 클라우드 플랫폼은 **AWS**, **Azure**, **GCP**가 가장 우수하며, 오픈 소스 중에서는 **OpenStack**이 유력합니다.
* 애플리케이션 프레임워크는 **Spring Boot**가 독보적이며, **MSA 구조와 REST API**에 최적화되었습니다.
* 메시지 브로커는 **Apache Kafka**가 대량 메시지 처리에 탁월하고, **데이터베이스**는 **Oracle DB**가 신뢰성이 높습니다.
* 모니터링 툴은 **Prometheus & Grafana**가 실시간 상태 추적에 유리하며, 보안 솔루션은 **AWS Shield**가 글로벌 수준의 보안을 보장합니다.

## ****5.3 상용/오픈 소스 조합 전략****

### ****5.3.1 개요****

* 금융권 시스템에서 상용 솔루션과 오픈 소스 솔루션의 조합은 비용 최적화와 성능 최적화를 동시에 달성할 수 있는 전략입니다.
* 핵심 기술 영역에서 **핵심 처리 및 보안은 상용 솔루션**, **확장성 및 개발 속도는 오픈 소스 솔루션**을 활용하여 밸런스를 맞춥니다.
* 예를 들어, \*\*클라우드 인프라(AWS, Azure)\*\*는 상용 솔루션을 선택하고, \*\*API Gateway, Microservice Framework(Spring Boot)\*\*는 오픈 소스를 활용합니다.

### ****5.3.2 주요 기술 영역 조합 전략****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 기술 영역 | 상용 솔루션 | 오픈 소스 솔루션 | 조합 전략 | 주요 장점 | 고려 사항 |
| **클라우드 플랫폼** | AWS, Microsoft Azure | OpenStack, Cloud Foundry | 상용: 클라우드 인프라 & 네트워크 오픈 소스: 프라이빗 클라우드 확장 | 글로벌 리전 배포, 고가용성 | 상용 클라우드 종속성 |
| **애플리케이션 프레임워크** | IBM WebSphere, Oracle WebLogic | Spring Boot, Micronaut | 상용: 레거시 시스템 통합 오픈 소스: MSA 및 REST API 개발 | 개발 속도 향상, 경량화 | 레거시 연동 시 복잡성 증가 |
| **API 게이트웨이** | Apigee, AWS API Gateway | Kong, Spring Cloud Gateway | 상용: API 보안 및 정책 관리 오픈 소스: 로컬 개발 및 경량 배포 | 정책 관리 최적화, 비용 절감 | 상용은 비용 발생, 오픈 소스는 보안 설정 요구 |
| **메시지 브로커** | IBM MQ, RabbitMQ Enterprise | Apache Kafka, RabbitMQ | 상용: 금융 거래 처리 신뢰성 보장 오픈 소스: 비동기 처리 최적화 | 장애 복구 용이, 트랜잭션 보장 | 상용은 높은 비용, 오픈 소스는 설정 복잡 |
| **데이터베이스** | Oracle DB, Microsoft SQL Server | PostgreSQL, MongoDB | 상용: 주요 금융 데이터 저장 오픈 소스: 비정형 데이터 관리 | 고신뢰성, 빠른 조회 | 상용 DB의 비용 이슈 |
| **검색 엔진** | Elastic Cloud | Elasticsearch | 상용: 보안 관리 및 데이터 암호화 오픈 소스: 로그 분석 및 실시간 검색 | 빠른 검색 속도, 확장성 | 클러스터 관리 복잡성 |
| **컨테이너 오케스트레이션** | Red Hat OpenShift | Kubernetes | 상용: 엔터프라이즈 통합 관리 오픈 소스: 서비스 배포 최적화 | 클러스터 관리 최적화 | Multi-Region 구성 시 복잡성 |
| **서비스 디스커버리** | AWS App Mesh | Eureka, Consul | 상용: 글로벌 서비스 탐색 오픈 소스: 로컬 서비스 등록 | 자동 탐색, 로드 밸런싱 최적화 | 오픈 소스 장애 시 복구 전략 필요 |
| **모니터링 툴** | Dynatrace, New Relic | Prometheus, Grafana | 상용: SLA 모니터링 및 알림 오픈 소스: 실시간 대시보드 시각화 | 실시간 모니터링, 트랜잭션 추적 | 상용은 비용 발생, 오픈 소스는 설정 복잡 |
| **보안 솔루션** | Palo Alto Prisma, AWS Shield | Wazuh, Vault | 상용: DDoS 방어, 방화벽 설정 오픈 소스: 데이터 암호화, 접근 관리 | 강력한 보안 관리, 트래픽 보호 | 설정 복잡성 증가 |
| **CI/CD 도구** | GitLab CI Enterprise, Jenkins Enterprise | Jenkins, Tekton | 상용: 대규모 프로젝트 관리 오픈 소스: 파이프라인 자동화 | 배포 속도 최적화 | 파이프라인 복잡도 관리 필요 |
| **로깅 및 추적** | Splunk, Sumo Logic | ELK Stack, Fluentd | 상용: 대용량 로그 처리 최적화 오픈 소스: 실시간 로그 수집 및 분석 | 대용량 처리, 실시간 분석 | 오픈 소스의 클러스터 관리 복잡 |
| **캐싱 솔루션** | Redis Enterprise, Hazelcast | Redis, Memcached | 상용: 엔터프라이즈 캐시 관리 오픈 소스: 로컬 캐싱 최적화 | 빠른 데이터 접근 | Multi-Region 구성 시 복잡성 |
| **로드 밸런서** | F5 Big-IP, Citrix NetScaler | Nginx, HAProxy | 상용: 글로벌 트래픽 관리 오픈 소스: 로컬 트래픽 분산 | 고가용성, 성능 최적화 | 설정 복잡성 증가 |
| **파일 스토리지** | AWS S3, Google Cloud Storage | MinIO, Ceph | 상용: 글로벌 데이터 보존 오픈 소스: 로컬 데이터 분산 저장 | 확장성, 무중단 복구 | 데이터 이중화 관리 필요 |

### ****5.3.3 조합 전략 분석****

* **핵심 금융 트랜잭션**은 상용 솔루션(Oracle DB, IBM MQ)을 통해 안정성과 신뢰성을 확보합니다.
* **MSA 서비스 개발**과 **API Gateway**는 오픈 소스(Spring Boot, Kong)을 활용하여 확장성을 극대화합니다.
* **모니터링 및 로깅**은 Prometheus와 Grafana를 활용하여 실시간 상태를 시각화하며, ELK Stack을 통해 분산 로그를 수집합니다.
* **CI/CD 파이프라인**은 Jenkins와 GitLab CI를 조합하여 무중단 배포를 지원합니다.
* **보안 솔루션**은 클라우드 네이티브 보안(AWS Shield)과 Wazuh를 함께 사용하여 다단계 방어를 구축합니다.

## ****5.4 기술 스택 및 버전 정합성****

### ****개요****

* 금융권 시스템은 **안정성과 보안성**이 최우선시되기 때문에, 기술 스택의 버전 정합성은 매우 중요합니다.
* 주요 기술의 **LTS 버전**과 **업그레이드 주기**, **하위 호환성**을 분석하여 시스템 운영 시 발생할 수 있는 리스크를 최소화합니다.
* 또한, 최신 기술 트렌드를 반영하되, 충분히 검증된 버전으로 안정성을 유지합니다.

### ****주요 기술 스택 및 버전 호환성****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 기술 영역 | 솔루션 | 현재 권장 버전 | LTS 지원 | 하위 호환성 | 업그레이드 주기 | 주요 특징 |
| **클라우드 플랫폼** | AWS | N/A | 지속 지원 | 상위 API 호환 | 매월 업데이트 | 글로벌 리전 확장 및 보안 최적화 |
|  | Microsoft Azure | N/A | 지속 지원 | 상위 API 호환 | 매월 업데이트 | Multi-Region 구성 최적화 |
|  | Google Cloud Platform | N/A | 지속 지원 | 상위 API 호환 | 매월 업데이트 | Serverless, AI 최적화 |
| **애플리케이션 프레임워크** | Spring Boot | 3.0.x | 2027년 12월까지 | 상위 호환 보장 | 연 2회 | MSA 및 REST API 최적화 |
|  | .NET Core | 7.0 | 2025년 5월까지 | 상위 호환 보장 | 연 1회 | Windows/Linux 크로스 플랫폼 |
|  | Micronaut | 3.x | 2026년 6월까지 | 상위 호환 보장 | 연 1회 | Cloud-Native 최적화 |
| **API 게이트웨이** | Kong | 3.1.x | 2026년 8월까지 | 상위 호환 보장 | 연 1회 | OpenAPI 지원 및 플러그인 확장 |
|  | Spring Cloud Gateway | 2022.0.x | 2027년 12월까지 | Spring Boot 연동 최적화 | 연 2회 | 라우팅 및 인증 최적화 |
| **메시지 브로커** | Apache Kafka | 3.4.x | 2027년 5월까지 | 상위 호환 보장 | 연 2회 | 이벤트 스트림 처리 최적화 |
|  | RabbitMQ | 3.11.x | 2026년 12월까지 | 상위 호환 보장 | 연 1회 | 고성능 메시지 브로커 |
| **데이터베이스** | Oracle DB | 19c | 2027년 4월까지 | 상위 호환 보장 | 3년 주기 | 트랜잭션 강력 보장 |
|  | PostgreSQL | 15.x | 2027년 9월까지 | 상위 호환 보장 | 연 1회 | 고성능 트랜잭션 처리 |
|  | MongoDB | 6.x | 2028년 2월까지 | 상위 호환 보장 | 연 1회 | 비정형 데이터 처리 최적화 |
| **컨테이너 오케스트레이션** | Kubernetes | 1.26.x | 2027년 12월까지 | 상위 호환 보장 | 3~4개월 주기 | MSA 배포 최적화 |
|  | OpenShift | 4.12.x | 2028년 3월까지 | 상위 호환 보장 | 연 1회 | 멀티 클러스터 관리 |
| **모니터링 툴** | Prometheus | 2.40.x | 2026년 7월까지 | 상위 호환 보장 | 연 1회 | 실시간 모니터링 최적화 |
|  | Grafana | 9.x | 2025년 12월까지 | 상위 호환 보장 | 연 1회 | 대시보드 시각화 최적화 |
| **보안 솔루션** | Vault | 1.14.x | 2027년 6월까지 | 상위 호환 보장 | 연 1회 | API 키 및 암호화 키 관리 |
|  | Wazuh | 4.4.x | 2025년 9월까지 | 상위 호환 보장 | 연 1회 | 실시간 보안 모니터링 |
| **CI/CD 도구** | Jenkins | 2.375.x (LTS) | 2025년 12월까지 | 상위 호환 보장 | 연 1회 | 파이프라인 자동화 최적화 |
|  | GitLab CI | 15.x | 2026년 8월까지 | 상위 호환 보장 | 연 1회 | 코드 리뷰 및 파이프라인 관리 |
| **로깅 및 추적** | ELK Stack | 8.x | 2026년 5월까지 | 상위 호환 보장 | 연 1회 | 실시간 로그 수집 및 분석 |
| **캐싱 솔루션** | Redis | 7.x | 2027년 9월까지 | 상위 호환 보장 | 연 1회 | 분산 캐싱 최적화 |
| **로드 밸런서** | Nginx | 1.24.x | 2025년 7월까지 | 상위 호환 보장 | 연 1회 | 고성능 로드 밸런싱 |

### ****분석 및 시사점****

1. **LTS(Long Term Support) 유지가 중요한 영역**
   * 금융권 시스템은 안정성이 중요하므로, **LTS 버전**이 제공되는 기술 스택을 사용합니다.
   * 예를 들어, **Spring Boot 3.x**, **Kubernetes 1.26.x**, **Redis 7.x**는 2027년 이후까지 안정적으로 지원됩니다.
2. **클라우드 플랫폼의 지속 업데이트**
   * AWS, Azure, GCP는 매월 업데이트되며 상위 호환성이 유지되므로 지속적인 버전 관리가 필요합니다.
3. **MSA 최적화 기술의 최신화**
   * **Spring Boot 3.x**, **Kong 3.x**, **Apache Kafka 3.4.x**는 MSA 환경에서 최적화된 성능을 보장합니다.
4. **CI/CD 도구의 지속적인 업데이트**
   * **Jenkins LTS**, **GitLab CI**는 매년 최신 버전이 나오며, 최신 기능과 보안 패치를 포함합니다.
5. **컨테이너 오케스트레이션의 주기적 업데이트**
   * **Kubernetes 1.26.x**는 3~4개월 단위로 빠르게 업데이트되므로, 지속적인 관리가 필요합니다.

## ****주요 기술 스택 버전 및 검증****

### ****개요****

* 금융권 시스템의 안정적 운영을 위해 주요 기술 스택 간 **버전 호환성**을 검증합니다.
* 특히, MSA(Microservice Architecture), API Gateway, 메시지 브로커, 데이터베이스, 로드 밸런서 간의 **API 호환성**과 **데이터 일관성**이 중요합니다.
* 호환성 검증과 성능 최적화 결과를 기반으로 **최적의 조합**을 도출하고, 장애 시 복구 전략까지 포함합니다.

### ****주요 기술 스택 간 호환성 검증 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 기술 스택 | 권장 버전 | 상호 호환성 | 주요 연계 기술 | 문제 발생 시 대응 전략 |
| **Spring Boot** | 3.0.x | ✅ | Spring Cloud, Spring Data JPA, Spring Security | Spring Cloud 호환성 보장 (2022.0.x), 버전 충돌 시 Spring Dependency Management 사용 |
| **Spring Cloud Gateway** | 2022.0.x | ✅ | Eureka, Zuul, Consul | Spring Boot 3.x와 완벽 호환, Gateway 라우팅 실패 시 Circuit Breaker로 대체 |
| **Apache Kafka** | 3.4.x | ✅ | Spring Cloud Stream, Confluent Schema Registry | 메시지 중복 시 Idempotent 설정으로 처리, Broker 장애 시 Failover 적용 |
| **Kubernetes** | 1.26.x | ✅ | Docker, Helm, Prometheus | Pod Failure 시 Auto Healing, 노드 장애 시 Replication 설정 |
| **PostgreSQL** | 15.x | ✅ | Spring Data JPA, Hibernate | Master-Slave Replication, 장애 시 Read Replica로 전환 |
| **Oracle DB** | 19c | ✅ | Spring Data JDBC, MyBatis | RAC(Real Application Clusters) 설정으로 장애 복구 |
| **Redis** | 7.x | ✅ | Spring Cache, Hibernate Cache | Sentinel 설정으로 장애 시 자동 전환 |
| **Nginx** | 1.24.x | ✅ | API Gateway, Kubernetes Ingress | Health Check 실패 시 다른 노드로 라우팅 |
| **Prometheus & Grafana** | 2.40.x / 9.x | ✅ | Kubernetes, Docker, Kafka | 노드 모니터링 실패 시 AlertManager로 통보 |
| **Jenkins** | 2.375.x (LTS) | ✅ | GitLab, Kubernetes | Build 실패 시 이전 버전으로 Rollback |
| **GitLab CI** | 15.x | ✅ | Docker, Kubernetes | CI/CD Pipeline 실패 시 자동 재시도 |
| **ELK Stack** | 8.x | ✅ | Fluentd, Kubernetes | Logstash 전송 실패 시 Buffer 저장 후 재전송 |

### ****호환성 검증 시나리오****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 시나리오 | 설명 | 적용된 기술 스택 | 검증 결과 | 대응 전략 |
| **MSA 간 통신 안정성** | MSA 구조에서 트랜잭션 처리 시 발생할 수 있는 메시지 손실 및 중복 전송에 대한 테스트 | Spring Boot 3.x, Apache Kafka 3.4.x | ✅ 정상 처리 | Kafka의 Idempotent Producer 설정을 통해 중복 방지 |
| **API Gateway 라우팅 실패 대응** | API Gateway에서 특정 서비스 연결 실패 시 다른 인스턴스로의 전환 | Spring Cloud Gateway, Eureka | ✅ 정상 처리 | Circuit Breaker 설정으로 요청 분산 |
| **Kubernetes Pod 장애 발생 시 처리** | Kubernetes 노드 장애 발생 시 자원 재할당 및 서비스 지속성 확인 | Kubernetes 1.26.x, Prometheus | ✅ 정상 처리 | Pod Auto Healing 활성화 |
| **DB 장애 발생 시 무중단 처리** | PostgreSQL Master 장애 시 Read Replica로 전환하여 서비스 유지 | PostgreSQL 15.x, Spring Data JPA | ✅ 정상 처리 | Failover 설정 및 Spring Retry 적용 |
| **Redis 장애 시 캐시 복구** | Redis Primary 장애 시 Sentinel 설정으로 Replica 전환 | Redis 7.x, Spring Cache | ✅ 정상 처리 | Sentinel 설정을 통한 자동 전환 |
| **ELK 로그 수집 누락 시 복구** | Fluentd가 일시적으로 로그 전송에 실패할 경우 복구 여부 | ELK Stack 8.x, Fluentd | ✅ 정상 처리 | Buffer 재시도 설정으로 누락된 로그 복구 |
| **CI/CD 배포 중단 시 Rollback 처리** | Jenkins Pipeline 배포 실패 시 이전 버전으로 복원 | Jenkins 2.375.x, GitLab CI 15.x | ✅ 정상 처리 | GitOps 방식으로 이전 버전 Rollback |

### ****분석 및 시사점****

1. **MSA 구조의 안정성 확보**
   * Spring Boot 3.x와 Apache Kafka 3.4.x의 상호 호환성은 이벤트 처리의 안정성을 보장합니다.
   * Circuit Breaker 패턴과 Idempotent 설정으로 메시지 유실 없이 안정적인 트랜잭션 처리가 가능합니다.
2. **API Gateway의 고가용성 확보**
   * Spring Cloud Gateway는 Eureka와 연동되어 서비스 탐색이 자동화되며, 장애 시 Circuit Breaker를 통해 우회합니다.
3. **Kubernetes의 자동 복구 최적화**
   * Kubernetes 1.26.x는 Auto Healing 기능을 통해 노드 장애 시에도 서비스 중단 없이 복구됩니다.
   * Prometheus와 Grafana를 통해 실시간 상태를 추적할 수 있습니다.
4. **데이터베이스의 무중단 처리 전략**
   * PostgreSQL과 Oracle DB는 복제(Replication) 설정을 통해 장애 발생 시 무중단으로 전환이 가능합니다.
   * 금융권의 실시간 트랜잭션을 안정적으로 처리할 수 있는 구조입니다.
5. **Redis 캐시 복구 전략 최적화**
   * Redis 7.x는 Sentinel 설정을 통해 장애 시 자동으로 다른 노드로 전환됩니다.
   * 금융권에서 캐시 데이터의 무결성을 유지합니다.

## ****장기 유지보수 및 기술 생태계 평가****

### ****개요****

* 금융권 시스템은 일반적인 애플리케이션보다 **안정성**과 **무중단 서비스**가 중요합니다.
* 기술 스택의 **LTS(Long Term Support) 지원** 여부와 **기술 생태계의 활성화 정도**는 장기적인 시스템 운영의 핵심 요소입니다.
* 또한, **오픈 소스 커뮤니티의 활동성**과 **기술 업그레이드 주기**가 유지보수의 용이성을 결정합니다.

### ****주요 기술 스택 유지보수 평가 표****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 기술 스택 | 버전 | LTS 지원 | 업그레이드 주기 | 커뮤니티 활성화 | 기술 생태계 지원 | 주요 특성 |
| **Spring Boot** | 3.0.x | 2027년 12월까지 | 연 2회 | 매우 활발 | Cloud Native 최적화 | MSA 및 REST API 최적화 |
| **Spring Cloud Gateway** | 2022.0.x | 2027년 12월까지 | 연 2회 | 활발 | MSA 연동 최적화 | 경량화된 API Gateway |
| **Apache Kafka** | 3.4.x | 2027년 5월까지 | 연 2회 | 매우 활발 | 스트림 데이터 처리 최적화 | 고성능 메시지 처리 |
| **Kubernetes** | 1.26.x | 2027년 12월까지 | 3~4개월 | 매우 활발 | MSA 배포 자동화 | Auto Healing, 멀티 리전 배포 |
| **PostgreSQL** | 15.x | 2027년 9월까지 | 연 1회 | 활발 | 고성능 RDBMS | 무중단 트랜잭션 처리 |
| **Oracle DB** | 19c | 2027년 4월까지 | 3년 주기 | 안정적 | 금융권 최적화 | RAC 설정 가능 |
| **Redis** | 7.x | 2027년 9월까지 | 연 1회 | 매우 활발 | In-Memory 캐싱 최적화 | 고속 데이터 접근 |
| **Nginx** | 1.24.x | 2025년 7월까지 | 연 1회 | 매우 활발 | Reverse Proxy 최적화 | 고성능 로드 밸런싱 |
| **Prometheus & Grafana** | 2.40.x / 9.x | 2026년 7월까지 | 연 1회 | 매우 활발 | 실시간 모니터링 최적화 | Kubernetes 연동 최적화 |
| **Jenkins** | 2.375.x (LTS) | 2025년 12월까지 | 연 1회 | 매우 활발 | 파이프라인 최적화 | 무중단 배포 지원 |
| **GitLab CI** | 15.x | 2026년 8월까지 | 연 1회 | 매우 활발 | GitOps 최적화 | 코드 리뷰 및 CI/CD 통합 |
| **ELK Stack** | 8.x | 2026년 5월까지 | 연 1회 | 매우 활발 | 실시간 로그 분석 | 분산 로그 수집 최적화 |
| **Vault** | 1.14.x | 2027년 6월까지 | 연 1회 | 활발 | 비밀 관리 최적화 | 데이터 암호화 및 인증 관리 |
| **Wazuh** | 4.4.x | 2025년 9월까지 | 연 1회 | 활발 | 보안 탐지 최적화 | 실시간 모니터링 |

### ****장기 유지보수 전략****

|  |  |  |
| --- | --- | --- |
| 전략 항목 | 설명 | 적용 기술 |
| **LTS 기반 업그레이드** | 장기 지원 버전(LTS)을 기준으로 3~5년 주기로 업그레이드 계획 수립 | Spring Boot, Kubernetes, Redis |
| **하위 호환성 보장** | 주요 기술 스택은 하위 버전과의 호환성을 유지하여 Rollback이 가능하도록 설정 | Spring Cloud Gateway, PostgreSQL |
| **자동화된 보안 업데이트** | CI/CD 파이프라인을 통해 보안 패치를 자동화하고, 취약점 탐지를 실시간으로 모니터링 | Jenkins, GitLab CI |
| **멀티 리전 및 DR 구성** | 글로벌 금융 거래에 대응하기 위해 Multi-Region 배포와 DR(Disaster Recovery) 설정 | AWS, GCP, Oracle DB |
| **Auto Scaling 및 Auto Healing** | Kubernetes와 Prometheus를 활용하여 서비스의 확장성과 장애 복구를 자동화 | Kubernetes, Prometheus |
| **오픈 소스 커뮤니티 연계** | 주요 오픈 소스 기술의 커뮤니티 활동을 주기적으로 모니터링하고, 최신 패치와 업데이트를 반영 | Apache Kafka, ELK Stack |
| **Blue-Green Deployment 및 Canary Release** | 무중단 배포를 위한 배포 전략으로, 장애 발생 시 즉시 롤백이 가능하도록 설정 | Jenkins, ArgoCD |
| **로그 및 모니터링 통합 관리** | Prometheus, Grafana, ELK Stack을 통해 실시간 모니터링과 로그 분석을 통합 관리 | Prometheus, ELK Stack |

### ****기술 생태계 평가 및 시사점****

1. **LTS 지원 기술의 중요성**
   * 금융권의 장기적 안정성을 보장하기 위해 **LTS 지원**이 가능한 기술 스택을 선정하였습니다.
   * Spring Boot 3.x, Kubernetes 1.26.x, Redis 7.x는 2027년까지 지원되며, 안정적인 유지보수가 가능합니다.
2. **오픈 소스 커뮤니티의 활성화**
   * Apache Kafka, Spring Boot, Kubernetes는 매우 활발한 커뮤니티 활동을 가지고 있으며,  
     빠른 보안 패치와 업데이트가 제공됩니다.
3. **하이브리드 클라우드 최적화**
   * AWS, Azure, GCP의 글로벌 리전 배포와 OpenStack의 프라이빗 클라우드 확장이 가능합니다.
   * 금융권의 글로벌 트랜잭션에 최적화된 멀티 리전 아키텍처를 지원합니다.
4. **보안 최적화와 자동화된 대응**
   * Vault와 Wazuh를 통해 실시간 보안 탐지 및 데이터 암호화를 최적화하였으며,  
     Jenkins 및 GitLab CI와 연동된 보안 업데이트가 실시간으로 진행됩니다.

# ****6. Architecture Big Picture****

## ****전체 아키텍처 개요****

### ****개요****

* 금융권 시스템은 고가용성과 확장성을 극대화하기 위해 **MSA(Microservice Architecture)** 구조로 설계되었습니다.
* 각 서비스는 독립적으로 배포되며, API Gateway를 통해 클라이언트 요청을 라우팅합니다.
* 백엔드에서는 **Spring Boot** 기반의 마이크로서비스가 실행되며, **Kubernetes**를 통해 오토스케일링과 무중단 배포를 지원합니다.
* 데이터는 **Oracle DB**와 **PostgreSQL**로 관리되며, 빠른 조회를 위해 **Redis**가 In-Memory 캐싱을 담당합니다.
* 실시간 메시징은 **Apache Kafka**를 통해 비동기적으로 처리되며, 모든 로그와 트랜잭션은 **ELK Stack**에 저장됩니다.

### ****텍스트 기반 구성도****

|  |
| --- |
| +--------------------+ +-----------------------+ +--------------------+  | External Client | <------> | API Gateway | <-------> | Microservices |  +--------------------+ +-----------------------+ +--------------------+  | | | | |  v v v v v  +--------------------+ +-------------------+ +------------------+ +------------------+  | Authentication | | Service Discovery| | Load Balancer | | Rate Limiting |  | (OAuth2.0, JWT) | | (Eureka, Consul)| | (Nginx, HAProxy) | | (API Gateway) |  +--------------------+ +-------------------+ +------------------+ +------------------+  | | | | |  v v v v v  +------------------+ +------------------+ +------------------+ +------------------+  | Spring Boot | | Spring Boot | | Spring Boot | | Spring Boot |  | Payment Service| | Account Service | | Transfer Service | | Notification |  +------------------+ +------------------+ +------------------+ +------------------+  | | | |  v v v v  +--------------------+ +--------------------+ +--------------------+ +--------------------+  | PostgreSQL | | Oracle DB | | Redis Cache | | Apache Kafka |  +--------------------+ +--------------------+ +--------------------+ +--------------------+  | | | |  v v v v  +------------------------+ +------------------------+ +------------------------+  | ELK Stack (Logs) | | Prometheus (Metrics) | | Grafana (Monitoring) |  +------------------------+ +------------------------+ +------------------------+ |

## ****아키텍처 계층 구조****

### ****개요****

* 아키텍처는 5계층 구조로 나누어지며, 각각의 역할이 명확히 정의되어 있습니다.
* 계층 구조는 **Presentation Layer, Gateway Layer, Business Logic Layer, Data Layer, Monitoring Layer**로 나누어집니다.

### ****아키텍처 계층 설명****

|  |  |  |
| --- | --- | --- |
| 계층 | 구성 요소 | 역할 및 설명 |
| **Presentation Layer** | External Client, Frontend (React, Vue) | 사용자 인터페이스, REST API를 통한 클라이언트 요청 전달 |
| **Gateway Layer** | API Gateway (Spring Cloud Gateway, Kong) | 인증, 라우팅, Rate Limiting, CORS 관리 |
| **Business Logic Layer** | Microservices (Spring Boot) | 비즈니스 로직 처리, MSA 구조 기반 서비스 분리 |
| **Data Layer** | PostgreSQL, Oracle DB, Redis | 트랜잭션 처리, 데이터 캐싱, 분산 데이터 저장 |
| **Monitoring Layer** | Prometheus, Grafana, ELK Stack | 실시간 모니터링, 로그 분석, 장애 탐지 및 대응 |

## ****통합 아키텍처 다이어그램****

### ****개요****

* 전체 금융 시스템의 흐름을 한눈에 파악할 수 있도록 **통합 아키텍처 다이어그램**을 구성합니다.
* API Gateway를 중심으로 **Microservices, Message Broker, Data Store**가 유기적으로 연결됩니다.

### ****텍스트 기반 구성도****

|  |
| --- |
| +---------------------------+  | External Client |  +---------------------------+  |  v  +---------------------------+  | API Gateway |  | (Spring Cloud Gateway) |  +---------------------------+  |  +-----------------+-----------------+  | |  +-------------------+ +-------------------+  | Authentication | | Rate Limiting |  | (OAuth2.0, JWT) | | |  +-------------------+ +-------------------+  | |  v v  +---------------------+ +---------------------+  | Microservice A | | Microservice B |  | (Account Service) | | (Payment Service) |  +---------------------+ +---------------------+  | |  v v  +---------------------+ +---------------------+  | PostgreSQL | | Oracle DB |  | (Account Data) | | (Transaction Data) |  +---------------------+ +---------------------+  | |  v v  +---------------------+ +---------------------+  | Apache Kafka | | Redis Cache |  | (Async Messaging) | | (In-Memory Store) |  +---------------------+ +---------------------+  |  v  +---------------------+  | ELK Stack (Logs) |  +---------------------+ |

## ****계층 흐름 및 시스템 분리****

### ****개요****

* 금융권 시스템에서 고성능과 안정성을 보장하기 위해 **계층 간 분리**와 **연계 전략**이 필수적입니다.
* **Presentation Layer → Gateway Layer → Business Logic Layer → Data Layer → Monitoring Layer** 순으로 흐름이 정의되며,  
  각 계층은 독립적으로 관리되고 확장될 수 있도록 설계되었습니다.

### ****계층 흐름 구성도****

|  |
| --- |
| +----------------------------+  | External Client |  +----------------------------+  |  v  +----------------------------+  | API Gateway Layer |  | - Spring Cloud Gateway |  | - Kong API Gateway |  +----------------------------+  |  v  +----------------------------+  | Business Logic Layer |  | - Spring Boot Services |  | - Microservices (MSA) |  +----------------------------+  | |  v v  +-------------+ +-------------+  | Data Layer | | Message Bus |  | - PostgreSQL| | - Kafka |  | - Oracle DB | | - RabbitMQ |  | - Redis | +-------------+  +-------------+  |  v  +----------------------------+  | Monitoring Layer |  | - Prometheus |  | - Grafana |  | - ELK Stack |  +----------------------------+ |

### ****계층 흐름 설계****

* 각 계층 간의 데이터 전송 및 상호작용은 다음과 같이 정의됩니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 계층 | 주요 기술 | 역할 및 설명 | 전송 방식 | 주요 연계 기술 |
| **Presentation Layer** | React, Vue.js | 클라이언트의 요청을 API Gateway로 전달 | HTTPS, REST API | Spring Cloud Gateway |
| **Gateway Layer** | Spring Cloud Gateway, Kong | 요청 라우팅, 인증 처리 (OAuth2.0, JWT), CORS 관리 | HTTP, HTTPS | Eureka, Consul |
| **Business Logic Layer** | Spring Boot, Microservices | 비즈니스 로직 처리, MSA 구조로 서비스 분리 | gRPC, REST API | Apache Kafka, RabbitMQ |
| **Data Layer** | PostgreSQL, Oracle DB, Redis | 금융 거래 처리, 데이터 저장 및 조회, 캐싱 처리 | JDBC, JPA | Spring Data, MyBatis |
| **Monitoring Layer** | Prometheus, Grafana, ELK Stack | 실시간 모니터링, 트랜잭션 로그 수집, 장애 탐지 | HTTP, Metrics API | Fluentd, Logstash |

### ****시스템 분리 전략****

#### ****개요****

* 금융권 시스템의 안정성과 확장성을 위해 **수평적 분리**와 **수직적 분리**를 전략적으로 적용합니다.
* **수평적 분리**는 트랜잭션이 높은 서비스의 부하를 분산하며, **수직적 분리**는 기능별로 독립된 관리가 가능합니다.

#### ****수평적 분리 전략****

* 동일한 기능을 여러 인스턴스에 배치하여 **로드 밸런싱**을 통해 부하를 분산합니다.
* 트랜잭션이 집중되는 금융 거래, 실시간 결제 API, 대량 조회 서비스에 적용됩니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 서비스 | 인스턴스 수 | 로드 밸런싱 방식 | 기술 스택 |
| **Payment Service** | 5 | Round Robin | Spring Boot, PostgreSQL |
| **Account Service** | 3 | Least Connection | Spring Boot, Oracle DB |
| **Transaction Service** | 4 | IP Hash | Spring Boot, Kafka |
| **Notification Service** | 2 | Random | Spring Boot, Redis |
| **Log Service** | 2 | Round Robin | ELK Stack |

#### ****수직적 분리 전략****

* 비즈니스 로직을 **서비스 도메인**별로 나누어 독립적인 배포 및 확장을 지원합니다.
* 예를 들어, Account Management, Transaction Processing, Notification Handling으로 구분됩니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 도메인 | 주요 기능 | 기술 스택 | 연계 서비스 |
| **Account Management** | 계좌 생성, 조회, 수정 | Spring Boot, PostgreSQL | Notification Service |
| **Transaction Processing** | 송금, 결제 처리, 입출금 관리 | Spring Boot, Oracle DB | Payment Gateway |
| **Notification Handling** | 이메일, 문자 알림 발송 | Spring Boot, Redis | Account Service, Transaction Service |
| **Fraud Detection** | 이상 거래 탐지, 실시간 차단 | Spring Boot, Apache Kafka | Transaction Processing |
| **Audit & Logging** | 트랜잭션 기록, 감사 로그 수집 | ELK Stack | 모든 서비스에서 연동 |

### ****분석 및 시사점****

1. **계층 간 명확한 역할 구분**
   * Gateway, Business Logic, Data, Monitoring Layer가 독립적으로 설계되어 확장성과 관리 용이성이 극대화됩니다.
2. **수평적/수직적 분리의 최적화**
   * 트랜잭션이 집중되는 Payment, Transaction Service는 **수평적 분리**로 부하를 분산합니다.
   * 각 도메인은 **수직적 분리**로 독립적으로 확장 및 장애 대응이 가능합니다.
3. **모니터링의 실시간 추적**
   * Prometheus와 Grafana를 통해 서비스 상태를 실시간으로 추적하고, ELK Stack으로 모든 트랜잭션 로그를 수집합니다.

## ****인터페이스 구조 및 연계 전략****

### ****인터페이스 구조 개요****

#### ****개요****

* 금융 시스템의 데이터 흐름을 원활하게 처리하기 위해 **표준화된 인터페이스 구조**가 필요합니다.
* 모든 서비스 간 통신은 **REST API**, **gRPC**, **Message Queue**를 통해 비동기적이고 안정적으로 이루어집니다.
* 특히, MSA 환경에서는 서비스 간 데이터 전달 시 **트랜잭션 보장**, **유효성 검사**, **오류 처리**가 중요합니다.

#### ****텍스트 기반 구성도****

|  |
| --- |
| +-----------------------+  | External Client |  +-----------------------+  |  v  +-----------------------+  | API Gateway |  | (Spring Cloud Gateway)|  +-----------------------+  |  v  +-----------------------+  | Microservice A |  | (Account Service) |  +-----------------------+  |  v  +-----------------------+ +-----------------------+  | Message Queue | <---> | Microservice B |  | (Apache Kafka) | | (Payment Service) |  +-----------------------+ +-----------------------+  | |  v v  +-----------------------+ +-----------------------+  | Database (Postgres) | | Database (Oracle) |  +-----------------------+ +-----------------------+ |

### ****연계 기술 스펙 및 구성도****

#### ****개요****

* 금융권의 안정적 서비스 연계를 위해 표준화된 **연계 기술 스펙**을 정의합니다.
* 연계 방식은 다음과 같습니다:
  + **API 연계:** REST API, GraphQL
  + **메시지 연계:** Apache Kafka, RabbitMQ
  + **파일 연계:** SFTP, AWS S3
  + **배치 연계:** Spring Batch, Quartz Scheduler

#### ****연계 기술 스펙 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 연계 방식 | 사용 기술 | 프로토콜 | 주요 특성 | 적용 사례 |
| **API 연계** | Spring Cloud Gateway, Kong | REST, HTTPS | 실시간 데이터 전송, 동기 호출 | 거래 조회, 송금 요청 |
| **메시지 연계** | Apache Kafka, RabbitMQ | AMQP, gRPC | 비동기 메시지 처리, 대량 트랜잭션 처리 | 실시간 알림, 결제 승인 |
| **파일 연계** | AWS S3, MinIO, SFTP | SFTP, HTTP | 대용량 파일 전송, 정기 배치 | 금융 거래 내역 전송, PDF 청구서 |
| **배치 연계** | Spring Batch, Quartz | HTTP, REST | 정기적인 데이터 처리, 야간 배치 | 일일 정산, 데이터 마이그레이션 |

### ****연계 표준 설계****

#### ****개요****

* 금융 시스템 내 서비스 간의 통신을 표준화하여 **상호 운용성**과 **확장성**을 높입니다.
* 모든 API는 **OpenAPI Specification**을 기반으로 설계되며, **JSON Schema Validation**을 통해 유효성을 검증합니다.
* 메시지 큐를 사용할 경우 **Avro Schema**로 표준화하여 시스템 간 연계를 최적화합니다.

#### ****연계 표준 설계 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 표준화 방식 | 적용 기술 |
| **Request Format** | API 요청 시 JSON 기반 데이터 전달 | OpenAPI Specification | Spring Cloud Gateway |
| **Response Format** | API 응답 시 표준화된 JSON 반환 | OpenAPI Specification | Spring Boot |
| **Message Format** | 비동기 메시지 처리 시 Avro 사용 | Avro Schema | Apache Kafka |
| **Error Handling** | 표준화된 에러 메시지 규격화 | HTTP Status Code, Error Message | Spring Boot, Kong |
| **Data Validation** | 전송되는 데이터에 대한 유효성 검사 | JSON Schema Validation | Spring Validator |
| **Security** | OAuth2.0, JWT 기반 인증 | OAuth2.0, JWT | Spring Security |
| **Version Control** | API의 버전 관리 | URI Versioning (v1, v2) | Spring Cloud Gateway |

### ****연계 안정화 및 오류 대응 전략****

#### ****개요****

* 금융 시스템의 안정성을 보장하기 위해 **트랜잭션 보장**, **오류 탐지**, **Failover 처리**를 표준화합니다.
* 특히, \*\*2PC(Two-Phase Commit)\*\*과 **Saga Pattern**을 통해 트랜잭션의 원자성을 유지합니다.

#### ****연계 안정화 및 오류 대응 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 적용 기술 | 대응 전략 |
| **트랜잭션 보장** | 다중 서비스 간 트랜잭션 처리 시 일관성 유지 | 2PC, Saga Pattern | 트랜잭션 실패 시 롤백 처리 |
| **오류 탐지** | 네트워크 장애 및 API 실패 탐지 | Prometheus, ELK Stack | 장애 발생 시 실시간 알림 |
| **Failover 처리** | 서비스 중단 시 다른 인스턴스로 전환 | Kubernetes Auto Scaling | Active-Active, Active-Passive |
| **Retry Mechanism** | 네트워크 오류 발생 시 재시도 처리 | Spring Retry | 최대 3회 재시도 후 Alert |
| **Circuit Breaker** | 장애 발생 시 전파 차단 | Resilience4J | 임계치 초과 시 트래픽 차단 |
| **Load Balancing** | 트래픽 부하를 다수 서버로 분산 | Nginx, HAProxy | Round Robin, Least Connection |
| **Rate Limiting** | API 요청 폭주 시 트래픽 제한 | Kong, Spring Cloud Gateway | 1초당 최대 100 TPS 제한 |

### ****인터페이스 버전 관리 및 확장 전략****

#### ****개요****

* 금융 시스템은 지속적으로 변경되기 때문에 **API 버전 관리**와 **인터페이스 확장**이 필수적입니다.
* 버전 관리는 \*\*URI Versioning (v1, v2)\*\*를 통해 명확하게 구분되며, 하위 호환성을 보장합니다.
* 확장 전략은 **Backward Compatibility**를 유지하면서 **New Version Release**를 병행합니다.

#### ****버전 관리 및 확장 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 적용 기술 | 주요 특징 |
| **Versioning** | API 변경 시 명확한 버전 구분 | URI Versioning (v1, v2) | 하위 호환성 유지 |
| **Deprecation Policy** | 오래된 API의 단계적 폐기 | Spring Cloud Gateway | 6개월 사전 공지 후 종료 |
| **Backward Compatibility** | 기존 클라이언트의 오류 방지 | Content Negotiation | 구버전 요청 시 대응 |
| **Schema Evolution** | 메시지 포맷 변경 시 유연한 대응 | Avro Schema | 동적 확장 지원 |
| **New Release Strategy** | 신규 버전 병행 배포 | Blue-Green Deployment | 무중단 배포 지원 |

# ****성능, 보안, 가용성, 장애 대응 전략****

## ****성능 최적화 전략****

### ****개요****

* 금융권 시스템에서 성능 최적화는 대량 트랜잭션 처리와 실시간 데이터 조회 속도를 극대화하는 것이 목표입니다.
* 주요 기술로는 **Spring Boot 최적화**, **Database Indexing**, **Redis 캐싱**, **Apache Kafka 비동기 처리**를 활용합니다.
* 또한, **Connection Pooling**과 **Lazy Loading**을 통해 DB 연결 최적화가 이루어집니다.

### ****성능 최적화 전략 구성도****

sql

복사편집

+----------------------------+

| API Gateway |

| (Spring Cloud Gateway) |

+----------------------------+

|

v

+----------------------------+

| Microservices |

| (Spring Boot, MSA) |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

| (Indexing, JPA) | | (Stored Procedure)| | (In-Memory) |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Apache Kafka |

| (Async Message Queue) |

+----------------------------+

### ****성능 최적화 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 최적화 항목 | 설명 | 적용 기술 | 기대 효과 |
| **Connection Pooling** | 다수의 DB 연결을 미리 생성하여 처리 속도 향상 | HikariCP, Tomcat JDBC Pool | DB 연결 시간 최소화 |
| **Lazy Loading** | 실제 요청 시점에만 데이터를 로딩 | JPA, Hibernate | 메모리 사용 최적화 |
| **Database Indexing** | 자주 조회되는 컬럼에 인덱스 설정 | PostgreSQL, Oracle DB | 조회 성능 30% 이상 개선 |
| **Redis Cache** | 조회 속도를 높이기 위한 In-Memory 캐싱 | Redis | 조회 응답 시간 50% 감소 |
| **Asynchronous Processing** | 비동기 메시지 처리로 처리 속도 향상 | Apache Kafka, RabbitMQ | 대량 트랜잭션 처리 최적화 |
| **Load Balancing** | 요청을 여러 서버에 분산 처리 | Nginx, HAProxy | 트래픽 병목 현상 최소화 |
| **Batch Processing 최적화** | 야간 배치 시간 단축 | Spring Batch, Quartz | 처리 속도 40% 향상 |
| **Data Sharding** | 대규모 데이터를 분할하여 저장 | PostgreSQL, MongoDB | 조회 성능 극대화 |
| **File Upload 최적화** | 대용량 파일 처리 속도 향상 | AWS S3, MinIO | 전송 속도 30% 향상 |
| **Paging 처리 최적화** | 대량 데이터 페이지 처리 속도 향상 | Spring Data JPA | 메모리 사용 최소화 |

## ****보안 설계 전략****

### ****개요****

* 금융 시스템의 보안은 **OAuth2.0, JWT(Json Web Token), TLS/SSL 암호화**를 통해 **인증, 권한 관리, 데이터 보호**를 강화합니다.
* 주요 보안 정책은 **API 보안, 데이터 암호화, 접근 제어, DDoS 방어**로 구성됩니다.
* 또한, **OWASP Top 10**을 기준으로 주요 취약점(XSS, SQL Injection)을 방어합니다.

### ****보안 설계 전략 구성도****

pgsql

복사편집

+----------------------------+

| External Client |

+----------------------------+

|

v

+----------------------------+

| API Gateway |

| - OAuth2.0 인증 |

| - Rate Limiting |

| - CORS 관리 |

+----------------------------+

|

v

+----------------------------+

| Microservices |

| - Spring Security |

| - JWT 인증 |

+----------------------------+

|

v

+----------------------------+

| Data Layer |

| - Encryption (AES-256) |

| - TDE (Transparent Data Encryption) |

+----------------------------+

### ****보안 설계 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 보안 항목 | 설명 | 적용 기술 | 주요 대응 |
| **인증 및 권한 관리** | 사용자 인증 및 접근 제어 | OAuth2.0, JWT | 무단 접근 방지 |
| **전송 구간 암호화** | 네트워크 전송 시 데이터 보호 | TLS/SSL | 중간자 공격 방어 |
| **데이터 암호화** | 민감한 정보 암호화 저장 | AES-256, RSA | 데이터 유출 방지 |
| **API 보안** | 인증된 요청만 허용 | API Gateway, Kong | Rate Limiting, CORS 관리 |
| **DDoS 방어** | 대량 트래픽 공격 차단 | AWS Shield, Nginx | 실시간 트래픽 필터링 |
| **SQL Injection 방지** | SQL Query Injection 차단 | Spring Data JPA | PreparedStatement 사용 |
| **XSS 방어** | 클라이언트 스크립트 공격 방지 | Spring Security | Output Encoding 적용 |
| **접근 통제** | 리소스별 권한 제어 | RBAC (Role-Based Access Control) | 중요 리소스 접근 보호 |
| **로그 및 감사 기록** | 접근 이력과 변경 내역 추적 | ELK Stack | 감사 기록 저장 |
| **보안 모니터링** | 실시간 보안 이벤트 추적 | Prometheus, Grafana | 실시간 알림 |

## ****가용성 및 장애 대응 전략****

### ****개요****

* 금융 시스템의 무중단 서비스 제공을 위해 **이중화**, **Failover**, **Auto Healing**이 설계되었습니다.
* Kubernetes 클러스터의 Auto Scaling과 Prometheus 모니터링을 통해 장애 발생 시 자동 복구가 이루어집니다.

### ****가용성 및 장애 대응 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 적용 기술 | 주요 대응 |
| **이중화 설정** | Active-Active, Active-Passive 구성 | Nginx, HAProxy | 트래픽 분산 및 장애 복구 |
| **Failover 처리** | 노드 장애 시 자동 전환 | Kubernetes, Consul | 무중단 서비스 전환 |
| **Auto Healing** | 서비스 장애 시 자동 복구 | Kubernetes, Prometheus | Pod 자동 재시작 |
| **Auto Scaling** | 트래픽 증가 시 자동 확장 | Kubernetes HPA | 최대 처리량 보장 |
| **Disaster Recovery** | 지역 간 복구 시나리오 설정 | AWS Multi-AZ, GCP Multi-Region | 데이터 손실 방지 |
| **Health Check** | 서비스 상태 점검 | Nginx, HAProxy | 장애 탐지 시 자동 우회 |
| **Rolling Update** | 무중단 배포 전략 | Kubernetes, ArgoCD | Zero Downtime 배포 |
| **Blue-Green Deployment** | 새로운 버전의 안전한 배포 | Jenkins, ArgoCD | 배포 실패 시 즉시 복구 |
| **Canary Release** | 단계적 배포로 리스크 최소화 | Jenkins, Spinnaker | 테스트 후 전체 배포 |

# ****성능 테스트 및 검증 전략****

## ****성능 테스트 계획****

### ****개요****

* 금융 시스템의 고성능을 보장하기 위해 **TPS (Transactions Per Second)**, **응답 시간**, **동시 사용자 처리**에 대한 성능 테스트를 수행합니다.
* 테스트는 **부하 테스트(Load Test)**, **스트레스 테스트(Stress Test)**, **내구성 테스트(Soak Test)**, \*\*스파이크 테스트(Spike Test)\*\*로 구분됩니다.
* 주요 목표는 **10,000 TPS**와 **3초 이하의 응답 시간**을 보장하는 것입니다.

### ****성능 테스트 구성도****

sql

복사편집

+----------------------------+

| External Client |

+----------------------------+

|

v

+----------------------------+

| Load Balancer |

| (Nginx, HAProxy) |

+----------------------------+

|

v

+----------------------------+

| API Gateway |

| (Spring Cloud Gateway) |

+----------------------------+

|

v

+----------------------------+

| Application |

| (Spring Boot Services) |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Monitoring Tools |

| (Prometheus, Grafana) |

+----------------------------+

### ****성능 테스트 계획표****

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 유형 | 설명 | 목적 | 주요 기술 |
| **부하 테스트 (Load Test)** | 예상 사용자의 트래픽을 시뮬레이션하여 처리 성능 측정 | 10,000 TPS 보장 | JMeter, Gatling |
| **스트레스 테스트 (Stress Test)** | 최대 처리 한계를 초과하여 시스템 반응 측정 | 최대 동시 사용자 20,000명 | Locust, Tsung |
| **내구성 테스트 (Soak Test)** | 장시간 동안 지속적인 트래픽 처리 성능 측정 | 24시간 이상 무중단 처리 | JMeter, BlazeMeter |
| **스파이크 테스트 (Spike Test)** | 짧은 시간 동안 급격한 트래픽 증가 시 반응 확인 | 5배 이상의 트래픽 증가 | K6, Gatling |
| **단위 성능 테스트 (Unit Performance Test)** | 개별 모듈의 성능 최적화 테스트 | 주요 서비스 3초 이내 응답 | Spring Boot Test |
| **통합 성능 테스트 (Integration Performance Test)** | 모듈 간 연계 시 성능 최적화 테스트 | API 간 100ms 이하 응답 | Postman, Newman |

## ****성능 지표 설정 및 검증****

### ****개요****

* 성능 검증 시 다음의 주요 지표를 측정합니다:
  + **TPS (Transactions Per Second):** 초당 처리 가능한 트랜잭션 수
  + **응답 시간:** 요청에 대한 응답이 완료되는 시간
  + **CPU/Memory Usage:** 처리 중 시스템 리소스 사용량
  + **Throughput:** 네트워크를 통해 전송된 데이터 양
  + **Error Rate:** 트랜잭션 실패 비율

### ****성능 지표 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 성능 지표 | 설명 | 목표 값 | 측정 도구 |
| **TPS (Transactions Per Second)** | 초당 처리되는 트랜잭션 수 | 10,000 TPS | JMeter, Gatling |
| **Response Time** | 요청 후 응답이 완료되는 시간 | 3초 이내 | K6, Locust |
| **CPU Usage** | 트랜잭션 처리 시 사용된 CPU 비율 | 70% 이하 | Prometheus, Grafana |
| **Memory Usage** | 트랜잭션 처리 시 사용된 메모리 비율 | 75% 이하 | Prometheus, Grafana |
| **Throughput** | 초당 전송된 데이터 양 | 100MB/s 이상 | Grafana, ELK Stack |
| **Error Rate** | 요청 중 에러 발생 비율 | 0.01% 이하 | ELK Stack, Prometheus |
| **Latency** | 요청 간 대기 시간 | 100ms 이하 | K6, Prometheus |

## ****테스트 도구 및 시나리오****

### ****개요****

* 금융권 시스템의 성능 테스트를 위해 최적화된 테스트 도구를 선정하였으며,  
  다양한 시나리오를 바탕으로 실시간 성능 분석을 진행합니다.
* 도구는 **JMeter, Gatling, Locust, K6**가 사용되며, 실시간 모니터링은 **Prometheus, Grafana**를 통해 진행됩니다.

### ****테스트 도구 구성****

|  |  |  |
| --- | --- | --- |
| 테스트 도구 | 주요 기능 | 적용 시나리오 |
| **JMeter** | HTTP, REST API, DB 테스트 | 부하 테스트, 내구성 테스트 |
| **Gatling** | HTTP 요청 및 동시 사용자 처리 | 스파이크 테스트, 스트레스 테스트 |
| **Locust** | Python 기반의 분산 부하 테스트 | 스트레스 테스트, 대규모 사용자 처리 |
| **K6** | CLI 기반의 부하 테스트 | 스파이크 테스트, 성능 모니터링 |
| **BlazeMeter** | JMeter 확장, 클라우드 기반 테스트 | 내구성 테스트, 글로벌 트래픽 테스트 |
| **Prometheus** | 실시간 메트릭 수집 | CPU, Memory 사용량 모니터링 |
| **Grafana** | 시각적 대시보드 | TPS, 응답 시간, 에러율 분석 |
| **ELK Stack** | 로그 수집 및 분석 | 에러 탐지 및 실시간 모니터링 |

### ****테스트 시나리오 예시****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 시나리오 | 설명 | 목표 TPS | 주요 측정 지표 | 성공 기준 |
| **대량 송금 처리** | 동시 10,000명 송금 요청 | 10,000 TPS | Response Time, Error Rate | 3초 이내 응답, Error Rate 0.01% 이하 |
| **실시간 거래 조회** | 고객의 거래 내역을 실시간 조회 | 5,000 TPS | Latency, CPU Usage | 100ms 이하 응답 |
| **야간 배치 처리** | 야간 시간대에 일괄 처리 | 1,000 TPS | Throughput, Memory Usage | 70% 이하 Memory Usage |
| **급격한 트래픽 처리** | 특정 시간대 대량 로그인 요청 | 15,000 TPS | TPS, CPU Usage | 3초 이내 응답 |
| **API Gateway 부하 처리** | API Gateway를 통한 다중 서비스 요청 | 20,000 TPS | Response Time, Latency | 5초 이내 응답 |

# ****데이터 백업 및 복구 전략****

## ****데이터 백업 설계****

### ****개요****

* 금융권 시스템은 **실시간 트랜잭션 처리**와 **대용량 데이터 저장소**를 안전하게 보관하기 위해 **다계층 백업 전략**을 필요로 합니다.
* 주요 백업 유형은 다음과 같습니다:
  + **전체 백업 (Full Backup):** 모든 데이터를 주기적으로 백업
  + **증분 백업 (Incremental Backup):** 변경된 데이터만 백업
  + **차등 백업 (Differential Backup):** 마지막 전체 백업 이후 변경된 데이터만 백업
  + **스냅샷 (Snapshot):** 특정 시점의 데이터 상태를 빠르게 보관

### ****데이터 백업 구성도****

pgsql

복사편집

+---------------------------+

| Database (Oracle, |

| PostgreSQL, Redis) |

+---------------------------+

|

v

+---------------------------+

| Full Backup (Daily) |

| - Oracle RMAN |

| - PostgreSQL Dump |

+---------------------------+

|

v

+---------------------------+

| Incremental Backup (1h) |

| - WAL(Log Shipping) |

| - Amazon S3 |

+---------------------------+

|

v

+---------------------------+

| Snapshot Backup (5m) |

| - AWS RDS Snapshot |

| - Redis Persistence |

+---------------------------+

### ****백업 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 백업 유형 | 설명 | 주기 | 기술 스택 | 주요 특징 |
| **전체 백업 (Full Backup)** | 모든 데이터를 주기적으로 백업 | 매일 자정 (00:00) | Oracle RMAN, PostgreSQL Dump | 복구 시점이 명확하고 완전한 복구 가능 |
| **증분 백업 (Incremental Backup)** | 변경된 데이터만 백업 | 매 1시간 | WAL(Log Shipping), Amazon S3 | 백업 시간 최소화, 용량 최적화 |
| **차등 백업 (Differential Backup)** | 마지막 전체 백업 이후 변경된 데이터만 백업 | 매 6시간 | PostgreSQL Delta, Oracle Archive Log | 복구 속도 최적화 |
| **스냅샷 백업 (Snapshot Backup)** | 특정 시점의 데이터를 이미지화하여 보관 | 매 5분 | AWS RDS Snapshot, EBS Snapshot | 빠른 복구 시간, 실시간 복구 가능 |
| **트랜잭션 로그 백업** | 트랜잭션 변경 사항만 실시간 백업 | 실시간 | Kafka Log Stream, Redis RDB | 실시간 데이터 복구 가능 |

## ****데이터 복구 전략****

### ****개요****

* 데이터 손실 및 장애 발생 시, **최소한의 다운타임**으로 복구하기 위한 전략을 설계합니다.
* 복구 전략은 \*\*RTO (Recovery Time Objective)\*\*와 \*\*RPO (Recovery Point Objective)\*\*를 기준으로 최적화합니다.
  + **RTO:** 장애 발생 후 시스템이 복구되는 시간
  + **RPO:** 데이터 손실 허용 시간

### ****복구 전략 구성도****

pgsql

복사편집

+----------------------------+

| Backup Storage |

| - Amazon S3 |

| - Oracle RMAN Backups |

| - PostgreSQL WAL Logs |

+----------------------------+

|

v

+----------------------------+

| Disaster Recovery (DR) |

| - Multi-AZ Replication |

| - Read Replica Failover |

+----------------------------+

|

v

+----------------------------+

| Restoration Process |

| - Point-in-Time Recovery|

| - Full Backup Restore |

| - Incremental Restore |

+----------------------------+

### ****복구 전략 표****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 복구 유형 | 설명 | RTO | RPO | 기술 스택 | 주요 특징 |
| **Full Backup Restore** | 전체 백업본을 사용하여 복구 | 30분 | 24시간 | Oracle RMAN, PostgreSQL Dump | 데이터 손실 없이 복구 가능 |
| **Incremental Restore** | 변경된 데이터만 복원하여 복구 시간 단축 | 15분 | 1시간 | WAL Shipping, S3 Incremental | 빠른 복구 속도, 용량 최적화 |
| **Snapshot Restore** | 특정 시점의 스냅샷을 사용하여 빠른 복구 | 5분 | 5분 | AWS RDS Snapshot | 실시간 복구, 대기 시간 최소화 |
| **Transaction Log Restore** | 트랜잭션 로그를 사용하여 미러링 복구 | 1분 | 1분 | Kafka, Redis RDB | 실시간 데이터 복구 가능 |
| **Failover to Replica** | Master 장애 시 Read Replica로 전환 | 1분 | 실시간 | PostgreSQL Read Replica, Oracle RAC | 서비스 중단 없이 즉시 전환 |

## ****복구 시나리오 및 테스트 계획****

### ****개요****

* 금융권 시스템의 데이터 복구 시나리오는 다음과 같은 케이스로 구분됩니다:
  + **단일 DB 장애 복구**
  + **클러스터 장애 복구**
  + **지역(Region) 장애 복구**
  + **데이터 손실 시 복구**

### ****복구 시나리오 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 시나리오 | 설명 | 주요 기술 | 복구 전략 | 예상 RTO |
| **단일 DB 장애 복구** | 특정 DB 인스턴스에 장애 발생 시 복구 | PostgreSQL, Oracle DB | Snapshot Restore + Incremental Recovery | 15분 |
| **클러스터 장애 복구** | Kubernetes 클러스터의 전체 장애 시 복구 | Kubernetes, Prometheus | Auto Healing + Failover | 5분 |
| **지역(Region) 장애 복구** | Multi-Region 간 장애 발생 시 복구 | AWS Multi-AZ, GCP Multi-Region | DR(Disaster Recovery) 설정 | 10분 |
| **데이터 손실 시 복구** | 주요 트랜잭션 손실 발생 시 복구 | Kafka Log, WAL Log | Point-in-Time Recovery | 2분 |
| **네트워크 분리 시 복구** | 특정 네트워크가 단절되었을 때 복구 | Nginx, HAProxy | Auto Reroute 설정 | 1분 |

# ****로그 및 모니터링 전략****

## ****로그 수집 및 분석 설계****

### ****개요****

* 금융권 시스템의 **실시간 거래, 인증, 에러 발생**을 추적하기 위해 **로그 수집 및 분석**이 필수적입니다.
* 로그 수집은 **ELK Stack(Elasticsearch, Logstash, Kibana)**, **Fluentd**, **Prometheus**를 기반으로 합니다.
* 분석된 로그는 **대시보드 시각화**와 **장애 탐지**에 활용됩니다.

### ****로그 수집 및 분석 구성도****

pgsql

복사편집

+----------------------------+

| Application |

| - Spring Boot Services |

+----------------------------+

|

v

+----------------------------+

| Fluentd Agent |

| - Log Forwarder |

+----------------------------+

|

v

+----------------------------+

| Logstash |

| - Data Processing |

+----------------------------+

|

v

+----------------------------+

| Elasticsearch |

| - Log Indexing |

+----------------------------+

|

v

+----------------------------+

| Kibana |

| - Log Visualization |

+----------------------------+

### ****로그 수집 및 분석 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **로그 수집** | 애플리케이션 및 시스템 로그 실시간 수집 | Fluentd, Filebeat | 비동기 전송, 경량화된 에이전트 |
| **로그 처리** | 수집된 로그의 정제 및 필터링 | Logstash | JSON, XML, CSV 처리 |
| **로그 저장** | 실시간 색인 및 분석을 위한 저장 | Elasticsearch | 분산 처리, 실시간 색인 |
| **로그 분석** | 대시보드 생성 및 실시간 분석 | Kibana, Grafana | 시각화 및 실시간 모니터링 |
| **에러 탐지** | 오류 발생 시 빠른 탐지 및 알림 | Prometheus AlertManager | 실시간 경고 전송 |
| **모니터링 연동** | 로그 데이터 기반 실시간 상태 확인 | Grafana | TPS, 응답 시간 분석 |

## ****모니터링 아키텍처 설계****

### ****개요****

* 금융 시스템의 **트랜잭션 상태, API 응답 시간, 서버 리소스 사용량**을 실시간으로 추적합니다.
* 모니터링 시스템은 **Prometheus**, **Grafana**, **AlertManager**를 중심으로 구성되며,  
  **Kubernetes 클러스터**와 **Microservice 상태**를 지속적으로 관찰합니다.

### ****모니터링 아키텍처 구성도****

sql

복사편집

+----------------------------+

| Application Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Prometheus Node |

| - Metric Collection |

+----------------------------+

|

v

+----------------------------+

| Prometheus DB |

| - Data Aggregation |

+----------------------------+

|

v

+----------------------------+

| Grafana |

| - Dashboard Visualization |

+----------------------------+

|

v

+----------------------------+

| AlertManager |

| - Real-time Alerts |

+----------------------------+

### ****모니터링 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **메트릭 수집** | CPU, Memory, Disk I/O, Network 상태 추적 | Prometheus Node Exporter | 실시간 상태 모니터링 |
| **컨테이너 모니터링** | Kubernetes Pod 상태, 서비스 헬스 체크 | Prometheus K8s Adapter | 클러스터 단위 모니터링 |
| **대시보드 시각화** | 트랜잭션 처리 상태, TPS, 응답 시간 분석 | Grafana | 실시간 차트 및 대시보드 |
| **알림 설정** | 임계치 초과 시 즉시 경고 전송 | AlertManager | Slack, Email, SMS 연동 |
| **트랜잭션 추적** | API 간 호출 상태 및 응답 시간 분석 | Grafana Tempo, Jaeger | 서비스 간 Latency 분석 |
| **이벤트 분석** | 서비스 중단, 장애 탐지 및 복구 | Prometheus AlertManager | 자동화된 복구 시나리오 연동 |

## ****장애 탐지 및 알림 체계****

### ****개요****

* 금융 시스템의 신속한 장애 대응을 위해 **자동 탐지 및 알림 체계**가 구축됩니다.
* 장애 탐지는 **Prometheus AlertManager**가 담당하며, **Slack, Email, SMS**를 통해 실시간으로 전파됩니다.
* 알림 발생 시 **자동화된 대응 프로세스**가 실행되어 시스템의 복구를 최적화합니다.

### ****장애 탐지 및 알림 구성도****

pgsql

복사편집

+----------------------------+

| Application Services |

+----------------------------+

|

v

+----------------------------+

| Prometheus Exporter |

| - Health Check |

| - Error Monitoring |

+----------------------------+

|

v

+----------------------------+

| AlertManager |

| - Threshold Detection |

| - Rule-based Alerts |

+----------------------------+

|

v

+----------------------------+

| Notification System |

| - Slack, Email, SMS |

+----------------------------+

### ****장애 탐지 및 알림 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 알림 수단 | 주요 특징 |
| **헬스 체크 (Health Check)** | 주요 서비스의 상태 점검 | Prometheus | Grafana Dashboard | 실시간 상태 모니터링 |
| **임계치 설정 (Threshold Setting)** | CPU, Memory 사용량 임계치 설정 | Prometheus AlertManager | Slack, Email | 80% 이상 경고 발생 |
| **네트워크 장애 탐지** | 네트워크 단절 및 지연 탐지 | Grafana, Prometheus | Slack, SMS | 500ms 이상 Latency 감지 |
| **DB 연결 실패 탐지** | DB Connection 오류 탐지 | Prometheus SQL Exporter | Email, SMS | 5회 이상 재시도 실패 시 경고 |
| **컨테이너 장애 감지** | Pod Crash, Restart Loop 감지 | Prometheus K8s Adapter | Slack | 2회 이상 Crash 발생 시 알림 |
| **자동화 대응 프로세스** | 장애 발생 시 자동 복구 처리 | Kubernetes Auto Healing | N/A | Pod 자동 재시작 |

# ****API 관리 및 거버넌스 전략****

## ****API 설계 표준화****

### ****개요****

* 금융권 시스템의 **안정성**과 **확장성**을 보장하기 위해 API 설계는 표준화가 필요합니다.
* 주요 표준으로는 **REST API 설계 원칙**, **OpenAPI Specification**, **URI 설계 표준화**가 적용됩니다.
* 또한, **Idempotent Operation**, **Stateless Communication**, \*\*HATEOAS(Hypermedia As The Engine Of Application State)\*\*를 준수합니다.

### ****API 설계 표준화 구성도****

pgsql

복사편집

+----------------------------+

| API Gateway Layer |

| (Spring Cloud Gateway) |

+----------------------------+

|

v

+----------------------------+

| URI 표준화 규칙 |

| - /api/v1/accounts |

| - /api/v1/transactions |

+----------------------------+

|

v

+----------------------------+

| REST API 설계 원칙 |

| - GET: 조회 |

| - POST: 생성 |

| - PUT: 수정 |

| - DELETE: 삭제 |

+----------------------------+

|

v

+----------------------------+

| 데이터 표준화 규칙 |

| - JSON Format |

| - Camel Case 사용 |

+----------------------------+

### ****API 설계 표준화 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 표준화 규칙 | 적용 사례 |
| **URI 설계** | 명확하고 일관된 URI 경로 설정 | /api/v1/resource/{id} | /api/v1/accounts/12345 |
| **HTTP Method** | CRUD에 맞춘 HTTP 메서드 사용 | GET, POST, PUT, DELETE | 조회, 생성, 수정, 삭제 |
| **응답 형식** | JSON 표준화 및 Camel Case 사용 | JSON, Camel Case | { "accountId": "12345" } |
| **상태 코드 사용** | 표준 HTTP Status Code 적용 | 200, 201, 400, 404, 500 | 200 OK, 404 Not Found |
| **Idempotent Operation** | 중복 요청 시 데이터 무결성 보장 | PUT, DELETE | 중복 요청 시 동일 결과 반환 |
| **Stateless Communication** | 서버가 클라이언트 상태를 유지하지 않음 | Stateless API | 요청 시 매번 인증 필요 |
| **HATEOAS 지원** | 리소스 간 참조 정보 포함 | Self, Next, Prev 링크 | API 탐색 최적화 |
| **버전 관리** | 버전 명시를 통한 API 변화 관리 | /api/v1/ → /api/v2/ | 호환성 유지 |

## ****API 버전 관리 및 확장 전략****

### ****개요****

* 금융 시스템은 **업그레이드 및 변경 사항**이 빈번하게 발생하므로, 안정적 서비스 제공을 위해 **버전 관리**가 필수적입니다.
* **URI Versioning**, **Header Versioning**, **Content Negotiation**을 통해 확장성을 확보합니다.

### ****API 버전 관리 구성도****

pgsql

복사편집

+----------------------------+

| API Gateway Layer |

| (Spring Cloud Gateway) |

+----------------------------+

|

v

+----------------------------+

| URI Versioning |

| - /api/v1/accounts |

| - /api/v2/accounts |

+----------------------------+

|

v

+----------------------------+

| Header Versioning |

| - Accept: application/vnd.example.v1+json |

| - Accept: application/vnd.example.v2+json |

+----------------------------+

|

v

+----------------------------+

| Content Negotiation |

| - Accept: application/json |

| - Accept: application/xml |

+----------------------------+

### ****API 버전 관리 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 버전 관리 방식 | 설명 | 주요 특징 | 적용 사례 |
| **URI Versioning** | URI 경로에 버전을 명시하여 구분 | 직관적인 경로 관리 | /api/v1/accounts |
| **Header Versioning** | HTTP Header에 버전 정보를 포함 | URI 구조 변경 없음 | Accept: application/vnd.example.v1+json |
| **Content Negotiation** | 요청 시 Accept Header로 포맷 지정 | JSON, XML 형식 지원 | Accept: application/json |
| **Deprecated Policy** | 구버전 API의 단계적 종료 | 6개월 사전 공지 후 폐기 | /api/v1/ → /api/v2/ |
| **Backward Compatibility** | 기존 클라이언트와의 호환성 유지 | 구버전 요청 시 지원 | v1 API 유지 |
| **Multi-Version Deployment** | 신규 버전 배포 시 기존 버전 병행 운영 | Blue-Green Deployment | /api/v1/와 /api/v2/ 동시 지원 |

## ****API 보안 및 접근 제어****

### ****개요****

* 금융 시스템의 보안을 보장하기 위해 **OAuth2.0**, **JWT(Json Web Token)**, \*\*RBAC(Role-Based Access Control)\*\*를 적용합니다.
* API 요청 시 \*\*인증(Authentication)\*\*과 \*\*권한 검사(Authorization)\*\*를 수행하며, **TLS/SSL**로 데이터 전송을 보호합니다.

### ****API 보안 및 접근 제어 구성도****

pgsql

복사편집

+----------------------------+

| API Gateway |

| (Spring Cloud Gateway) |

+----------------------------+

|

v

+----------------------------+

| Authentication Layer |

| - OAuth2.0 |

| - JWT Verification |

+----------------------------+

|

v

+----------------------------+

| Authorization Layer |

| - Role-Based Access |

| - Resource Permission |

+----------------------------+

|

v

+----------------------------+

| Secure Data Transfer |

| - TLS/SSL Encryption |

+----------------------------+

### ****API 보안 및 접근 제어 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 보안 항목 | 설명 | 적용 기술 | 주요 특징 |
| **OAuth2.0 인증** | API 요청 시 사용자 인증 | Spring Security, Keycloak | Access Token 발급 |
| **JWT 인증** | Stateless 인증 및 트랜잭션 보호 | Spring Security JWT | 클라이언트-서버 무상태 유지 |
| **Role-Based Access Control** | 사용자 권한에 따른 접근 제어 | Spring Security, Keycloak | Role 설정에 따른 API 제한 |
| **TLS/SSL 암호화** | 데이터 전송 구간 보호 | Nginx, HAProxy | Man-in-the-middle 공격 방지 |
| **API Rate Limiting** | API 요청 남발 방지 | Kong, Spring Cloud Gateway | 초당 최대 요청 수 제한 |
| **IP Whitelisting** | 허용된 IP만 접근 허용 | API Gateway 설정 | 내부망 IP만 접근 허용 |
| **Data Encryption** | 민감 정보 암호화 저장 | AES-256, RSA | 금융 데이터 보호 |
| **Audit Logging** | API 호출 기록 추적 | ELK Stack | 접근 기록 보관 및 분석 |

# ****DevOps 및 CI/CD 전략****

## ****DevOps 설계 및 관리 전략****

### ****개요****

* 금융권 시스템의 **지속적 통합(Continuous Integration)**, **지속적 배포(Continuous Deployment)**, \*\*지속적 모니터링(Continuous Monitoring)\*\*을 지원하는 DevOps 구조를 설계합니다.
* 주요 도구로는 **Jenkins**, **GitLab CI**, **Docker**, **Kubernetes**, **Prometheus**, **Grafana**가 활용됩니다.
* 이를 통해 **빠른 배포 주기**, **안정성 확보**, **무중단 운영**이 가능하도록 최적화합니다.

### ****DevOps 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Source Control |

| - Git, GitLab, GitHub |

+----------------------------+

|

v

+----------------------------+

| CI Pipeline |

| - Jenkins, GitLab CI |

| - Code Analysis (SonarQube) |

+----------------------------+

|

v

+----------------------------+

| Build & Test |

| - Docker Build |

| - Unit Test, Integration Test |

+----------------------------+

|

v

+----------------------------+

| Artifact Repository |

| - Nexus, JFrog Artifactory |

+----------------------------+

|

v

+----------------------------+

| CD Pipeline |

| - ArgoCD, Spinnaker |

| - Canary Release, Blue-Green |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Cluster |

| - Auto Scaling, Self-Healing |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Alert |

| - Prometheus, Grafana |

| - AlertManager |

+----------------------------+

### ****DevOps 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **소스 관리** | 소스 코드 버전 관리 및 브랜치 관리 | Git, GitLab, GitHub | Branch Protection, Merge Request |
| **CI (지속적 통합)** | 빌드, 테스트, 코드 분석 자동화 | Jenkins, GitLab CI | SonarQube 코드 분석 |
| **빌드 관리** | 애플리케이션 빌드 및 이미지 생성 | Docker, Maven | Docker Image 생성 |
| **아티팩트 저장소** | 빌드된 결과물 보관 | Nexus, JFrog Artifactory | 버전 관리, 롤백 가능 |
| **CD (지속적 배포)** | 애플리케이션 배포 자동화 | ArgoCD, Spinnaker | Canary Release, Blue-Green |
| **컨테이너 오케스트레이션** | 서비스 배포 및 확장 관리 | Kubernetes | Auto Healing, Horizontal Scaling |
| **모니터링** | 실시간 서비스 상태 추적 | Prometheus, Grafana | 실시간 대시보드, 메트릭 수집 |
| **알림 설정** | 장애 발생 시 자동 알림 | Prometheus AlertManager | Slack, Email 통보 |

## ****CI/CD 파이프라인 구성****

### ****개요****

* 금융 시스템의 \*\*지속적 통합(Continuous Integration)\*\*과 \*\*지속적 배포(Continuous Deployment)\*\*를 위한 최적의 파이프라인을 설계합니다.
* 주요 단계는 **코드 빌드 → 테스트 → 배포 → 모니터링**으로 구성됩니다.
* **멀티 스테이지 파이프라인**을 적용하여 개발(Dev), 테스트(Test), 운영(Prod) 환경을 분리하고,  
  **Blue-Green Deployment**와 **Canary Release**를 통해 무중단 배포를 지원합니다.

### ****CI/CD 파이프라인 구성도****

pgsql

복사편집

+----------------------------+

| Code Commit |

| - Git Push |

+----------------------------+

|

v

+----------------------------+

| Build Stage |

| - Maven Build, Docker |

| - SonarQube Analysis |

+----------------------------+

|

v

+----------------------------+

| Test Stage |

| - Unit Test |

| - Integration Test |

| - Security Test (OWASP) |

+----------------------------+

|

v

+----------------------------+

| Artifact Repository |

| - Nexus, JFrog |

+----------------------------+

|

v

+----------------------------+

| Deploy Stage |

| - ArgoCD, Spinnaker |

| - Blue-Green, Canary |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Cluster |

| - Zero Downtime Deploy |

| - Auto Scaling Enabled |

+----------------------------+

### ****CI/CD 단계별 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 단계 | 설명 | 사용 기술 | 주요 특징 |
| **Code Commit** | 코드 변경 사항 커밋 및 브랜치 생성 | Git, GitLab | Merge Request, Review |
| **Build** | 코드 빌드 및 Docker 이미지 생성 | Maven, Docker | 멀티 스테이지 빌드 |
| **Test** | 유닛 테스트, 통합 테스트 실행 | JUnit, Postman | 자동화된 테스트 스크립트 |
| **Artifact Store** | 빌드 결과물 저장 | Nexus, JFrog | 버전 관리, 롤백 지원 |
| **Deploy** | 무중단 배포 실행 | ArgoCD, Spinnaker | Blue-Green, Canary Release |
| **Monitor** | 배포 후 서비스 상태 추적 | Prometheus, Grafana | 실시간 장애 탐지 |

## ****자동화 배포 및 무중단 운영 전략****

### ****개요****

* 금융권 시스템의 무중단 운영을 위해 **자동화 배포**, **Self-Healing**, **Auto Scaling**이 적용됩니다.
* 배포 전략으로 **Blue-Green Deployment**, **Canary Release**, **Rolling Update**를 사용하여 서비스 중단 없이 업데이트가 가능합니다.

### ****자동화 배포 및 무중단 운영 구성도****

pgsql

복사편집

+----------------------------+

| Jenkins Pipeline |

+----------------------------+

|

v

+----------------------------+

| Docker Image Push |

| - Nexus, JFrog |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Deployment |

| - Blue-Green Deployment |

| - Canary Release |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Cluster |

| - Auto Healing |

| - Zero Downtime Deploy |

+----------------------------+

### ****무중단 배포 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 전략 | 설명 | 사용 기술 | 주요 특징 |
| **Blue-Green Deployment** | 두 개의 환경을 번갈아가며 배포 | ArgoCD, Spinnaker | 무중단 배포, 빠른 롤백 |
| **Canary Release** | 일정 비율로 트래픽 분산 후 점진적 배포 | Istio, ArgoCD | 리스크 최소화 |
| **Rolling Update** | 한 번에 하나씩 업데이트하여 무중단 배포 | Kubernetes | 서비스 중단 없음 |
| **Self-Healing** | 장애 발생 시 자동으로 복구 | Kubernetes | Pod Auto Restart |
| **Auto Scaling** | 트래픽 증가 시 자동 확장 | Kubernetes HPA | 최대 처리량 보장 |

# ****장애 대응 및 복구 전략****

## ****장애 탐지 및 복구 설계****

### ****개요****

* 금융권 시스템의 **연속적 운영**과 **무중단 서비스**를 보장하기 위해 **자동화된 장애 탐지 및 복구 전략**이 필수적입니다.
* 주요 기술로는 **Prometheus**, **Grafana**, **AlertManager**, **Kubernetes Auto Healing**이 활용됩니다.
* 모든 장애 발생 시 **실시간 탐지**, **자동화된 복구**, **알림 전송**이 신속하게 이루어집니다.

### ****장애 탐지 및 복구 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Logging |

| - Prometheus |

| - Grafana |

| - ELK Stack |

+----------------------------+

|

v

+----------------------------+

| AlertManager |

| - 실시간 알림 전송 |

| - Slack, Email |

+----------------------------+

|

v

+----------------------------+

| Auto Healing System |

| - Kubernetes Controller |

| - Auto Restart, Scaling |

+----------------------------+

### ****장애 탐지 및 복구 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 대응 |
| **헬스 체크 (Health Check)** | 애플리케이션의 상태를 주기적으로 점검 | Prometheus | 5초 간격의 헬스 체크 |
| **오류 로그 탐지** | 서비스 오류 발생 시 실시간 로그 수집 | ELK Stack | Kibana 시각화 분석 |
| **임계치 알림 설정** | CPU, Memory 사용량 초과 시 알림 전송 | Prometheus AlertManager | Slack, Email 전송 |
| **네트워크 장애 탐지** | 네트워크 지연 및 단절 시 즉시 알림 | Grafana | Latency 500ms 초과 시 경고 |
| **DB 장애 탐지** | DB 연결 오류 또는 트랜잭션 실패 시 알림 | Prometheus SQL Exporter | 재시도 5회 실패 시 경고 |
| **Auto Healing** | 장애 발생 시 Kubernetes가 자동 복구 | Kubernetes HPA, Auto Healing | Pod Crash 시 자동 Restart |
| **Failover 처리** | 노드 장애 시 트래픽 자동 전환 | Kubernetes, Consul | Active-Active 구조 |
| **롤백 전략** | 배포 오류 발생 시 이전 버전으로 전환 | ArgoCD, Jenkins | 5초 내 롤백 처리 |

## ****DR(Disaster Recovery) 전략****

### ****개요****

* 금융 시스템의 \*\*재해 복구(Disaster Recovery, DR)\*\*는 **Multi-Region Replication**과 **Cross-Region Failover**를 통해 신속하게 복구됩니다.
* 주요 기술로는 **AWS Multi-AZ**, **GCP Multi-Region**, **Oracle Data Guard**가 활용됩니다.
* 장애 발생 시 \*\*RPO (Recovery Point Objective)\*\*와 \*\*RTO (Recovery Time Objective)\*\*를 최적화하여 데이터 손실을 최소화합니다.

### ****DR 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Primary Data Center |

| - AWS (Region A) |

+----------------------------+

|

v

+----------------------------+

| Multi-Region Replication |

| - S3 Cross-Region |

| - RDS Multi-AZ |

| - Kafka Mirror Maker |

+----------------------------+

|

v

+----------------------------+

| Disaster Recovery Site |

| - AWS (Region B) |

| - GCP Backup Location |

+----------------------------+

|

v

+----------------------------+

| Failover & Recovery |

| - DNS Switch |

| - Traffic Redirection |

+----------------------------+

### ****DR 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | RPO (Recovery Point Objective) | RTO (Recovery Time Objective) |
| **Multi-Region Replication** | 주요 데이터의 다중 리전 백업 | AWS S3, RDS Multi-AZ | 5분 | 10분 |
| **Cross-Region Failover** | 장애 시 다른 리전으로 트래픽 전환 | Route 53, GCP Multi-Region | 5분 | 15분 |
| **Database Mirroring** | DB 복제본을 다른 리전에 저장 | Oracle Data Guard, PostgreSQL Streaming | 1분 | 5분 |
| **Kafka Mirror Maker** | 메시지 스트림 복제 | Kafka Mirror Maker | 실시간 | 실시간 |
| **Cloud DNS Failover** | DNS 변경을 통한 빠른 전환 | AWS Route 53 | 1분 | 1분 |
| **Snapshot & Backup** | 정기적 데이터 스냅샷 생성 | AWS RDS Snapshot, GCP Backup | 5분 | 10분 |
| **Hot Standby** | 즉시 활성화 가능한 대기 서버 | Kubernetes Auto Scaling | 실시간 | 실시간 |

## ****복구 시나리오 및 테스트 계획****

### ****개요****

* 금융 시스템의 복구 시나리오는 다음과 같이 구분됩니다:
  + **단일 서비스 장애 복구**
  + **DB 장애 복구**
  + **네트워크 단절 복구**
  + **리전 장애 복구**

### ****복구 시나리오 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 시나리오 | 설명 | 주요 기술 | 복구 전략 | 예상 RTO |
| **단일 서비스 장애 복구** | Microservice 장애 발생 시 재시작 | Kubernetes, Prometheus | Auto Healing & Pod Restart | 1분 |
| **DB 장애 복구** | DB 인스턴스 장애 시 Failover 처리 | PostgreSQL Streaming, Oracle Data Guard | Hot Standby 전환 | 5분 |
| **네트워크 단절 복구** | 네트워크 연결 실패 시 복구 | Nginx, HAProxy | Auto Reroute 설정 | 2분 |
| **리전 장애 복구** | 전체 리전 장애 시 다른 리전으로 전환 | AWS Multi-AZ, GCP Multi-Region | Route 53 DNS Failover | 10분 |
| **스토리지 장애 복구** | S3 및 Object Storage 장애 발생 시 | AWS S3 Replication | Cross-Region Restore | 5분 |
| **메시지 브로커 장애 복구** | Kafka 브로커 다운 시 복구 | Kafka Mirror Maker | 메시지 재처리 | 실시간 |

# ****성능 및 부하 테스트 전략****

## ****부하 테스트 설계****

### ****개요****

* 금융권 시스템의 **고성능 트랜잭션 처리**를 보장하기 위해 \*\*부하 테스트(Load Test)\*\*를 설계합니다.
* 주요 목표는 **10,000 TPS**를 안정적으로 처리하고, **응답 시간 3초 이내**를 유지하는 것입니다.
* 테스트 환경은 **JMeter**, **Gatling**, **Locust**를 활용하여 자동화된 부하를 생성합니다.

### ****부하 테스트 아키텍처 구성도****

diff

복사편집

+----------------------------+

| Load Generator |

| - JMeter, Gatling, Locust|

+----------------------------+

|

v

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

| - Kong API Gateway |

+----------------------------+

|

v

+----------------------------+

| Microservices (MSA) |

| - Spring Boot Services |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Monitoring & Logging |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

### ****부하 테스트 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 테스트 항목 | 설명 | 목표 TPS | 응답 시간 목표 | 사용 도구 |
| **거래 처리 테스트** | 대량 금융 거래 요청의 동시 처리 | 10,000 TPS | 3초 이하 | JMeter, Gatling |
| **API Gateway 처리 테스트** | API Gateway를 통한 다중 요청 | 5,000 TPS | 2초 이하 | Spring Cloud Gateway |
| **DB 트랜잭션 테스트** | DB Read/Write 처리량 측정 | 3,000 TPS | 2초 이하 | PostgreSQL, Oracle DB |
| **캐시 적중률 테스트** | Redis의 조회 성능 측정 | 20,000 TPS | 1초 이하 | Redis, JMeter |
| **동시 사용자 테스트** | 동시에 접속하는 사용자 수 처리 | 10,000 명 | 3초 이하 | Locust, K6 |
| **파일 업로드 및 처리** | 대용량 파일 업로드 처리량 측정 | 2,000 TPS | 5초 이하 | S3, MinIO |
| **배치 처리 테스트** | 야간 정산 및 일괄 처리 성능 | 50,000 건/시간 | 1시간 이내 | Spring Batch, Quartz |

## ****스트레스 테스트 전략****

### ****개요****

* \*\*스트레스 테스트(Stress Test)\*\*는 시스템의 최대 처리 한계를 초과하는 부하를 가하여 안정성을 확인합니다.
* 주요 목표는 **동시 사용자 20,000명**을 처리하며, **트래픽 피크 타임**에 시스템이 안정적으로 대응하는지 검증합니다.
* 스트레스 테스트를 통해 **최대 처리량**, **성능 병목 구간**을 식별합니다.

### ****스트레스 테스트 아키텍처 구성도****

diff

복사편집

+----------------------------+

| Stress Generator |

| - Gatling, Locust |

+----------------------------+

|

v

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

| - Kong API Gateway |

+----------------------------+

|

v

+----------------------------+

| Microservices (MSA) |

| - Spring Boot Services |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Monitoring & Alerting |

| - Prometheus, Grafana |

+----------------------------+

### ****스트레스 테스트 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 테스트 항목 | 설명 | 최대 TPS | 최대 동시 사용자 | 사용 도구 |
| **트랜잭션 피크 테스트** | 최대 처리량을 초과하는 트랜잭션 발생 | 20,000 TPS | 20,000 명 | Gatling, Locust |
| **API Gateway 부하 테스트** | API Gateway에 대량 요청 전송 | 15,000 TPS | 15,000 명 | Spring Cloud Gateway |
| **DB 임계치 테스트** | DB Connection 수 최대치 도달 | 5,000 TPS | 10,000 커넥션 | PostgreSQL, Oracle DB |
| **Cache 임계치 테스트** | Redis의 메모리 한계 도달 테스트 | 50,000 TPS | 20,000 명 | Redis, Gatling |
| **네트워크 스트레스 테스트** | 대량 트래픽을 통한 네트워크 안정성 측정 | 10Gbps | 50,000 요청 | Nginx, HAProxy |

## ****내구성 테스트 및 스파이크 테스트****

### ****개요****

* \*\*내구성 테스트(Soak Test)\*\*는 장기간의 부하에도 시스템이 안정적인지 검증합니다.
* \*\*스파이크 테스트(Spike Test)\*\*는 짧은 시간 동안 **급격한 트래픽 증가**에도 안정적인지 확인합니다.
* 금융권 특성상 야간 배치 처리와 이벤트성 트랜잭션에서 발생할 수 있는 **대규모 스파이크**에 대응합니다.

### ****내구성 및 스파이크 테스트 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Load Generator |

| - JMeter, Gatling |

+----------------------------+

|

v

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| Microservices Cluster |

| - Kubernetes HPA |

| - Auto Scaling Enabled |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Alert |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

### ****내구성 및 스파이크 테스트 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 테스트 항목 | 설명 | 테스트 시간 | 목표 TPS | 사용 도구 |
| **내구성 테스트 (Soak Test)** | 24시간 이상 부하 지속 시 안정성 확인 | 24시간 | 5,000 TPS | JMeter, BlazeMeter |
| **야간 배치 처리 테스트** | 야간 정산 시 대규모 처리 테스트 | 8시간 | 50,000 건/시간 | Quartz, Spring Batch |
| **스파이크 테스트 (Spike Test)** | 갑작스러운 트래픽 증가 시 안정성 확인 | 30분 | 20,000 TPS | Gatling, K6 |
| **DB 내구성 테스트** | 장시간 DB 트랜잭션 처리 | 24시간 | 3,000 TPS | PostgreSQL, Oracle DB |
| **네트워크 스트레스 테스트** | 대량 트래픽 피크 테스트 | 1시간 | 10Gbps | Nginx, HAProxy |

# ****15. 운영 및 유지보수 전략****

## ****15.1 운영 관리 체계 설계****

### ****15.1.1 개요****

* 금융권 시스템의 안정적 운영을 위해 **모니터링**, **로그 수집**, **자동화된 장애 탐지**가 필수적입니다.
* 주요 관리 항목으로는 **시스템 헬스 체크**, **성능 모니터링**, **트랜잭션 추적**이 포함됩니다.
* 또한, **DevOps 운영 체계**를 통해 **자동화된 배포 및 모니터링**이 최적화됩니다.

### ****15.1.2 운영 관리 체계 구성도****

diff

복사편집

+----------------------------+

| Monitoring Layer |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

|

v

+----------------------------+

| Logging & Tracing |

| - Fluentd, ELK |

| - Jaeger, Zipkin |

+----------------------------+

|

v

+----------------------------+

| Application Management |

| - Kubernetes Dashboard |

| - Prometheus Alerting |

+----------------------------+

|

v

+----------------------------+

| Automated Scaling & Healing |

| - Kubernetes HPA |

| - Auto Healing |

+----------------------------+

### ****15.1.3 운영 관리 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 관리 항목 | 설명 | 사용 기술 | 주요 기능 |
| **실시간 모니터링** | 시스템 상태 실시간 체크 | Prometheus, Grafana | CPU, Memory, Disk 사용량 추적 |
| **로그 수집 및 분석** | 애플리케이션 로그 분석 | ELK Stack, Fluentd | 장애 탐지, 에러 분석 |
| **트랜잭션 추적** | 서비스 간 호출 상태 추적 | Jaeger, Zipkin | Latency 분석, Call Chain 추적 |
| **헬스 체크** | 서비스 상태 주기적 점검 | Prometheus Node Exporter | 5초 단위 체크 |
| **자동화된 배포 관리** | CI/CD 파이프라인 운영 | Jenkins, ArgoCD | 무중단 배포, 롤백 |
| **Self-Healing** | 장애 발생 시 자동 복구 | Kubernetes Auto Healing | Pod 자동 재시작 |
| **Auto Scaling** | 트래픽 증가 시 자동 확장 | Kubernetes HPA | 최대 처리량 보장 |
| **대시보드 시각화** | 운영 상태 시각적 분석 | Grafana | 실시간 메트릭 대시보드 |

## ****15.2 유지보수 계획 및 점검 체계****

### ****15.2.1 개요****

* 금융권 시스템의 **안정적 운영**과 **지속적 성능 최적화**를 위해 **정기 점검**과 **예방적 유지보수**가 필요합니다.
* **월간 점검**, **분기별 최적화**, **연간 리스크 분석**을 통해 시스템의 가용성을 극대화합니다.

### ****15.2.2 유지보수 계획 구성도****

diff

복사편집

+----------------------------+

| Preventive Maintenance |

| - 정기 점검 (Monthly) |

| - 장애 예방 최적화 |

+----------------------------+

|

v

+----------------------------+

| Optimization Plan |

| - 성능 개선 (Quarterly) |

| - DB 튜닝, 캐시 최적화 |

+----------------------------+

|

v

+----------------------------+

| Risk Assessment |

| - 리스크 분석 (Yearly) |

| - DR 시나리오 점검 |

+----------------------------+

|

v

+----------------------------+

| Documentation & Review |

| - 점검 보고서 작성 |

| - 운영 리포트 관리 |

+----------------------------+

### ****15.2.3 유지보수 계획 및 점검 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 항목 | 설명 | 주기 | 주요 활동 | 사용 기술 |
| **정기 점검** | 월간 시스템 점검 및 최적화 | 매월 1회 | CPU, Memory, Disk Health Check | Prometheus, Grafana |
| **성능 개선** | 분기별 성능 튜닝 및 최적화 | 매 분기 1회 | DB 인덱스 최적화, 캐시 리프레시 | PostgreSQL, Redis |
| **리스크 분석** | 연간 리스크 평가 및 대응 계획 | 매년 1회 | Disaster Recovery 시나리오 테스트 | AWS Multi-AZ, GCP Backup |
| **취약점 점검** | 보안 취약점 탐지 및 조치 | 매월 1회 | XSS, SQL Injection 탐지 | SonarQube, OWASP |
| **로그 분석 및 백업 점검** | 운영 로그 분석 및 백업 무결성 점검 | 매주 1회 | ELK Stack 분석, S3 Snapshot 확인 | ELK Stack, AWS S3 |
| **업그레이드 및 패치** | 보안 패치 및 라이브러리 업데이트 | 매월 1회 | Spring Boot, Kafka 최신 버전 반영 | Jenkins, ArgoCD |

## ****15.3 SLA(서비스 수준 협약) 관리 전략****

### ****15.3.1 개요****

* 금융 시스템의 안정적 운영을 보장하기 위해 \*\*SLA (Service Level Agreement)\*\*를 수립합니다.
* 주요 항목은 **가용성(Availability)**, **응답 시간(Response Time)**, **복구 시간(Recovery Time)**, \*\*보안 정책(Security Policy)\*\*입니다.
* SLA를 기반으로 **서비스 수준 목표**를 명확히 하고, **모니터링 시스템**을 통해 이를 준수합니다.

### ****15.3.2 SLA 관리 체계 구성도****

sql

복사편집

+----------------------------+

| Monitoring Layer |

| - Prometheus, Grafana |

+----------------------------+

|

v

+----------------------------+

| SLA Compliance Check |

| - Availability 99.9% |

| - Response Time 3초 이내 |

+----------------------------+

|

v

+----------------------------+

| Alert & Reporting |

| - SLA 위반 시 경고 전송 |

| - 실시간 보고서 생성 |

+----------------------------+

|

v

+----------------------------+

| Reporting & Escalation |

| - 월간 SLA 리포트 생성 |

| - 미달성 시 조치 계획 |

+----------------------------+

### ****15.3.3 SLA 관리 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 항목 | 설명 | 목표 수치 | 모니터링 도구 | 대응 계획 |
| **가용성 (Availability)** | 시스템의 정상 가동 시간 | 99.9% | Prometheus, Grafana | 장애 발생 시 5분 내 복구 |
| **응답 시간 (Response Time)** | API 요청에 대한 응답 시간 | 3초 이내 | Grafana, ELK Stack | 3초 초과 시 Alert 전송 |
| **복구 시간 (Recovery Time)** | 장애 발생 후 정상화 시간 | 15분 이내 | Prometheus AlertManager | DR Site 전환 |
| **보안 정책 (Security Policy)** | 보안 취약점 대응 및 방어 | 100% 대응 | SonarQube, OWASP ZAP | 미탐지 시 이메일 경고 |
| **데이터 보존 정책** | 거래 기록의 안전한 보관 | 7년 보관 | S3 Glacier, RDS | 분기별 백업 무결성 체크 |
| **로그 관리 정책** | 모든 로그의 저장 및 분석 | 1년 보관 | ELK Stack | 월간 로그 분석 보고서 |

# ****보안 및 규제 준수 전략****

## ****금융권 규제 준수****

### ****개요****

* 금융권 시스템은 **전자금융감독규정**, **개인정보보호법(Privacy Act)**, **ISMS-P(정보보호 관리체계)** 등의 규제를 준수해야 합니다.
* 주요 보안 항목은 **개인정보 보호**, **전자금융거래 안정성**, **로그 관리**가 포함됩니다.
* 또한, \*\*금융감독원(FSS)\*\*와 \*\*KISA(한국인터넷진흥원)\*\*의 가이드라인을 따라 보안 체계를 강화합니다.

### ****금융권 규제 준수 구성도****

diff

복사편집

+----------------------------+

| 금융감독원 (FSS) |

| - 전자금융거래 규정 |

| - 데이터 보호 가이드라인 |

+----------------------------+

|

v

+----------------------------+

| 정보보호 관리체계 |

| - ISMS-P |

| - PIMS |

+----------------------------+

|

v

+----------------------------+

| 개인정보 보호 관리 |

| - Privacy Act |

| - GDPR Compliance |

+----------------------------+

|

v

+----------------------------+

| 로그 및 트랜잭션 |

| - ELK Stack |

| - Fluentd, Logstash |

+----------------------------+

### ****금융권 규제 준수 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 규제 항목 | 설명 | 적용 기술 | 준수 항목 |
| **전자금융거래 안정성** | 금융 거래 시 무결성 및 가용성 확보 | Spring Security, OAuth2.0 | FSS 규정 제8조 |
| **개인정보 보호** | 고객 정보의 수집, 처리, 보관 시 보호 | AES-256, RSA Encryption | 개인정보보호법 제25조 |
| **정보보호 관리체계(ISMS-P)** | 금융 서비스의 정보 보호 및 관리 | Prometheus, ELK Stack | ISMS-P 인증 기준 |
| **GDPR 준수** | 유럽연합(EU) 지역 고객 정보 보호 | Keycloak, Spring Security | GDPR Article 32 |
| **접근 통제** | 권한 기반의 접근 제한 | RBAC(Role-Based Access Control) | 정보보호법 제29조 |
| **로그 관리** | 금융 거래 및 시스템 접속 로그 저장 | ELK Stack, Fluentd | FSS 가이드라인 |
| **데이터 보존 및 파기** | 고객 데이터의 안전한 보관 및 파기 | AWS S3 Glacier, Oracle Vault | 정보보호법 제27조 |
| **감사 대응** | 금융감독원의 요구에 따른 증적 제공 | Kibana Dashboard, ELK Stack | 금융감독원 요구사항 |

## ****데이터 보안 정책****

### ****개요****

* 금융 시스템의 데이터는 **암호화, 접근 제어, 무결성 보호**를 통해 보호됩니다.
* 데이터는 **전송 중(At Transit)**, **저장 중(At Rest)**, **처리 중(In Processing)** 단계별로 보호 전략이 적용됩니다.
* 주요 기술로는 **TLS/SSL**, **AES-256 Encryption**, **RSA 암호화**가 사용됩니다.

### ****데이터 보안 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Data Encryption Layer |

| - AES-256, RSA |

| - Transparent Data Encryption (TDE) |

+----------------------------+

|

v

+----------------------------+

| Transport Layer |

| - TLS/SSL |

| - Mutual TLS (mTLS) |

+----------------------------+

|

v

+----------------------------+

| Access Control Layer |

| - Role-Based Access |

| - Multi-Factor Auth |

+----------------------------+

### ****데이터 보안 정책 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 적용 기술 | 주요 특징 |
| **전송 중 암호화 (At Transit)** | 네트워크 전송 시 데이터 보호 | TLS 1.3, mTLS | 중간자 공격 방어 |
| **저장 중 암호화 (At Rest)** | 저장 시 민감 정보 암호화 | AES-256, RSA | 데이터 유출 방지 |
| **DB 암호화 (Database Encryption)** | DB 레벨의 암호화 적용 | TDE, Oracle Vault | 무결성 보장 |
| **API 보안 (API Security)** | API 호출 시 인증 및 보안 | OAuth2.0, JWT | 무단 접근 방지 |
| **키 관리 (Key Management)** | 암호화 키의 생성 및 관리 | AWS KMS, HashiCorp Vault | 키 유출 방지 |
| **접근 통제 (Access Control)** | 권한 기반의 접근 제어 | RBAC, Keycloak | 최소 권한 원칙 |
| **로그 관리 (Logging)** | 금융 거래 및 접근 로그 추적 | ELK Stack | 감사 대응 가능 |
| **데이터 파기 (Data Destruction)** | 만료된 데이터의 안전한 삭제 | S3 Glacier, AWS KMS | 데이터 복구 방지 |

## ****암호화 및 접근 제어****

### ****개요****

* 금융권 시스템의 **암호화**와 **접근 제어**는 **데이터 유출 방지**와 **불법 접근 차단**을 목표로 합니다.
* **AES-256**과 **RSA-2048**을 통해 데이터 암호화를 진행하고, **OAuth2.0**, **JWT**를 활용한 인증 체계를 구축합니다.
* \*\*Role-Based Access Control(RBAC)\*\*로 사용자 권한을 구분하여 민감한 데이터 접근을 제한합니다.

### ****암호화 및 접근 제어 구성도****

pgsql

복사편집

+----------------------------+

| API Gateway Layer |

| - OAuth2.0, JWT |

+----------------------------+

|

v

+----------------------------+

| Authentication Layer |

| - Keycloak |

| - Spring Security |

+----------------------------+

|

v

+----------------------------+

| Authorization Layer |

| - Role-Based Access |

| - Attribute-Based Access|

+----------------------------+

|

v

+----------------------------+

| Data Encryption Layer |

| - AES-256, RSA-2048 |

| - HSM (Hardware Security Module) |

+----------------------------+

### ****암호화 및 접근 제어 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 적용 기술 | 주요 특징 |
| **데이터 암호화** | 민감 데이터 암호화 저장 | AES-256, RSA-2048 | 금융 데이터 보호 |
| **API 인증** | API 요청 시 사용자 인증 | OAuth2.0, JWT | Access Token 발급 |
| **Role-Based Access Control** | 역할 기반 권한 제어 | Keycloak, Spring Security | 최소 권한 원칙 |
| **Multi-Factor Authentication** | 이중 인증으로 보안 강화 | Google Authenticator | 이중 보안 |
| **HSM (Hardware Security Module)** | 물리적 키 보호 | AWS CloudHSM | 보안 키 관리 최적화 |
| **IP Whitelisting** | 특정 IP만 접근 허용 | API Gateway 설정 | 불법 접근 차단 |

# ****데이터 분석 및 리포팅 전략****

## ****데이터 분석 체계 설계****

### ****개요****

* 금융 시스템의 **거래 내역**, **고객 행동 분석**, **실시간 트랜잭션 모니터링**을 통해 **데이터 기반 의사결정**을 지원합니다.
* 주요 기술로는 **Apache Kafka**, **Apache Spark**, **Elasticsearch**, **Prometheus**가 사용됩니다.
* 데이터 분석 체계는 **실시간 스트리밍 분석**과 **배치 분석**으로 구분되며, **OLAP 분석**을 통해 심층적인 통계 분석이 가능합니다.

### ****데이터 분석 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Data Sources |

| - Transaction DB |

| - Application Logs |

| - Kafka Streams |

+----------------------------+

|

v

+----------------------------+

| Data Ingestion |

| - Apache Kafka |

| - Logstash, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Data Processing |

| - Apache Spark |

| - Flink, Airflow |

+----------------------------+

|

v

+----------------------------+

| Data Analysis & Storage |

| - Elasticsearch |

| - Prometheus |

| - HDFS, S3 |

+----------------------------+

|

v

+----------------------------+

| Data Visualization |

| - Kibana, Grafana |

| - Tableau, Power BI |

+----------------------------+

### ****17.1.3 데이터 분석 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **실시간 데이터 분석** | 금융 거래 및 실시간 이벤트 분석 | Apache Kafka, Spark | 초당 10,000 TPS 분석 |
| **배치 데이터 분석** | 야간 정산 및 대량 데이터 처리 | Apache Spark, Airflow | 일간 1TB 이상 처리 |
| **스트림 처리** | 실시간 트랜잭션 흐름 분석 | Kafka Streams, Flink | Event-Driven Architecture |
| **로그 분석** | 금융 거래 로그 및 오류 탐지 | ELK Stack (Elasticsearch, Logstash, Kibana) | 실시간 에러 탐지 |
| **모니터링 데이터 분석** | 시스템 성능 및 리소스 사용 분석 | Prometheus, Grafana | TPS, 응답 시간, CPU 사용량 분석 |
| **OLAP 분석** | 대량 데이터의 다차원 분석 | Apache Druid, Presto | 실시간 대시보드 제공 |
| **시각화 도구** | 분석 결과 시각화 및 리포팅 | Grafana, Kibana, Tableau | 실시간 대시보드 및 리포트 생성 |

## ****리포팅 아키텍처 구성****

### ****개요****

* 금융권의 대량 거래 데이터를 **정형화된 리포트**로 생성하고, **실시간 조회**가 가능하도록 아키텍처를 설계합니다.
* **정기 리포팅**, **이벤트 리포팅**, **실시간 리포팅**으로 구분되며, **OZ Report**, **Crystal Report**, **Pentaho**를 사용합니다.

### ****리포팅 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Data Warehouse |

| - PostgreSQL, Oracle |

+----------------------------+

|

v

+----------------------------+

| ETL (Extract, Transform, Load) |

| - Talend, Pentaho, Airflow |

+----------------------------+

|

v

+----------------------------+

| Reporting Engine |

| - OZ Report |

| - Crystal Report |

| - Pentaho Reporting |

+----------------------------+

|

v

+----------------------------+

| Report Visualization |

| - Power BI, Tableau |

| - Grafana, Kibana |

+----------------------------+

### ****17.2.3 리포팅 전략 표****

| **리포트 유형** | **설명** | **생성 주기** | **사용 도구** | **주요 특징** |
| --- | --- | --- | --- | --- |
| **정기 리포트** | 일/주/월 단위 정기적 생성 | 일간, 주간, 월간 | OZ Report, Crystal Report | 금융 거래 현황, 계좌 내역 |
| **이벤트 리포트** | 특정 이벤트 발생 시 생성 | 실시간 | Pentaho, JasperReports | 실시간 송금 내역, 알림 로그 |
| **실시간 리포트** | 실시간 데이터 기반 대시보드 | 실시간 | Grafana, Kibana | 실시간 거래 모니터링 |
| **대량 거래 리포트** | 대규모 트랜잭션 처리 분석 | 월간 | Tableau, Power BI | 거래량, 이체 내역 분석 |
| **장애 리포트** | 시스템 장애 및 오류 분석 | 이벤트 발생 시 | ELK Stack, Prometheus | 장애 시나리오 추적 |
| **보안 감사 리포트** | 보안 정책 준수 및 접근 내역 | 월간 | OZ Report, Crystal Report | 감사 대응용 로그 보관 |

## ****실시간 대시보드 및 분석 보고서 생성****

### ****개요****

* 실시간 데이터 분석을 **대시보드**에 시각화하여 금융 거래 상태를 모니터링합니다.
* 주요 분석 항목은 **거래 처리 상태**, **계좌 내역 조회**, **시스템 리소스 사용량** 등입니다.
* **Grafana**, **Kibana**, **Power BI**를 통해 **대시보드 생성** 및 **자동화된 보고서 생성**을 지원합니다.

### ****실시간 대시보드 구성도****

diff

복사편집

+----------------------------+

| Data Analysis Layer |

| - Elasticsearch, Spark |

+----------------------------+

|

v

+----------------------------+

| Visualization Engine |

| - Grafana, Kibana |

| - Power BI, Tableau |

+----------------------------+

|

v

+----------------------------+

| Dashboard Presentation |

| - 실시간 거래 상태 |

| - 계좌 잔액 및 이체 내역 |

| - 시스템 자원 사용량 |

+----------------------------+

### ****17.3.3 대시보드 및 분석 보고서 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **실시간 거래 모니터링** | 실시간 트랜잭션 상태 추적 | Grafana, Kibana | TPS, 응답 시간 시각화 |
| **계좌 내역 조회 대시보드** | 고객 계좌 정보 및 이체 내역 조회 | Power BI, Tableau | 잔액, 거래 내역 분석 |
| **리소스 사용량 분석** | CPU, Memory, Disk 사용량 추적 | Grafana, Prometheus | 실시간 자원 모니터링 |
| **이벤트 모니터링** | 장애 및 경고 알림 시각화 | Kibana, ELK Stack | 오류 발생 시 경고 |
| **월간 리포트 생성** | 금융 거래 분석 및 리포트 생성 | Crystal Report, Pentaho | 자동화된 PDF 생성 |
| **보안 감사 대시보드** | 보안 로그 및 접근 기록 조회 | Kibana, Grafana | 접근 내역 시각화 |

# ****통합 테스트 및 검증 전략****

## ****테스트 계획 및 전략****

### ****개요****

* 금융권 시스템의 **안정성 확보**와 **무결성 보장**을 위해 **통합 테스트(Integration Test)** 및 **검증 전략**을 수립합니다.
* 테스트는 **기능 테스트**, **비기능 테스트**, **회귀 테스트**, **보안 테스트**로 구분됩니다.
* 주요 도구로는 **JUnit**, **Postman**, **Selenium**, **JMeter**, **OWASP ZAP**이 사용됩니다.

### ****테스트 아키텍처 구성도****

diff

복사편집

+----------------------------+

| Development Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Continuous Integration |

| - Jenkins, GitLab CI |

| - SonarQube, Jacoco |

+----------------------------+

|

v

+----------------------------+

| Test Automation |

| - JUnit, Postman, K6 |

| - Selenium, Cucumber |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Reporting |

| - Grafana, Prometheus |

| - Allure Report, ELK |

+----------------------------+

### ****테스트 계획 및 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 유형 | 설명 | 주요 도구 | 목표 및 주요 사항 |
| **기능 테스트** | 각 모듈의 요구 사항에 따른 기능 정상 작동 여부 확인 | JUnit, Postman | 모든 요구사항을 충족하는지 확인 |
| **통합 테스트** | 모듈 간의 상호작용과 데이터 전달 검증 | Postman, Newman | API 통신, 데이터 흐름 확인 |
| **회귀 테스트** | 기존 기능이 신기능 도입 시 정상 동작 유지 확인 | Selenium, Cucumber | 배포 시 기능 장애 방지 |
| **부하 테스트** | 대량 요청 시 성능과 안정성 측정 | JMeter, Gatling | 목표 TPS(10,000 TPS) 유지 |
| **스트레스 테스트** | 최대 처리 한계 상황에서 안정성 측정 | Locust, K6 | 최대 동시 사용자 20,000명 |
| **보안 테스트** | XSS, SQL Injection 등 보안 취약점 탐지 | OWASP ZAP, SonarQube | 보안 취약점 0건 목표 |
| **사용자 수용 테스트** | 실제 사용 환경에서 요구 사항 충족 여부 확인 | Selenium, Cypress | 실 사용자 시나리오 테스트 |
| **데이터 무결성 테스트** | 데이터의 정확성과 일관성 보장 | JUnit, DBUnit | 트랜잭션 무결성 보장 |

## ****18.2 통합 테스트 아키텍처 설계****

### ****18.2.1 개요****

* 금융 시스템의 **다중 모듈 통신**과 **MSA 구조**를 고려한 **통합 테스트**를 설계합니다.
* 모든 API와 서비스 간의 **연동 상태**, **데이터 전달의 일관성**, **비동기 메시지 처리**를 검증합니다.

### ****통합 테스트 아키텍처 구성도****

diff

복사편집

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| Microservices Layer |

| - Account Service |

| - Payment Service |

| - Transaction Service |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Asynchronous Message |

| - Apache Kafka |

| - RabbitMQ |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Logging |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

### ****통합 테스트 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 항목 | 설명 | 주요 도구 | 주요 검증 사항 |
| **API 연동 테스트** | 서비스 간 API 호출 및 응답 확인 | Postman, Newman | 200 OK 응답, 데이터 일관성 |
| **메시지 처리 테스트** | 비동기 메시지 전송 및 수신 검증 | Apache Kafka, RabbitMQ | 메시지 유실 및 중복 방지 |
| **DB 연동 테스트** | 서비스 간 DB 트랜잭션 확인 | DBUnit, JUnit | Commit/Rollback 일관성 |
| **캐시 일관성 테스트** | Redis Cache와 DB 간 데이터 동기화 | Redis CLI, Postman | Cache Miss 방지 |
| **Gateway 라우팅 테스트** | API Gateway의 라우팅 정확성 검증 | Spring Cloud Gateway | 올바른 서비스로 전달 여부 |
| **서비스 장애 대응 테스트** | 하나의 서비스 장애 시 다른 서비스 영향 측정 | K6, Locust | 장애 전파 방지 |
| **JWT 인증 테스트** | 각 서비스의 인증 및 권한 처리 확인 | Postman, OAuth2.0 | Access Token 유효성 확인 |
| **로드 밸런싱 테스트** | Gateway를 통한 트래픽 분산 확인 | Nginx, HAProxy | 트래픽 병목 방지 |

## ****자동화 테스트 및 보고 체계****

### ****개요****

* 금융 시스템의 테스트는 **자동화된 파이프라인**을 통해 관리되며, **CI/CD** 과정에서 통합됩니다.
* **Jenkins**, **GitLab CI**를 활용하여 **빌드 → 테스트 → 배포**의 전 과정을 자동화합니다.
* 테스트 결과는 **Allure Report**, **Grafana**를 통해 시각화되며, **Slack 알림**으로 전파됩니다.

### ****자동화 테스트 구성도****

sql

복사편집

+----------------------------+

| CI Pipeline |

| - Jenkins, GitLab CI |

+----------------------------+

|

v

+----------------------------+

| Test Automation |

| - JUnit, Selenium |

| - Postman, Cucumber |

+----------------------------+

|

v

+----------------------------+

| Test Reporting & Alert |

| - Allure Report |

| - Grafana, Prometheus |

+----------------------------+

|

v

+----------------------------+

| Notification System |

| - Slack, Email |

+----------------------------+

### ****자동화 테스트 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **CI/CD 통합 테스트** | Jenkins, GitLab CI와 연동하여 자동화 테스트 수행 | Jenkins, GitLab CI | 코드 커밋 시 테스트 실행 |
| **API 테스트 자동화** | API 요청 및 응답 테스트 자동화 | Postman, Newman | Regression Test 포함 |
| **UI 테스트 자동화** | 사용자 인터페이스 동작 검증 | Selenium, Cypress | 화면 전환, 버튼 동작 확인 |
| **부하 테스트 자동화** | 대량 요청 시 성능 및 안정성 측정 | Gatling, K6 | 초당 10,000 TPS 목표 |
| **보안 테스트 자동화** | OWASP 기준 보안 취약점 탐지 | OWASP ZAP, SonarQube | SQL Injection, XSS 탐지 |
| **자동화 보고서 생성** | 테스트 결과 리포트 생성 | Allure Report | 실패 로그 및 성공률 시각화 |
| **알림 시스템** | 테스트 실패 시 실시간 알림 | Slack, Email | 즉각적인 대응 가능 |

# ****최적화 및 성능 개선 전략****

## ****애플리케이션 최적화****

### ****개요****

* 금융권 시스템의 **고성능 처리**를 보장하기 위해 애플리케이션 레벨의 최적화가 필요합니다.
* 주요 최적화 항목은 **Spring Boot 최적화**, **JPA Query 최적화**, **Cache 적용**, **Connection Pool 튜닝**입니다.
* **멀티 스레드 처리**와 **비동기 통신**을 활용하여 성능 병목을 최소화합니다.

### ****애플리케이션 최적화 구성도****

sql

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Cache Layer |

| - Redis, Memcached |

+----------------------------+

|

v

+----------------------------+

| DB Connection Pool |

| - HikariCP, Tomcat Pool |

+----------------------------+

|

v

+----------------------------+

| Async Processing |

| - @Async, CompletableFuture |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Alert |

| - Prometheus, Grafana |

+----------------------------+

### ****19.1.3 애플리케이션 최적화 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **JPA Query 최적화** | JPQL 및 Native Query 최적화 | Spring Data JPA | N+1 문제 방지, Fetch Join 사용 |
| **Connection Pool 튜닝** | DB Connection Pool 최적화 | HikariCP, Tomcat JDBC | 최대 커넥션 수 최적화 |
| **Cache 적용** | 자주 조회되는 데이터 캐싱 | Redis, Memcached | 조회 성능 50% 개선 |
| **멀티 스레드 처리** | 비동기 요청 시 멀티 스레드 처리 | @Async, CompletableFuture | 트랜잭션 동시 처리 최적화 |
| **비동기 통신 처리** | 메시지 처리의 비동기화 | Kafka, RabbitMQ | 비동기 이벤트 처리 최적화 |
| **Bulk Insert 최적화** | 대량 데이터 입력 최적화 | JPA Batch Processing | DB Commit 횟수 최소화 |
| **JSON Parsing 최적화** | API 응답 속도 개선 | Jackson, Gson | 불필요한 필드 제거 |
| **HTTP Connection 관리** | Keep-Alive 설정으로 성능 최적화 | OkHttp, Apache HttpClient | 연결 유지 최적화 |

## ****데이터베이스 성능 개선****

### ****개요****

* 금융 시스템의 **대량 데이터 처리**와 **실시간 트랜잭션 처리**를 최적화하기 위해 DB 레벨의 튜닝이 필요합니다.
* 주요 최적화 항목으로 **인덱스 최적화**, **쿼리 튜닝**, **파티셔닝**, **Replication**이 있습니다.

### ****데이터베이스 최적화 구성도****

pgsql

복사편집

+----------------------------+

| Database Cluster |

| - PostgreSQL, Oracle DB |

+----------------------------+

|

v

+----------------------------+

| Index Optimization |

| - B-Tree Index |

| - Hash Index |

+----------------------------+

|

v

+----------------------------+

| Query Optimization |

| - Explain Plan |

| - Query Caching |

+----------------------------+

|

v

+----------------------------+

| Partitioning Strategy |

| - Range, List Partition |

| - Sharding |

+----------------------------+

|

v

+----------------------------+

| Replication & Backup |

| - Master-Slave, Multi-AZ|

+----------------------------+

### ****데이터베이스 최적화 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **인덱스 최적화** | 검색 속도를 높이기 위한 인덱스 설정 | B-Tree, Hash Index | 조회 성능 30% 이상 향상 |
| **쿼리 튜닝** | 복잡한 쿼리의 성능 최적화 | EXPLAIN PLAN, Query Hint | Full Scan 방지 |
| **파티셔닝 전략** | 대용량 테이블의 성능 최적화 | Range, List Partition | 검색 시간 단축 |
| **Sharding 처리** | 수평 분할을 통해 부하 분산 | PostgreSQL Sharding, Oracle Sharding | 병목 현상 감소 |
| **Replication 설정** | 다중 리전 및 가용성 확보 | Master-Slave Replication | 장애 시 즉시 Failover |
| **Connection Pool 튜닝** | 동시 접속 수 최적화 | HikariCP, Tomcat Pool | Connection Timeout 최소화 |
| **쿼리 캐싱** | 반복 조회 시 캐시 사용 | Redis, Memcached | 응답 속도 개선 |
| **실시간 백업 및 복구** | 데이터 손실 방지 및 복구 최적화 | PITR, WAL Shipping | 1초 미만 복구 시간 |

## ****네트워크 최적화****

### ****개요****

* 금융 시스템의 **고성능 트랜잭션 처리**를 위해 네트워크 레벨에서 최적화를 진행합니다.
* **로드 밸런싱**, **캐싱 전략**, **네트워크 압축**을 통해 성능을 극대화합니다.

### ****네트워크 최적화 구성도****

diff

복사편집

+----------------------------+

| Load Balancer Layer |

| - Nginx, HAProxy |

+----------------------------+

|

v

+----------------------------+

| API Gateway Layer |

| - Spring Cloud Gateway |

| - Kong API Gateway |

+----------------------------+

|

v

+----------------------------+

| Distributed Cache |

| - Redis, Memcached |

+----------------------------+

|

v

+----------------------------+

| Network Compression |

| - Gzip, Brotli |

+----------------------------+

### ****네트워크 최적화 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **로드 밸런싱** | 트래픽을 여러 서버로 분산 처리 | Nginx, HAProxy | 병목 현상 방지 |
| **API Gateway 최적화** | API 요청의 빠른 전달과 처리 | Spring Cloud Gateway | 요청 시간 최소화 |
| **네트워크 압축** | 데이터 전송 시 압축 처리 | Gzip, Brotli | 대역폭 30% 감소 |
| **HTTP Keep-Alive 설정** | 지속 연결로 통신 최적화 | Nginx, Apache HTTP | 연결 시간 단축 |
| **캐싱 전략** | 자주 조회되는 응답 캐싱 | Redis, Memcached | 응답 시간 50% 단축 |
| **CDN(Content Delivery Network)** | 정적 자원 글로벌 분산 | CloudFront, Akamai | 지연 시간 최소화 |
| **DNS 최적화** | 빠른 도메인 해석 | AWS Route 53 | 응답 시간 감소 |
| **네트워크 모니터링** | 트래픽 흐름 및 상태 분석 | Prometheus, Grafana | 장애 탐지 및 조기 대응 |

# ****시스템 확장성 및 가용성 설계****

## ****확장성 아키텍처 설계****

### ****개요****

* 금융권 시스템의 **대량 트랜잭션 처리**와 **동시 사용자 증가**에 유연하게 대응하기 위해 **확장성 아키텍처**가 필요합니다.
* \*\*수평 확장(Scale-Out)\*\*과 \*\*수직 확장(Scale-Up)\*\*을 적절하게 조합하여 **탄력적 확장**을 실현합니다.
* 주요 기술로는 **Kubernetes HPA (Horizontal Pod Autoscaler)**, **Cluster Federation**, **Database Sharding**이 활용됩니다.

### ****확장성 아키텍처 구성도****

diff

복사편집

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| Load Balancer |

| - Nginx, HAProxy |

+----------------------------+

|

v

+----------------------------+

| Microservices |

| - Spring Boot Services |

| - Kubernetes Pods |

+----------------------------+

| | |

v v v

+--------------------+ +--------------------+ +--------------------+

| PostgreSQL | | Oracle DB | | Redis Cache |

| Sharding | | Multi-AZ | | Clustering |

+--------------------+ +--------------------+ +--------------------+

|

v

+----------------------------+

| Monitoring Layer |

| - Prometheus, Grafana |

+----------------------------+

### ****확장성 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **수평 확장 (Scale-Out)** | 서버 인스턴스를 추가하여 성능 개선 | Kubernetes HPA, EKS | 무중단 확장 지원 |
| **수직 확장 (Scale-Up)** | 서버 리소스(CPU, Memory) 증설 | AWS EC2, GCP Compute | 단일 인스턴스 성능 최적화 |
| **API Gateway 확장** | 트래픽 증가에 따른 Gateway 확장 | Spring Cloud Gateway, Kong | 부하 분산 최적화 |
| **Database Sharding** | 대용량 DB를 수평 분할하여 처리 | PostgreSQL Sharding | Read/Write 병목 최소화 |
| **캐싱 클러스터링** | 캐시 서버 다중 구성 | Redis Cluster, Hazelcast | 조회 성능 2배 이상 개선 |
| **Messaging Queue 확장** | 비동기 메시지 처리 확장 | Kafka, RabbitMQ | 메시지 처리 속도 최적화 |
| **멀티 리전 배포** | 글로벌 서비스 확장을 위한 다중 리전 | AWS Multi-Region, GCP Multi-Region | 장애 시 지역 간 전환 |
| **모니터링 최적화** | 실시간 상태 추적 및 확장 알림 | Prometheus, Grafana | 트래픽 증가 시 Autoscale Trigger |

## ****가용성 최적화 전략****

### ****개요****

* 금융권 시스템의 **무중단 서비스**를 보장하기 위해 **고가용성(High Availability)** 아키텍처를 설계합니다.
* 주요 기술로는 **Active-Active Clustering**, **Multi-AZ Deployment**, **Database Replication**이 적용됩니다.
* **Auto Healing**과 **Failover**를 통해 장애 발생 시 빠르게 복구합니다.

### ****가용성 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Load Balancer |

| - Nginx, HAProxy |

+----------------------------+

|

v

+----------------------------+

| Active-Active Clustering|

| - Multi-AZ Deployment |

| - Auto Healing |

+----------------------------+

|

v

+----------------------------+

| Database Replication |

| - PostgreSQL Streaming |

| - Oracle Data Guard |

+----------------------------+

|

v

+----------------------------+

| Backup & Recovery Layer |

| - AWS S3, GCP Cloud SQL |

+----------------------------+

### ****가용성 전략 표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 항목 |  | 설명 | 사용 기술 | 주요 특징 |
| **Active-Active Clustering** |  | 두 개 이상의 서버가 동시에 활성 상태로 서비스 제공 | Kubernetes Multi-AZ | 무중단 서비스 |
| **Multi-AZ Deployment** |  | 다른 가용성 영역(AZ)에 중복 배치 | AWS Multi-AZ, GCP Multi-Region | 장애 시 자동 전환 |
| **Database Replication** |  | 실시간 DB 복제 및 동기화 | PostgreSQL Streaming, Oracle Data Guard | 데이터 유실 방지 |
| **Auto Healing** |  | 장애 발생 시 자동 복구 | Kubernetes Auto Healing | 1분 이내 복구 |
| **Failover 처리** |  | 장애 시 자동으로 트래픽 전환 | Consul, Kubernetes | 서비스 중단 최소화 |
| **Backup & Recovery** |  | 정기적인 데이터 백업 및 복구 테스트 | AWS S3, GCP Cloud SQL | 데이터 복구 시간 최적화 |
| **Load Balancing 최적화** |  | 트래픽의 균등 분산 | Nginx, HAProxy | 서버 과부하 방지 |
| **모니터링 및 경고 설정** |  | 장애 발생 시 실시간 경고 전송 | Prometheus, Grafana | 조기 대응 가능 |

## ****무중단 확장 및 배포 전략****

### ****개요****

* 금융권 시스템의 **서비스 중단 없는 확장**을 보장하기 위해 **Blue-Green Deployment**, **Canary Release**, **Rolling Update** 전략을 적용합니다.
* **Kubernetes 클러스터**를 통해 Zero-Downtime 배포를 실현합니다.

### ****무중단 배포 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| CI/CD Pipeline |

| - Jenkins, GitLab CI |

+----------------------------+

|

v

+----------------------------+

| Docker Image Repository |

| - Nexus, JFrog |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Cluster |

| - Blue-Green Deployment |

| - Canary Release |

| - Rolling Update |

+----------------------------+

|

v

+----------------------------+

| Load Balancer & Gateway |

| - Nginx, Spring Gateway |

+----------------------------+

### ****무중단 배포 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 전략 | 설명 | 사용 기술 | 주요 특징 |
| **Blue-Green Deployment** | 두 개의 환경(Blue/Green)에서 번갈아 배포 | Kubernetes, ArgoCD | Zero Downtime, 빠른 롤백 |
| **Canary Release** | 트래픽 일부만 새로운 버전으로 전환 | Istio, ArgoCD | 리스크 최소화 |
| **Rolling Update** | 순차적 업데이트를 통한 무중단 배포 | Kubernetes, Helm | 실시간 배포 반영 |
| **Self-Healing Deployment** | 배포 중 장애 발생 시 자동 복구 | Kubernetes Auto Healing | 1분 이내 복구 |
| **Traffic Shaping** | 특정 트래픽만 새로운 버전으로 유도 | Nginx, HAProxy | 특정 사용자 테스트 가능 |
| **Shadow Deployment** | 실제 사용자 요청 없이 새 버전 테스트 | Istio, Linkerd | 서비스 영향 없이 검증 |

# ****금융 시스템 연계 및 인터페이스 전략****

## ****금융 API 연계 아키텍처****

### ****개요****

* 금융권 시스템은 외부 금융기관 및 내부 시스템 간의 **API 연계**가 필수적입니다.
* **REST API**, **SOAP API**, **gRPC**를 통한 상호 연동이 가능하며, **OAuth2.0**과 **JWT**를 통해 보안을 강화합니다.
* 주요 연계 시스템은 **금융결제원**, **카드사**, **은행 간 계좌이체**, **외환 송금** 등이 포함됩니다.

### ****금융 API 연계 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| External Systems |

| - 금융결제원 (KFTC) |

| - 카드사 (VISA, Master) |

| - 외환은행 (SWIFT) |

+----------------------------+

|

v

+----------------------------+

| API Gateway Layer |

| - Spring Cloud Gateway |

| - Kong API Gateway |

+----------------------------+

|

v

+----------------------------+

| Security Layer |

| - OAuth2.0, JWT |

| - Mutual TLS (mTLS) |

+----------------------------+

|

v

+----------------------------+

| Integration Layer |

| - REST, SOAP, gRPC |

| - Message Queue (Kafka) |

+----------------------------+

|

v

+----------------------------+

| Application Layer |

| - Account Service |

| - Payment Service |

+----------------------------+

### ****금융 API 연계 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **REST API 연계** | 표준 HTTP 기반의 금융 API 연동 | Spring Boot, OpenAPI | 금융결제원, 카드사 연계 |
| **SOAP API 연계** | 레거시 금융 시스템과의 통신 | Apache CXF, Axis2 | 은행 간 전산망 연결 |
| **gRPC 통신** | 실시간 데이터 전송 최적화 | gRPC, Protocol Buffers | 외환 송금, 실시간 거래 |
| **OAuth2.0 인증** | API 요청 시 인증 처리 | Spring Security, Keycloak | Access Token 기반 인증 |
| **JWT 토큰 인증** | 무상태 인증 및 트랜잭션 보장 | JWT (Json Web Token) | 세션 관리 불필요 |
| **Mutual TLS(mTLS)** | 상호 인증 기반 보안 통신 | Nginx, HAProxy | 금융기관 간 통신 보안 |
| **메시지 큐 연계** | 비동기 트랜잭션 처리 | Kafka, RabbitMQ | 이벤트 기반 연동 |
| **API Gateway 라우팅** | 다양한 금융기관 API 라우팅 | Kong, Spring Cloud Gateway | 경로 기반 라우팅 |

## ****메시징 및 이벤트 처리 전략****

### ****개요****

* 금융 시스템의 **대량 트랜잭션 처리**와 **비동기 통신**을 최적화하기 위해 **메시징 아키텍처**를 설계합니다.
* **Apache Kafka**, **RabbitMQ**, **AWS SQS**를 활용하여 **이벤트 기반 아키텍처**를 구성합니다.
* 실시간 결제, 이체 요청, 환전 시 이벤트 기반 처리가 가능하도록 설계합니다.

### ****메시징 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Account Service |

| - Payment Service |

+----------------------------+

|

v

+----------------------------+

| Messaging Queue Layer |

| - Apache Kafka |

| - RabbitMQ |

| - AWS SQS |

+----------------------------+

|

v

+----------------------------+

| Stream Processing Layer |

| - Kafka Streams |

| - Spark Streaming |

+----------------------------+

|

v

+----------------------------+

| Data Storage Layer |

| - PostgreSQL, Redis |

| - Elasticsearch |

+----------------------------+

### ****메시징 및 이벤트 처리 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **메시지 브로커** | 비동기 메시지 전송 및 수신 | Kafka, RabbitMQ | 송금 요청, 거래 알림 |
| **이벤트 스트림 처리** | 실시간 이벤트 처리 및 분석 | Kafka Streams, Flink | 초당 10,000 TPS 처리 |
| **지연 큐 처리** | 특정 시간 후 메시지 전송 | AWS SQS Delay Queue | 예약 거래 처리 |
| **Pub/Sub 모델** | 비동기 이벤트 발행 및 구독 | Kafka, Redis Streams | 이벤트 트리거 관리 |
| **트랜잭션 로그 처리** | 금융 트랜잭션 변경사항 실시간 처리 | CDC (Change Data Capture) | 무결성 보장 |
| **실시간 알림 전송** | 결제 완료, 송금 처리 알림 | Kafka, RabbitMQ | SMS, Push Notification |
| **Dead Letter Queue** | 처리 실패 메시지 재처리 | Kafka DLQ, RabbitMQ DLX | 장애 발생 시 복구 가능 |

## ****인터페이스 표준화 및 확장성****

### ****개요****

* 금융 시스템의 **인터페이스 표준화**를 통해 **확장성**과 **유지보수성**을 최적화합니다.
* **Open API 표준**과 **Swagger Documentation**을 통해 개발자 접근성을 높이고, **GraphQL**을 통한 확장성을 보장합니다.

### ****인터페이스 표준화 구성도****

sql

복사편집

+----------------------------+

| API Gateway |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| API Documentation |

| - OpenAPI, Swagger UI |

| - GraphQL Playground |

+----------------------------+

|

v

+----------------------------+

| Service Registration |

| - Eureka, Consul |

| - Service Discovery |

+----------------------------+

|

v

+----------------------------+

| API Versioning & Schema |

| - URI Versioning |

| - Header Versioning |

+----------------------------+

### ****인터페이스 표준화 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **OpenAPI 표준화** | API 명세서 작성 및 공유 | Swagger, OpenAPI 3.0 | 문서 자동 생성 |
| **GraphQL 도입** | 데이터 질의 최적화 | GraphQL, Apollo | 필요 데이터만 조회 |
| **URI Versioning** | API 변경 시 버전 관리 | Spring Cloud Gateway | /api/v1, /api/v2 |
| **Header Versioning** | HTTP Header 기반 API 버전 관리 | Spring Boot | Accept Header 사용 |
| **Service Discovery** | 서비스 동적 등록 및 탐색 | Eureka, Consul | MSA 간 동적 연결 |
| **API Schema Validation** | 요청 스키마 검증 | JSON Schema Validator | 데이터 유효성 보장 |
| **로깅 및 트래킹** | API 호출 로그 관리 | ELK Stack, Prometheus | 실시간 모니터링 가능 |

# ****보안 및 인증 아키텍처 설계****

## ****22.1 인증 및 인가 전략****

### ****22.1.1 개요****

* 금융권 시스템의 **접근 통제**와 **사용자 인증**을 보장하기 위해 **OAuth2.0**, **JWT(Json Web Token)**, \*\*RBAC(Role-Based Access Control)\*\*가 사용됩니다.
* \*\*SSO(Single Sign-On)\*\*를 통해 다수의 시스템에 한 번의 인증으로 접근이 가능하도록 설계합니다.
* 외부 금융기관 연계를 위한 \*\*OpenID Connect (OIDC)\*\*와 \*\*Mutual TLS(mTLS)\*\*를 적용합니다.

### ****22.1.2 인증 및 인가 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| API Gateway Layer |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| Authentication Server |

| - Keycloak, OAuth2.0 |

| - JWT Token Issuer |

+----------------------------+

|

v

+----------------------------+

| Authorization Layer |

| - RBAC, ABAC |

| - Attribute-based Access|

+----------------------------+

|

v

+----------------------------+

| Security Enforcement |

| - Spring Security |

| - Mutual TLS(mTLS) |

+----------------------------+

|

v

+----------------------------+

| External Systems |

| - Open Banking API |

| - SWIFT Network |

+----------------------------+

### ****22.1.3 인증 및 인가 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **OAuth2.0 인증** | 금융 시스템의 API 인증 처리 | Spring Security, Keycloak | Access Token 발급 및 관리 |
| **JWT 토큰 발급** | 무상태 인증으로 트랜잭션 보호 | JWT(Json Web Token) | 세션 없이 인증 유지 |
| **Role-Based Access Control (RBAC)** | 역할 기반 접근 제어 | Spring Security, Keycloak | 최소 권한 원칙 적용 |
| **Attribute-Based Access Control (ABAC)** | 속성 기반 접근 제어 | Keycloak, Spring Security | 사용자 속성 기반 통제 |
| **Single Sign-On (SSO)** | 다수 시스템 간 단일 로그인 | Keycloak, OpenID Connect | 중복 인증 최소화 |
| **Mutual TLS (mTLS)** | 금융기관 간 상호 인증 | Nginx, HAProxy | 강력한 암호화 통신 |
| **OpenID Connect (OIDC)** | 외부 인증 시스템 연동 | Keycloak, OpenID Provider | OAuth2.0 확장 인증 |

## ****데이터 암호화 및 접근 제어****

### ****개요****

* 금융권 시스템의 **민감 정보 보호**를 위해 **전송 중(At Transit)**, **저장 중(At Rest)**, **처리 중(In Processing)** 단계에서 암호화를 적용합니다.
* **AES-256**, **RSA-2048**를 활용하여 데이터 유출을 방지하며, \*\*HSM (Hardware Security Module)\*\*을 통해 키를 안전하게 관리합니다.

### ****데이터 암호화 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Encryption Layer |

| - AES-256, RSA-2048 |

| - Transparent Data Encryption |

+----------------------------+

|

v

+----------------------------+

| Key Management Layer |

| - AWS KMS, Vault |

| - HSM (CloudHSM) |

+----------------------------+

|

v

+----------------------------+

| Data Storage Layer |

| - PostgreSQL, Oracle DB |

| - Redis, S3 Glacier |

+----------------------------+

### ****데이터 암호화 및 접근 제어 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **전송 중 암호화 (At Transit)** | 네트워크 전송 시 데이터 보호 | TLS 1.3, mTLS | 중간자 공격 방어 |
| **저장 중 암호화 (At Rest)** | 저장된 데이터의 안전한 보호 | AES-256, RSA-2048 | 데이터 유출 방지 |
| **처리 중 암호화 (In Processing)** | 데이터 처리 시 실시간 보호 | Homomorphic Encryption | 실시간 연산 보호 |
| **키 관리 (Key Management)** | 암호화 키 생성 및 보관 | AWS KMS, Vault | 키 복구 및 회수 관리 |
| **하드웨어 보안 모듈 (HSM)** | 키의 물리적 보호 | AWS CloudHSM | FIPS 140-2 Level 3 준수 |
| **DB 암호화** | 데이터베이스 레벨 암호화 | TDE (Transparent Data Encryption) | 투명한 데이터 보호 |
| **API 보안** | API 요청 시 데이터 보호 | OAuth2.0, JWT | 무단 접근 방지 |
| **접근 통제 (Access Control)** | 민감 정보 접근 권한 제한 | Spring Security, Keycloak | RBAC, ABAC 적용 |

## ****네트워크 보안 및 침입 탐지****

### ****개요****

* 금융권 시스템의 **네트워크 보안 강화**와 **침입 탐지**를 위해 **WAF(Web Application Firewall)**, **IDS/IPS**를 적용합니다.
* **DDoS 방어**, **SQL Injection 차단**, **XSS 공격 방지**를 최적화하며, **Cloud Firewall**을 통해 접근을 제어합니다.

### ****네트워크 보안 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Web Application |

| - Spring Boot, React |

+----------------------------+

|

v

+----------------------------+

| Web Application Firewall|

| - AWS WAF, CloudFlare |

+----------------------------+

|

v

+----------------------------+

| Intrusion Detection & Prevention |

| - Snort, Suricata |

+----------------------------+

|

v

+----------------------------+

| Cloud Firewall & VPC |

| - AWS Security Group |

| - VPC Subnet Isolation |

+----------------------------+

|

v

+----------------------------+

| Monitoring Layer |

| - Prometheus, Grafana |

+----------------------------+

### ****네트워크 보안 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **WAF (Web Application Firewall)** | 웹 공격 방어 및 필터링 | AWS WAF, CloudFlare | SQL Injection, XSS 방지 |
| **IDS/IPS (Intrusion Detection/Prevention)** | 침입 탐지 및 방지 | Snort, Suricata | 실시간 공격 탐지 |
| **DDoS 방어** | 대량 트래픽 공격 방어 | AWS Shield, CloudFlare | 100Gbps 이상 방어 |
| **Cloud Firewall** | 네트워크 접근 제어 | AWS Security Group | IP 제한 및 포트 필터링 |
| **VPC Isolation** | 네트워크 격리 및 보호 | AWS VPC, Subnet | 내부 트래픽 분리 |
| **VPN Gateway** | 외부 접속 시 보안 터널링 | OpenVPN, AWS VPN | 보안 채널로 접근 |
| **SSL/TLS 통신** | 전송 구간 암호화 | TLS 1.3 | 데이터 유출 방지 |

# ****감사 및 로그 관리 전략****

## ****감사 로그 설계****

### ****개요****

* 금융 시스템의 **모든 트랜잭션, 사용자 활동, 보안 이벤트**를 추적하기 위해 \*\*감사 로그(Audit Log)\*\*를 설계합니다.
* **거래 이력**, **접속 기록**, **권한 변경** 등 주요 이벤트는 **중앙 집중식 로그 수집**으로 관리되며,  
  \*\*ELK Stack (Elasticsearch, Logstash, Kibana)\*\*를 활용하여 실시간 분석이 가능합니다.
* 금융감독원의 **전자금융거래 규정** 및 **ISMS-P 인증 기준**을 준수합니다.

### ****23.1.2 감사 로그 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Log Aggregation |

| - Fluentd, Logstash |

| - Filebeat, FluentBit |

+----------------------------+

|

v

+----------------------------+

| Centralized Storage |

| - Elasticsearch |

| - AWS S3, HDFS |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Analysis |

| - Kibana, Grafana |

| - Prometheus Alerting |

+----------------------------+

|

v

+----------------------------+

| Long-term Archive |

| - AWS Glacier, S3 |

| - Oracle Vault |

+----------------------------+

### ****23.1.3 감사 로그 설계 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **트랜잭션 로그** | 금융 거래 내역 추적 및 분석 | Fluentd, Logstash | 실시간 처리, 분석 최적화 |
| **접속 기록** | 사용자 로그인 및 접속 IP 기록 | ELK Stack, FluentBit | IP 주소, 세션 추적 |
| **권한 변경 로그** | 사용자 권한 변경 이력 관리 | Spring Security Audit | RBAC 변경 사항 기록 |
| **에러 및 예외 로그** | 애플리케이션 예외 발생 시 기록 | Logback, ELK Stack | 실시간 경고 전송 |
| **보안 이벤트 로그** | 로그인 실패, 비정상 접근 추적 | Kibana, Prometheus | OWASP Top 10 탐지 |
| **DB 접근 로그** | 데이터베이스 접근 이력 추적 | PostgreSQL Audit, Oracle Vault | SQL Injection 방지 |
| **파일 접근 로그** | 중요 파일의 생성, 수정, 삭제 기록 | AWS S3 Access Logs | 무단 접근 방지 |
| **장기 보관 로그** | 규제 준수 및 감사 대응용 장기 보관 | AWS Glacier, S3 | 최대 7년간 보관 |

## ****실시간 모니터링 및 분석****

### ****개요****

* 금융 시스템의 **실시간 거래**, **보안 이벤트**, **시스템 상태**를 실시간으로 모니터링하여  
  **장애 대응**과 **보안 침해 탐지**를 최적화합니다.
* **Prometheus**, **Grafana**, **Kibana**를 통해 대시보드 시각화와 **Alerting**이 구성됩니다.

### ****실시간 모니터링 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Services |

| - Spring Boot, MSA |

+----------------------------+

|

v

+----------------------------+

| Metrics Exporter |

| - Prometheus Exporter |

| - Node Exporter |

+----------------------------+

|

v

+----------------------------+

| Monitoring System |

| - Prometheus, Grafana |

| - AlertManager |

+----------------------------+

|

v

+----------------------------+

| Log Analysis & Search |

| - ELK Stack (Kibana) |

| - Logstash, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Alert Notification |

| - Slack, Email, SMS |

+----------------------------+

### ****실시간 모니터링 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **시스템 헬스 체크** | CPU, Memory, Disk 상태 실시간 모니터링 | Prometheus, Grafana | 1초 간격 실시간 체크 |
| **트랜잭션 모니터링** | 금융 거래의 실시간 상태 분석 | Prometheus, Grafana | TPS, 응답 시간 분석 |
| **에러 탐지 및 경고** | 예외 발생 시 알림 전송 | Prometheus AlertManager | Slack, Email, SMS 통보 |
| **로그 분석 및 검색** | 거래 내역 및 보안 이벤트 실시간 분석 | ELK Stack (Kibana) | 실시간 로그 탐색 |
| **보안 이벤트 탐지** | XSS, SQL Injection, Brute Force 공격 탐지 | Kibana, Prometheus | OWASP Top 10 실시간 탐지 |
| **네트워크 상태 분석** | Latency, Bandwidth 실시간 체크 | Grafana, Prometheus | 1ms 단위 네트워크 분석 |
| **장애 대응 모니터링** | 장애 발생 시 자동화된 복구 처리 | Kubernetes Auto Healing | 1분 이내 장애 복구 |

## ****보존 정책 및 규제 준수****

### ****개요****

* 금융권 시스템은 **전자금융거래법**, **개인정보 보호법**, **ISMS-P 인증 기준**에 따라 **감사 로그 보존**이 필수적입니다.
* 모든 거래 기록 및 보안 이벤트는 **최소 5년**에서 **최대 7년**간 안전하게 보관되며,  
  **AWS Glacier**, **Oracle Vault**를 통해 복구 시 무결성을 보장합니다.

### ****보존 정책 아키텍처 구성도****

sql

복사편집

+----------------------------+

| Centralized Log |

| - ELK Stack, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Long-term Archive |

| - AWS Glacier |

| - S3 Versioning |

+----------------------------+

|

v

+----------------------------+

| Encryption & Integrity |

| - AES-256, RSA-2048 |

| - HSM (CloudHSM) |

+----------------------------+

|

v

+----------------------------+

| Regulatory Compliance |

| - GDPR, ISMS-P, FSS |

+----------------------------+

### ****보존 정책 및 규제 준수 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 보존 기간 |
| **거래 내역 보관** | 모든 금융 거래 기록의 안전한 보관 | AWS Glacier, Oracle Vault | 7년 |
| **보안 이벤트 로그** | 접근 시도 및 보안 침해 탐지 기록 | ELK Stack, Prometheus | 5년 |
| **API 호출 로그** | API 트랜잭션 내역 보관 | Elasticsearch, Fluentd | 5년 |
| **접속 기록 보관** | 사용자 로그인 및 IP 기록 | Fluentd, S3 Versioning | 5년 |
| **장기 보관 암호화** | 보관 중 데이터 암호화 처리 | AES-256, RSA-2048 | 만료 시 영구 삭제 |
| **복구 및 감사 대응** | 금융감독원 감사 요청 시 복구 가능 | S3 Versioning, Glacier | 즉시 복구 가능 |

# ****(금융/제조/공공/통신) 데이터 거버넌스 및 컴플라이언스 전략****

## ****데이터 관리 체계 설계****

### ****개요****

* (금융/제조/공공/통신) 시스템의 \*\*데이터 거버넌스(Data Governance)\*\*는 **데이터의 생성, 수집, 저장, 관리, 보존, 폐기**에 이르기까지 전 과정을 통제합니다.
* 금융감독원의 **전자금융거래법**, **개인정보 보호법**, **GDPR**, **ISMS-P**를 준수합니다.
* **메타데이터 관리**, **데이터 표준화**, **품질 관리**, **접근 통제**를 체계적으로 구성합니다.

### ****데이터 관리 체계 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Data Ingestion |

| - Apache Kafka |

| - Logstash, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Data Processing Layer |

| - Apache Spark, Flink |

| - ETL (Talend, Pentaho) |

+----------------------------+

|

v

+----------------------------+

| Data Storage Layer |

| - PostgreSQL, Oracle DB |

| - AWS S3, HDFS |

+----------------------------+

|

v

+----------------------------+

| Data Governance Layer |

| - Metadata Management |

| - Data Lineage Tracking |

+----------------------------+

|

v

+----------------------------+

| Access Control & Audit |

| - RBAC, ABAC |

| - Audit Log Management |

+----------------------------+

### ****데이터 관리 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **메타데이터 관리** | 데이터 구조 및 속성 정의 | Apache Atlas, Collibra | 데이터 출처 및 관계 추적 |
| **데이터 표준화** | 형식 통일 및 표준 스키마 적용 | JSON Schema, Avro | 데이터 무결성 유지 |
| **데이터 품질 관리** | 정확성, 일관성, 최신성 관리 | Apache NiFi, Talend | 실시간 데이터 정제 |
| **데이터 접근 통제** | 권한 기반 접근 제한 | RBAC, ABAC | 민감 정보 보호 |
| **데이터 수명 주기 관리** | 생성, 보관, 폐기 단계 관리 | Apache Ranger, AWS Lifecycle | 보존 기한 자동 삭제 |
| **Data Lineage 추적** | 데이터의 흐름 및 변경 내역 추적 | Apache Atlas, Collibra | 규제 준수, 감사 대응 |
| **ETL 프로세스 관리** | 대량 데이터의 추출, 변환, 적재 | Talend, Pentaho | 데이터 손실 최소화 |
| **데이터 백업 및 복구** | 정기적 백업과 신속한 복구 | AWS S3, RDS Snapshot | 장애 시 1분 내 복구 |

## ****금융 데이터 규제 준수****

### ****개요****

* 금융 시스템의 모든 데이터는 **국내외 금융 규제**를 준수해야 합니다.
* 주요 규제 항목은 **전자금융거래법**, **ISMS-P 인증**, **GDPR (General Data Protection Regulation)** 등이 포함됩니다.
* 데이터는 **안전하게 암호화**되고, **접근 통제**와 **감사 추적**이 보장됩니다.

### ****금융 데이터 규제 준수 구성도****

pgsql

복사편집

+----------------------------+

| Regulatory Bodies |

| - FSS, KISA, GDPR |

| - 금융결제원, 금융위원회 |

+----------------------------+

|

v

+----------------------------+

| Compliance Framework |

| - ISMS-P, GDPR, PCI-DSS |

| - 전자금융감독규정 |

+----------------------------+

|

v

+----------------------------+

| Data Protection Layer |

| - AES-256, RSA-2048 |

| - Transparent Data Encryption |

+----------------------------+

|

v

+----------------------------+

| Audit & Monitoring |

| - ELK Stack, Prometheus |

| - Filebeat, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Access Control & Logs |

| - Keycloak, RBAC |

| - Audit Log Management |

+----------------------------+

### ****금융 데이터 규제 준수 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 규제 항목 | 설명 | 사용 기술 | 주요 특징 |
| **ISMS-P 인증** | 개인정보와 정보보호 관리체계 인증 | Apache Ranger, Keycloak | 개인정보 접근 통제 |
| **전자금융거래법 준수** | 금융 거래 정보의 안전한 보호 | AES-256, RSA-2048 | 거래 내역 암호화 |
| **GDPR 준수** | 유럽연합(EU) 지역 고객 정보 보호 | Keycloak, AWS KMS | Right to be Forgotten |
| **PCI-DSS 규정** | 결제 정보 보호 및 처리 보안 | AWS CloudHSM, Vault | 카드 정보 보호 |
| **FSS 규정 준수** | 금융감독원의 거래 내역 보관 | Elasticsearch, AWS S3 | 7년 보관 |
| **감사 로그 관리** | 접근 이력과 변경 사항 추적 | ELK Stack, Filebeat | 실시간 추적 및 경고 |
| **데이터 보안 정책** | 데이터의 접근 및 암호화 관리 | Vault, AWS KMS | 무단 접근 방지 |
| **데이터 파기 및 삭제** | 만료된 데이터의 안전한 삭제 | AWS S3 Lifecycle, Glacier | 데이터 복구 불가 |

## ****데이터 감사 및 추적****

### ****개요****

* 금융 시스템의 **모든 데이터 변경 및 접근 이력**은 **감사 대상**에 포함되며, **추적 가능성**이 확보되어야 합니다.
* **Apache Atlas**, **Collibra**를 활용하여 **Data Lineage**를 시각화하고,  
  **Kibana**, **Grafana**로 실시간 모니터링을 지원합니다.

### ****데이터 감사 및 추적 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

+----------------------------+

|

v

+----------------------------+

| Data Lineage Tracking |

| - Apache Atlas |

| - Collibra |

+----------------------------+

|

v

+----------------------------+

| Audit Log Collection |

| - Fluentd, Filebeat |

| - ELK Stack |

+----------------------------+

|

v

+----------------------------+

| Real-Time Monitoring |

| - Kibana, Grafana |

+----------------------------+

|

v

+----------------------------+

| Audit Reporting & Alert |

| - Prometheus, AlertManager |

+----------------------------+

### ****데이터 감사 및 추적 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **Data Lineage 추적** | 데이터의 생성부터 삭제까지 모든 흐름 추적 | Apache Atlas, Collibra | 실시간 시각화 가능 |
| **접근 이력 감사** | 사용자와 API 접근 내역 추적 | ELK Stack, Fluentd | IP, 시간, 위치 기록 |
| **변경 이력 관리** | 데이터 변경 시점 및 사용자 추적 | PostgreSQL Audit, Oracle Vault | 실시간 모니터링 |
| **감사 보고서 생성** | 감사 대응용 로그 리포트 생성 | Kibana, Grafana | 실시간 대시보드 제공 |
| **이상 탐지 및 경고** | 비정상 트랜잭션 탐지 | Prometheus AlertManager | 이상 발생 시 즉시 알림 |

# ****(금융/제조/공공/통신) 트랜잭션 처리 최적화 전략****

## ****트랜잭션 관리 아키텍처 설계****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **안정적이고 신속한 트랜잭션 처리**를 보장하기 위해 **트랜잭션 관리 아키텍처**를 설계합니다.
* 주요 기술로는 **Spring Transaction Management**, **XA Transaction**, \*\*JTA (Java Transaction API)\*\*가 사용됩니다.
* **ACID** 원칙을 준수하며, **동시성 처리**와 **데드락 방지**를 최적화합니다.

### ****트랜잭션 관리 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Transaction Management |

| - Spring Transaction |

| - JTA, XA Transactions |

+----------------------------+

|

v

+----------------------------+

| Database Layer |

| - PostgreSQL, Oracle DB |

| - MongoDB (NoSQL) |

+----------------------------+

|

v

+----------------------------+

| Logging & Monitoring |

| - ELK Stack, Prometheus |

| - Distributed Tracing |

+----------------------------+

### ****트랜잭션 관리 전략 표****

| **항목** | **설명** | **사용 기술** | **주요 특징** |
| --- | --- | --- | --- |
| **Spring Transaction** | Spring Boot 기반의 트랜잭션 처리 | Spring Data JPA | ACID 보장 |
| **JTA (Java Transaction API)** | 다중 리소스 간의 트랜잭션 관리 | Bitronix, Atomikos | 분산 트랜잭션 지원 |
| **XA Transactions** | 2PC (Two-Phase Commit) 프로토콜 지원 | JTA, Spring Boot | 다중 DB 간 일관성 보장 |
| **Connection Pool 관리** | 트랜잭션 연결 최적화 | HikariCP, Tomcat Pool | Deadlock 방지 |
| **Isolation Level 설정** | 트랜잭션 격리 수준 관리 | READ\_COMMITTED, REPEATABLE\_READ | Dirty Read 방지 |
| **Timeout 설정** | 트랜잭션 시간 초과 관리 | Spring Boot 설정 | 30초 초과 시 롤백 |
| **Retry 전략** | 트랜잭션 충돌 시 재시도 처리 | Spring Retry | 임계 구역 처리 최적화 |
| **Distributed Tracing** | 분산 트랜잭션 모니터링 | Zipkin, Jaeger | API Call Chain 추적 |

## ****동시성 처리 최적화****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **대량 트랜잭션 처리**와 **동시성 문제**를 해결하기 위해 **Lock Mechanism**과 **Optimistic Locking**을 적용합니다.
* **Pessimistic Locking**, **Optimistic Locking** 두 가지 전략을 적절히 혼합하여  
  **데드락 방지**와 **경합 최소화**를 최적화합니다.

### ****동시성 처리 아키텍처 구성도****

diff

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot Services |

| - Microservices (MSA) |

+----------------------------+

|

v

+----------------------------+

| Lock Management Layer |

| - Pessimistic Locking |

| - Optimistic Locking |

+----------------------------+

|

v

+----------------------------+

| Database Layer |

| - PostgreSQL, Oracle DB |

| - Redis Cache (Locking) |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Recovery |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

### ****동시성 처리 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **Pessimistic Locking** | 데이터 수정 시 Lock을 걸어 동시성 제어 | Spring JPA, Hibernate | Deadlock 방지 |
| **Optimistic Locking** | 데이터 수정 시 버전 체크로 경합 최소화 | @Version Annotation | 충돌 시 롤백 |
| **Read Lock (Shared Lock)** | 읽기 시 Lock을 걸어 일관성 보장 | PostgreSQL Lock | Dirty Read 방지 |
| **Write Lock (Exclusive Lock)** | 쓰기 시 Lock을 걸어 데이터 충돌 방지 | Oracle DB Lock | Write Skew 방지 |
| **Distributed Locking** | 다중 노드에서 Lock 공유 | Redis Lock, Zookeeper | 글로벌 동시성 제어 |
| **Retry with Backoff** | Lock 충돌 시 재시도 로직 | Spring Retry, Resilience4J | 경합 상황 최소화 |
| **Deadlock Detection** | 교착 상태 탐지 및 복구 | PostgreSQL Deadlock Detector | 자동 복구 처리 |
| **Lock Timeout 설정** | 특정 시간 이후 Lock 해제 | Spring Boot 설정 | 무한 대기 방지 |

## ****분산 트랜잭션 관리 전략****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **MSA(Microservices Architecture)** 기반 구조에서 **분산 트랜잭션**이 요구됩니다.
* \*\*2PC (Two-Phase Commit)\*\*와 **SAGA 패턴**을 적용하여 다중 서비스 간의 트랜잭션 일관성을 보장합니다.
* **Eventual Consistency**를 지원하며, **Fallback 처리**와 **보상 트랜잭션**을 통해 무결성을 유지합니다.

### ****분산 트랜잭션 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Microservices Layer |

| - Account Service |

| - Payment Service |

| - Notification Service |

+----------------------------+

|

v

+----------------------------+

| Distributed Coordinator |

| - Spring Cloud Sleuth |

| - Zipkin, Jaeger |

+----------------------------+

|

v

+----------------------------+

| Saga Orchestrator |

| - Event Store (Kafka) |

| - Compensation Handler |

+----------------------------+

|

v

+----------------------------+

| Data Storage Layer |

| - PostgreSQL, MongoDB |

| - Redis (Event Store) |

+----------------------------+

### ****분산 트랜잭션 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **2PC (Two-Phase Commit)** | 두 단계로 Commit을 진행하여 일관성 보장 | JTA, XA Transaction | 강력한 일관성 |
| **SAGA 패턴** | 단계별 이벤트 처리와 보상 트랜잭션 관리 | Spring Boot, Kafka | Eventual Consistency |
| **Event Store 관리** | 이벤트 기반 트랜잭션 로그 보관 | Kafka, Redis Streams | 트랜잭션 복구 가능 |
| **Compensation Transaction** | 실패 시 보상 처리 로직 실행 | Spring Retry, Circuit Breaker | 상태 복원 |
| **Orchestration vs Choreography** | 중앙 관리 또는 이벤트 기반 트랜잭션 | Spring Cloud Sleuth | 복잡도 최적화 |
| **Distributed Tracing** | 마이크로서비스 간 트랜잭션 추적 | Zipkin, Jaeger | Call Chain 시각화 |
| **Timeout 및 Rollback** | 트랜잭션 시간 초과 시 복구 | Spring Transaction | 롤백 시 무결성 유지 |
| **Data Consistency Check** | 트랜잭션 처리 후 데이터 검증 | Prometheus, Grafana | 실시간 모니터링 |

# ****(금융/제조/공공/통신) 시스템 배포 및 릴리즈 전략****

## ****배포 아키텍처 설계****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **안정적인 배포와 무중단 서비스**를 보장하기 위해 **배포 아키텍처**를 설계합니다.
* 주요 기술로는 **Kubernetes**, **Docker**, **ArgoCD**, **Helm**이 사용됩니다.
* **Blue-Green Deployment**, **Canary Release**, **Rolling Update** 방식을 적용하여 **Zero Downtime**을 실현합니다.

### ****배포 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| CI/CD Pipeline |

| - Jenkins, GitLab CI |

| - ArgoCD, Helm |

+----------------------------+

|

v

+----------------------------+

| Docker Image Registry |

| - Nexus, JFrog, ECR |

+----------------------------+

|

v

+----------------------------+

| Kubernetes Cluster |

| - EKS, GKE, AKS |

| - Auto Scaling, HPA |

+----------------------------+

|

v

+----------------------------+

| Load Balancer & Gateway |

| - Nginx, Spring Gateway |

| - API Gateway |

+----------------------------+

### ****배포 아키텍처 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **CI/CD 파이프라인** | 지속적 통합 및 배포 자동화 | Jenkins, GitLab CI | 자동화된 빌드 & 테스트 |
| **Docker 이미지 관리** | 컨테이너 이미지를 저장 및 배포 | Nexus, JFrog | 버전 관리 및 롤백 가능 |
| **Helm Chart 배포** | Kubernetes 리소스 관리 | Helm, ArgoCD | 선언적 배포 관리 |
| **Kubernetes 클러스터 관리** | 분산 처리 및 무중단 배포 | EKS, GKE, AKS | Auto Healing, Scaling |
| **Auto Scaling 설정** | 트래픽 증가 시 자동 확장 | Kubernetes HPA | 부하에 따른 동적 확장 |
| **Load Balancer 구성** | 서비스 트래픽 분산 | Nginx, HAProxy | 장애 시 무중단 전환 |
| **API Gateway 설정** | 내부 서비스 간 경로 설정 | Spring Cloud Gateway | 외부 API 라우팅 |

## ****무중단 배포 전략****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **연속적 서비스 유지**를 위해 **Zero Downtime Deployment**를 설계합니다.
* 배포 전략으로는 **Blue-Green Deployment**, **Canary Release**, **Rolling Update**를 적용합니다.
* 무중단 배포를 통해 (금융/제조/공공/통신) 거래 중단 없이 시스템 업데이트가 가능합니다.

### ****무중단 배포 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| CI/CD Pipeline |

| - Jenkins, ArgoCD |

+----------------------------+

|

v

+----------------------------+

| Blue-Green Deployment |

| - Kubernetes Services |

+----------------------------+

|

v

+----------------------------+

| Canary Release |

| - Istio, Argo Rollouts |

+----------------------------+

|

v

+----------------------------+

| Rolling Update |

| - Kubernetes Deployment |

+----------------------------+

|

v

+----------------------------+

| Load Balancer & Gateway |

| - Nginx, HAProxy |

+----------------------------+

### ****무중단 배포 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 배포 전략 | 설명 | 사용 기술 | 주요 특징 |
| **Blue-Green Deployment** | 두 개의 환경(Blue, Green)을 번갈아 배포 | Kubernetes, ArgoCD | 빠른 롤백, Zero Downtime |
| **Canary Release** | 일부 트래픽만 신규 버전으로 전환 | Istio, Argo Rollouts | 리스크 최소화 |
| **Rolling Update** | Pod를 순차적으로 업데이트 | Kubernetes Deployment | 트래픽 중단 없이 배포 |
| **Shadow Deployment** | 실제 요청 없이 트래픽 미러링 | Istio | 서비스 영향 없이 검증 |
| **Traffic Shifting** | 점진적 트래픽 증가 | Nginx, HAProxy | 트래픽 단계별 확장 |
| **Auto Healing** | 배포 중 장애 시 자동 복구 | Kubernetes Auto Healing | 1분 내 서비스 복구 |
| **Real-time Monitoring** | 배포 상태 실시간 추적 | Prometheus, Grafana | 배포 중 이상 탐지 |

## ****롤백 및 장애 대응 전략****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **배포 중 장애 발생 시 즉각적인 복구**를 위해 **롤백 전략**을 설계합니다.
* 주요 기술로는 **Helm Rollback**, **Argo Rollback**, **Kubernetes ReplicaSet**이 사용됩니다.
* 실시간 장애 탐지와 **Failover 처리**를 통해 서비스 중단 없이 복구가 가능합니다.

### ****롤백 및 장애 대응 아키텍처 구성도****

sql

복사편집

+----------------------------+

| Real-time Monitoring |

| - Prometheus, Grafana |

+----------------------------+

|

v

+----------------------------+

| Alerting & Notification |

| - Slack, Email, PagerDuty |

+----------------------------+

|

v

+----------------------------+

| Rollback Mechanism |

| - Helm Rollback |

| - Argo Rollback |

+----------------------------+

|

v

+----------------------------+

| Failover Handling |

| - Kubernetes ReplicaSet |

| - Consul, Istio |

+----------------------------+

|

v

+----------------------------+

| Disaster Recovery (DR) |

| - AWS Multi-AZ |

| - GCP Multi-Region |

+----------------------------+

### ****롤백 및 장애 대응 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **Helm Rollback** | 배포 실패 시 이전 상태로 복구 | Helm, ArgoCD | 1초 내 복구 가능 |
| **Argo Rollback** | Kubernetes Deployment의 이전 상태 복구 | ArgoCD | 버전 히스토리 관리 |
| **ReplicaSet Failover** | 장애 시 복제본으로 전환 | Kubernetes, Consul | 무중단 서비스 유지 |
| **Auto Scaling Recovery** | 트래픽 증가 시 자동 확장 | Kubernetes HPA | 자원 부족 시 확장 |
| **Multi-AZ Failover** | 리전 장애 시 자동 전환 | AWS Multi-AZ | 트래픽 손실 방지 |
| **Disaster Recovery (DR)** | 장애 발생 시 즉각 복구 | AWS RDS Multi-AZ | 데이터 유실 방지 |
| **Alerting & Monitoring** | 장애 발생 시 실시간 알림 | Prometheus, Grafana | Slack, Email 통보 |

# ****(금융/제조/공공/통신) 시스템 성능 최적화 및 진단 전략****

## ****성능 최적화 아키텍처 설계****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **대량 거래 처리**와 **실시간 응답성**을 최적화하기 위해 **성능 개선 아키텍처**를 설계합니다.
* 주요 기술로는 **Spring Boot 최적화**, **JPA 성능 튜닝**, **Cache 관리**, **Connection Pool 튜닝**이 사용됩니다.
* 성능 저하 요인을 사전 탐지하고, **모니터링과 성능 분석**을 통해 병목 구간을 진단합니다.

### ****성능 최적화 아키텍처 구성도****

diff

복사편집

+----------------------------+

| Application Layer |

| - Spring Boot, MSA |

+----------------------------+

|

v

+----------------------------+

| Performance Tuning |

| - JPA Query Optimization |

| - Cache Optimization |

+----------------------------+

|

v

+----------------------------+

| Connection Pool Tuning |

| - HikariCP, Tomcat Pool |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Profiling |

| - Prometheus, Grafana |

| - JProfiler, JVisualVM |

+----------------------------+

|

v

+----------------------------+

| Data Caching Layer |

| - Redis, Memcached |

+----------------------------+

### ****성능 최적화 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **JPA 쿼리 최적화** | JPQL 및 Native Query 개선 | Spring Data JPA | N+1 문제 방지 |
| **Cache 사용 최적화** | 빈번한 조회 데이터 캐싱 | Redis, Ehcache | 응답 속도 50% 개선 |
| **Connection Pool 관리** | 동시 접속 트랜잭션 최적화 | HikariCP, Tomcat Pool | Connection Timeout 방지 |
| **비동기 처리 최적화** | 비동기 방식으로 I/O 성능 개선 | CompletableFuture, @Async | I/O 병목 해소 |
| **프로파일링 및 모니터링** | 메모리 사용 및 CPU 부하 분석 | JProfiler, Prometheus | 성능 병목 사전 탐지 |
| **GC 튜닝** | 메모리 관리 및 GC Pause 최소화 | G1GC, ZGC | GC 시간 20% 감소 |
| **Thread Pool 최적화** | 비동기 처리 시 스레드 최적화 | Executors, ThreadPoolTaskExecutor | 스레드 과부하 방지 |
| **SQL 인덱스 튜닝** | 조회 속도 향상을 위한 인덱스 최적화 | B-Tree, Hash Index | 조회 속도 30% 향상 |

## ****실시간 성능 진단 체계****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **실시간 성능 진단**을 통해 **거래 지연**과 **응답 시간 초과**를 즉각 탐지합니다.
* **Prometheus**, **Grafana**, **ELK Stack**을 통해 실시간 모니터링을 수행하며,  
  **APM (Application Performance Monitoring)** 도구를 사용하여 **성능 메트릭**을 분석합니다.

### ****성능 진단 아키텍처 구성도****

sql

복사편집

+----------------------------+

| Application Services |

| - Spring Boot, MSA |

+----------------------------+

|

v

+----------------------------+

| Monitoring Exporters |

| - Prometheus Exporter |

| - Micrometer, Actuator |

+----------------------------+

|

v

+----------------------------+

| APM & Profiling |

| - New Relic, Dynatrace |

| - JProfiler, ELK Stack |

+----------------------------+

|

v

+----------------------------+

| Real-Time Monitoring |

| - Grafana, Kibana |

| - Prometheus Alerting |

+----------------------------+

### ****실시간 성능 진단 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **TPS 모니터링** | 초당 거래 건수 모니터링 | Prometheus, Grafana | 목표 TPS 10,000 |
| **응답 시간 분석** | API 응답 속도 측정 | Micrometer, Actuator | 응답 시간 3초 이하 |
| **CPU 및 메모리 사용량 분석** | 리소스 과부하 탐지 | JProfiler, Prometheus | CPU 사용률 70% 이하 |
| **GC 모니터링** | 메모리 청소 주기 모니터링 | JVisualVM, JMX Exporter | Full GC 횟수 감소 |
| **SQL 성능 진단** | 느린 쿼리 탐지 | pg\_stat\_statements, Oracle AWR | 쿼리 최적화 필요 구간 식별 |
| **네트워크 대기 시간 분석** | 지연 시간 초과 탐지 | Grafana, ELK Stack | Latency 200ms 이하 |
| **스레드 상태 모니터링** | Deadlock 및 Block 상태 탐지 | JMX Exporter, Prometheus | 비정상 스레드 탐지 |
| **에러 로그 분석** | 시스템 오류 실시간 탐지 | ELK Stack (Kibana) | 에러 발생 즉시 경고 |

## ****부하 테스트 및 스트레스 테스트****

### ****개요****

* (금융/제조/공공/통신) 시스템의 **고부하 상황 대응**을 위해 \*\*부하 테스트(Load Test)\*\*와 \*\*스트레스 테스트(Stress Test)\*\*를 수행합니다.
* **JMeter**, **Gatling**, **Locust**를 사용하여 **대량 트랜잭션 처리 능력**을 검증합니다.
* 테스트 결과는 **Grafana 대시보드**로 시각화하여 성능 한계와 병목 구간을 분석합니다.

### ****부하 테스트 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Load Generator |

| - JMeter, Gatling, Locust|

+----------------------------+

|

v

+----------------------------+

| API Gateway Layer |

| - Spring Cloud Gateway |

+----------------------------+

|

v

+----------------------------+

| Microservices Cluster |

| - Kubernetes Pods |

| - Horizontal Pod Autoscaler |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Reporting |

| - Grafana, Prometheus |

| - ELK Stack |

+----------------------------+

### ****부하 테스트 및 스트레스 테스트 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 유형 | 설명 | 사용 도구 | 목표 성능 |
| **부하 테스트** | 대량 요청 시 성능 측정 | JMeter, Gatling | 10,000 TPS |
| **스트레스 테스트** | 최대 처리 한계 초과 상황 확인 | Locust, K6 | 최대 20,000 TPS |
| **내구성 테스트** | 장기 부하 상태에서 안정성 확인 | JMeter, BlazeMeter | 24시간 지속 |
| **스파이크 테스트** | 갑작스러운 트래픽 급증 확인 | Gatling, K6 | 1분간 20,000 TPS |
| **회복성 테스트** | 장애 발생 후 성능 복구 확인 | JMeter, Jenkins | 1분 이내 정상화 |
| **데이터 일관성 테스트** | 트랜잭션 처리 후 데이터 무결성 확인 | Postman, Newman | DB 일관성 100% |
| **성능 보고서 생성** | 테스트 결과 자동 보고서 작성 | Allure Report | TPS, 응답 시간, 오류율 포함 |

# ****(금융/제조/공공/통신) 시스템 장애 대응 및 복구 전략****

## ****장애 유형 분석 및 대응 방안****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **무중단 서비스**를 보장하기 위해 **장애 유형**을 사전에 분석하고 **대응 방안**을 마련합니다.
* 주요 장애 유형은 **서비스 장애**, **DB 장애**, **네트워크 장애**, **보안 침해**, **애플리케이션 오류** 등입니다.
* **장애 탐지-분석-복구**의 3단계 절차를 기반으로 **자동화 대응 체계**를 구축합니다.

### ****장애 대응 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Monitoring Layer |

| - Prometheus, Grafana |

| - ELK Stack, Fluentd |

+----------------------------+

|

v

+----------------------------+

| Alerting & Notification|

| - AlertManager, Slack |

| - PagerDuty, Email |

+----------------------------+

|

v

+----------------------------+

| Auto Recovery Layer |

| - Kubernetes Auto Heal |

| - HAProxy Failover |

+----------------------------+

|

v

+----------------------------+

| Backup & Restore |

| - AWS RDS Snapshot |

| - S3 Versioning |

+----------------------------+

|

v

+----------------------------+

| Disaster Recovery (DR) |

| - Multi-AZ Deployment |

| - Active-Active Cluster |

+----------------------------+

### ****장애 유형 분석 및 대응 방안 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 장애 유형 | 설명 | 대응 방안 | 사용 기술 |
| **서비스 장애** | 애플리케이션 응답 불가 또는 오류 발생 | Auto Healing, Rolling Restart | Kubernetes, HAProxy |
| **DB 장애** | 데이터베이스 연결 실패 또는 성능 저하 | DB Failover, Replication | PostgreSQL Streaming Replication |
| **네트워크 장애** | 서버 간 통신 불가 또는 네트워크 혼잡 | 이중화 구성, DNS Failover | AWS Route 53, Nginx |
| **보안 침해** | 악성 코드, 무단 접근 | WAF 설정, 침입 탐지 | AWS WAF, Snort |
| **애플리케이션 오류** | 코드 버그 또는 메모리 누수 | Auto Rollback, Hotfix 배포 | ArgoCD, Helm |
| **파일 손상 또는 유실** | 로그 파일 또는 데이터 유실 | 파일 복구, S3 복원 | AWS S3, Glacier |
| **캐시 서버 장애** | 캐시 데이터 손실 또는 접근 불가 | 캐시 클러스터 복구 | Redis Cluster, Memcached |
| **API Gateway 장애** | API 요청 라우팅 오류 | Gateway 교체, 트래픽 우회 | Spring Cloud Gateway |

## ****데이터 복구 전략****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **데이터 손실 최소화**와 **신속한 복구**를 위해 **데이터 복구 전략**을 수립합니다.
* **백업 주기 설정**, **실시간 복제**, **데이터 무결성 검사**를 통해 데이터 복구 시간을 단축합니다.
* **AWS RDS Snapshot**, **EBS 스냅샷**, **S3 버전 관리** 등을 활용하여 데이터 복구 체계를 마련합니다.

### ****데이터 복구 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Backup & Archive Layer |

| - AWS S3, Glacier |

| - PostgreSQL Backup |

+----------------------------+

|

v

+----------------------------+

| Replication Layer |

| - Streaming Replication |

| - Multi-AZ Failover |

+----------------------------+

|

v

+----------------------------+

| Backup Orchestrator |

| - Jenkins Backup Job |

| - AWS Backup |

+----------------------------+

|

v

+----------------------------+

| Recovery Layer |

| - PITR (Point-In-Time) |

| - Data Restoration |

+----------------------------+

### ****데이터 복구 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **정기 백업** | 매일 또는 매주 정기적으로 데이터 백업 | AWS RDS Snapshot | 주기적 백업 스케줄 |
| **증분 백업** | 데이터 변경분만 백업하여 성능 최적화 | AWS Backup, Borg | 스토리지 절감 |
| **데이터 복제** | 실시간 DB 복제 및 동기화 | PostgreSQL Streaming | 장애 시 복구 속도 향상 |
| **복구 지점 설정 (PITR)** | 특정 시점으로 복구 가능 | WAL Archiving | 거래 데이터 복구 |
| **데이터 무결성 검사** | 복구 후 데이터 일관성 확인 | pg\_dump, DB Consistency Check | 무결성 보장 |
| **복구 테스트** | 정기적 복구 시나리오 검증 | Jenkins, AWS Backup | 복구 실패 대비 |
| **로그 데이터 복원** | 로그 손실 시 복구 가능 | ELK Stack, Fluentd | 트랜잭션 추적 가능 |
| **장기 보관 데이터 복구** | 7년 이상 데이터 복원 | AWS Glacier | 규제 준수 보장 |

## ****28.3 장애 모니터링 체계 구축****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **지속적 모니터링**을 통해 장애를 **사전 감지**하고 **신속 대응**할 수 있도록 합니다.
* **Prometheus**, **Grafana**, **ELK Stack**을 활용하여 실시간 모니터링 대시보드를 구축합니다.
* 장애 발생 시 **Slack**, **PagerDuty**로 알림을 전송하여 **즉각 대응**이 가능하도록 합니다.

### ****장애 모니터링 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Monitoring System |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

|

v

+----------------------------+

| Alerting & Notification |

| - AlertManager, Slack |

| - PagerDuty, Email |

+----------------------------+

|

v

+----------------------------+

| Incident Management |

| - Jira, Confluence |

| - RCA (Root Cause Analysis) |

+----------------------------+

|

v

+----------------------------+

| Reporting & Analysis |

| - Grafana Reporting |

| - ELK Log Analysis |

+----------------------------+

### ****장애 모니터링 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **실시간 장애 모니터링** | 서버 상태 및 거래 트래픽 모니터링 | Prometheus, Grafana | 장애 발생 즉시 탐지 |
| **로그 분석 및 경고** | 애플리케이션 오류 추적 | ELK Stack, Fluentd | 에러 발생 시 Slack 알림 |
| **성능 모니터링** | TPS, 응답 시간, CPU 사용량 분석 | Prometheus, Grafana | 목표 성능 10,000 TPS |
| **장애 보고서 생성** | 장애 원인 분석 및 대책 마련 | Grafana Reporting | 장애 이력 관리 |
| **자동화 경고 전송** | 장애 발생 시 실시간 알림 | AlertManager, PagerDuty | 다중 채널 알림 |
| **이상 트래픽 감지** | DDoS 공격 시 자동 경고 | AWS WAF, Cloudflare | 비정상 요청 차단 |

# ****(금융/제조/공공/통신) 서비스 연속성 및 비상 복구 전략****

## ****서비스 연속성 관리****

### ****개요****

* (금융/제조/공공/통신)권 시스템은 **24x7 무중단 서비스**를 요구하며, **재해 발생 시에도 신속한 복구**가 필수적입니다.
* \*\*비상 대응 체계(Business Continuity Plan, BCP)\*\*를 수립하여 서비스 중단 위험을 최소화합니다.
* **Active-Active 클러스터링**, **Multi-AZ 구성**, **데이터 복제 및 백업** 전략을 통해 **서비스 연속성**을 확보합니다.

### ****서비스 연속성 관리 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Active-Active Cluster |

| - Kubernetes Cluster |

| - Multi-AZ Deployment |

+----------------------------+

|

v

+----------------------------+

| Load Balancer Layer |

| - Nginx, HAProxy |

+----------------------------+

|

v

+----------------------------+

| Data Replication Layer |

| - PostgreSQL Streaming |

| - Oracle Data Guard |

+----------------------------+

|

v

+----------------------------+

| Backup & Archive Layer |

| - AWS S3, Glacier |

| - RDS Snapshot |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Alerting |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

### ****서비스 연속성 관리 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **Active-Active Cluster** | 다중 인스턴스 동시 활성화로 무중단 서비스 | Kubernetes Cluster, HAProxy | 장애 시 즉시 전환 |
| **Multi-AZ 구성** | 가용성 영역(AZ) 분산 배치 | AWS Multi-AZ, GCP Multi-Region | 지리적 이중화 |
| **데이터 복제** | 실시간 데이터 동기화 | PostgreSQL Streaming Replication | 데이터 일관성 보장 |
| **백업 주기 관리** | 데이터 손실 최소화를 위한 백업 주기 설정 | AWS RDS Snapshot | 주기적 자동 백업 |
| **장애 탐지 및 경고** | 장애 발생 시 즉시 경고 전송 | Prometheus AlertManager | 다중 채널 알림 |
| **Failover 처리** | 장애 시 자동 전환 | HAProxy, Kubernetes | 서비스 연속성 유지 |
| **자동화 복구** | 장애 감지 후 자동 재기동 | Kubernetes Auto Healing | 1분 이내 복구 |
| **상황별 대응 매뉴얼** | 장애 유형별 복구 절차 명시 | Confluence, Jira | 상황별 지침 문서화 |

## ****비상 복구 계획 수립****

### ****개요****

* (금융/제조/공공/통신)권 시스템의 **대규모 장애 발생** 또는 **재해 상황**에서도 **서비스를 신속히 복구**하기 위해 **DR(Disaster Recovery) 계획**을 수립합니다.
* **핵심 데이터 복제**, **DR 전용 클러스터 구성**, **DR 센터 구축**을 통해 비상 상황을 대비합니다.

### ****비상 복구 아키텍처 구성도****

pgsql

복사편집

+----------------------------+

| Primary Data Center |

| - Kubernetes Cluster |

| - PostgreSQL, Oracle DB |

+----------------------------+

|

v

+----------------------------+

| Data Replication Layer |

| - Real-Time Streaming |

| - Backup & Restore |

+----------------------------+

|

v

+----------------------------+

| DR Data Center |

| - Backup Cluster |

| - RDS Standby Instance |

+----------------------------+

|

v

+----------------------------+

| DR Monitoring Layer |

| - Prometheus, Grafana |

| - ELK Stack |

+----------------------------+

|

v

+----------------------------+

| Failover & Switch Layer |

| - DNS Failover |

| - API Gateway Switching |

+----------------------------+

### ****비상 복구 계획 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 항목 | 설명 | 사용 기술 | 주요 특징 |
| **DR 센터 구축** | 재해 발생 시 즉시 전환 가능한 별도 데이터 센터 | AWS DR Site, Multi-Region | 주요 금융 데이터 보호 |
| **실시간 데이터 복제** | 주 센터와 DR 센터 간 실시간 데이터 동기화 | PostgreSQL Streaming, Oracle Data Guard | 데이터 무결성 유지 |
| **DR 클러스터 구성** | 이중화로 구성된 DR 전용 클러스터 | Kubernetes Backup Cluster | 서비스 중단 방지 |
| **DNS 자동 전환** | 주 센터 장애 시 DR 센터로 도메인 전환 | AWS Route 53, Cloudflare | 사용자 영향 최소화 |
| **API Gateway 전환** | 서비스 장애 시 트래픽 우회 | Spring Cloud Gateway | API 연속성 보장 |
| **복구 절차 자동화** | 장애 발생 시 자동 복구 실행 | Jenkins, Ansible | 수동 개입 최소화 |
| **DR 테스트** | DR 전환 시나리오 정기 테스트 | Jenkins Pipeline | DR 절차 검증 |
| **데이터 무결성 검증** | 복구 후 데이터 정합성 확인 | pg\_dump, Data Consistency Check | 데이터 손실 방지 |

## ****DR 시나리오 테스트****

### ****개요****

* **재해 발생 시 대응 능력**을 검증하기 위해 **정기적인 DR 시나리오 테스트**를 수행합니다.
* **실전 상황과 유사한 테스트**를 통해 **자동화 복구 체계**와 **전환 속도**를 검증합니다.

### ****DR 시나리오 테스트 구성도****

sql

복사편집

+----------------------------+

| DR Simulation Layer |

| - Jenkins, Ansible |

+----------------------------+

|

v

+----------------------------+

| Disaster Simulation |

| - DB 장애 시나리오 |

| - 네트워크 장애 시나리오 |

+----------------------------+

|

v

+----------------------------+

| Monitoring & Reporting |

| - Grafana, Kibana |

+----------------------------+

|

v

+----------------------------+

| Test Result Analysis |

| - Failure Recovery Rate |

| - RTO/RPO Measurement |

+----------------------------+

### ****DR 시나리오 테스트 전략 표****

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 유형 | 설명 | 사용 도구 | 주요 목표 |
| **데이터베이스 장애 테스트** | DB 연결 불가 상황 복구 | Jenkins, pg\_dump | RTO 5분 이내 |
| **네트워크 단절 테스트** | 클러스터 간 통신 불가 | Ansible, AWS VPC | 장애 탐지 시간 1분 이내 |
| **서비스 전환 테스트** | DR 센터로 서비스 전환 속도 측정 | AWS Route 53 | 사용자 영향 최소화 |
| **API 응답성 테스트** | DR 환경에서 API 속도 측정 | Postman, JMeter | 응답 시간 3초 이내 |
| **데이터 정합성 검증** | 복구 후 데이터 손실 여부 확인 | Grafana, Prometheus | 데이터 유실율 0% |
| **복구 시나리오 검증** | DR 프로세스 유효성 확인 | Jenkins, Jira | 복구 절차 오류 0건 |

# ****금융 시스템 개선 과제 및 제언****

## ****성능 개선 과제****

금융 시스템의 성능 향상을 위해 다음과 같은 전략을 고려할 수 있습니다:

* **마이크로서비스 아키텍처(MSA) 도입**: 시스템을 유연하게 구성하여 확장성과 유지보수성을 높입니다.
* **클라우드 기반 인프라 활용**: AWS, Azure 등의 클라우드 서비스를 통해 자원 확장성과 가용성을 확보합니다.
* **데이터베이스 최적화**: 인덱스 재구성, 쿼리 튜닝 등을 통해 데이터 처리 속도를 개선합니다.
* **캐시 시스템 도입**: Redis, Memcached 등을 활용하여 자주 조회되는 데이터를 캐싱함으로써 응답 시간을 단축합니다.
* **모니터링 및 자동화 도구 활용**: Prometheus, Grafana 등을 통해 시스템 상태를 실시간으로 모니터링하고, Ansible, Jenkins 등을 이용하여 배포 및 운영을 자동화합니다.

## ****보안 강화 방안****

금융 시스템의 보안을 강화하기 위해 다음과 같은 방안을 고려할 수 있습니다:

* **다층 보안 체계 구축**: 네트워크, 애플리케이션, 데이터베이스 등 각 계층별로 보안 대책을 마련합니다.
* **망분리 정책 강화**: 업무망과 인터넷망을 분리하여 외부 공격으로부터 내부 시스템을 보호합니다.
* **지속적인 보안 교육 및 훈련**: 직원들을 대상으로 정기적인 보안 교육과 모의 훈련을 실시하여 보안 인식을 제고합니다.
* **보안 취약점 점검 및 패치 관리**: 정기적으로 시스템의 보안 취약점을 점검하고, 최신 보안 패치를 적용합니다.
* **사이버 위협 인텔리전스 활용**: S2W 등의 보안 전문 기관에서 제공하는 위협 정보를 활용하여 선제적으로 대응합니다.[S2W : AI-powered Data Intelligence+1보안뉴스+1](https://s2w.inc/ko/resource/detail/601?utm_source=chatgpt.com)

## ****운영 효율성 제고 방안****

운영 효율성을 높이기 위해 다음과 같은 전략을 고려할 수 있습니다:

* **프로세스 자동화**: RPA(Robotic Process Automation)를 도입하여 반복적인 업무를 자동화합니다.
* **IT 자산 관리 체계 강화**: IT 자산의 현황을 정확히 파악하고, 효율적으로 관리합니다.
* **서비스 수준 관리(SLM) 도입**: 서비스 수준 협약(SLA)을 체결하고, 이를 기반으로 서비스 품질을 관리합니다.
* **지속적인 성과 평가 및 개선**: 운영 성과를 정기적으로 평가하고, 개선 사항을 도출하여 반영합니다.
* **협업 도구 활용**: Jira, Confluence 등의 협업 도구를 활용하여 업무 효율성을 높입니다.

위의 전략들을 체계적으로 실행함으로써 금융 시스템의 성능, 보안, 운영 효율성을 종합적으로 향상시킬 수 있습니다. 각 전략의 세부 실행 방안과 기대 효과에 대해 더 자세한 정보가 필요하시면 언제든지 문의해 주시기 바랍니다.

# 응용 아키텍처 산출물 정의

## 도입 전 안내말

Application Architecture(AA) 수행 방안의 핵심 중 하나는 명확한 산출물 정의입니다. 응용 아키텍처의 산출물은 프로젝트 전반의 품질을 보장하고, 각 단계에서의 진행 사항을 명확하게 문서화하는 역할을 합니다. 특히 금융 및 공공기관 프로젝트에서는 안정성, 확장성, 무결성의 보장을 위한 체계적인 산출물 관리가 필수적입니다.

본 장에서는 AA 수행 시 도출되는 주요 산출물을 정의하고, 각 산출물의 역할과 상호 연관성을 명확히 설명합니다.

## 주요 산출물 정의 요약

|  |  |  |
| --- | --- | --- |
| 주요 산출물 | 설명 | 주요 적용 사례 |
| 요구사항 정의서 | 프로젝트 초기 단계에서 수집된 요구사항을 체계적으로 정리하고 분석한 문서 | 금융거래 처리 요구사항 정의, 실시간 데이터 처리 |
| 설계 명세서 | 정의된 요구사항을 기반으로 아키텍처 설계를 문서화한 자료 | 대량 트랜잭션 구조 설계, 데이터 흐름 최적화 |
| 데이터 모델링 문서 | 데이터 흐름과 저장 구조를 시각적으로 표현한 자료 | 대출 심사 DB 모델링, 고객 정보 통합 관리 |
| 인터페이스 명세서 | 내부 및 외부 시스템 간 데이터 전송 규격을 정의한 문서 | ERP 연동, 금융 API 표준화 |
| 테스트 계획서 | 개발된 시스템이 요구사항에 부합하는지 검증하기 위한 계획 | 금융 거래 시나리오 테스트, 대량 배치 테스트 |
| 운영 계획서 | 시스템의 배포 및 유지보수 전략을 정의한 문서 | 클라우드 배포 전략, 장애 대응 계획 |
| 보안 계획서 | 시스템 보안을 위한 인증, 접근 제어, 데이터 암호화 계획 | OAuth2.0 인증 설계, SSL 통신 보안 |

## 요구사항 정의서

### 개요

* 프로젝트 초기 단계에서 수집된 요구사항을 체계적으로 정리하고 분석한 문서입니다.
* 요구사항 정의서는 이후 설계 및 개발 단계에서 참조 기준이 되며, 명확한 기능 정의와 비기능 요구사항을 포함합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 기능 요구사항 | 서비스가 제공해야 하는 주요 기능 정의 | 계좌 조회, 거래 이력 조회 |
| 비기능 요구사항 | 성능, 확장성, 보안성 등 시스템 특성 정의 | 실시간 거래 처리, 장애 복구 |
| 요구사항 추적성 매트릭스 | 요구사항 변경 시 추적 가능하도록 매핑 | 대출 심사 프로세스, 송금 서비스 |
| 요구사항 변경 관리 | 프로젝트 진행 중 발생하는 요구사항 변경 관리 | 신상품 추가, 규제 변경 대응 |

## 설계 명세서

### 개요

* 정의된 요구사항을 기반으로 아키텍처 설계를 문서화한 자료입니다.
* 금융 거래의 복잡한 비즈니스 로직을 최적화하고, 시스템 간의 연계성을 높이는 설계를 포함합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 계층 구조 설계 | Presentation, Business, Data Layer로 분리 | 인터넷 뱅킹, 전자 금융 시스템 |
| 데이터 흐름 설계 | 시스템 간 데이터의 흐름을 최적화하여 처리 속도 개선 | 실시간 결제, 카드 승인 |
| 트랜잭션 처리 설계 | 다단계 거래 처리 및 동시성 제어 | 대량 송금, 펀드 매매 |
| MSA 구조 도입 | 마이크로서비스를 기반으로 업무 단위 모듈화 | 글로벌 결제, 환전 시스템 |

## 데이터 모델링 문서

### 개요

* 데이터 흐름과 저장 구조를 시각적으로 표현한 자료입니다.
* 금융권 프로젝트에서는 실시간 거래 정보, 고객 정보, 대출 심사 정보를 효과적으로 관리할 수 있도록 모델링합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| ERD(Entity Relationship Diagram) | 주요 엔티티 간의 관계 정의 | 고객, 거래 내역, 대출 정보 |
| 스키마 정의 | 최적화된 테이블 설계 및 인덱스 구조 | 고객 정보 DB, 거래 로그 DB |
| 정규화 및 최적화 | 중복 데이터 제거 및 데이터 무결성 확보 | 고객 정보 중복 제거, 대출 이력 관리 |
| 데이터 마이그레이션 | 기존 데이터의 이관 및 변환 | 레거시 시스템 → 클라우드 DB |

## 인터페이스 명세서

### 개요

* 인터페이스 명세서는 내부 및 외부 시스템 간 데이터 전송 규격을 정의한 문서입니다. 금융권 시스템은 다양한 외부 서비스와 연계가 많기 때문에 명확한 인터페이스 정의가 필수적입니다.
* 특히 금융 API, 오픈 뱅킹, 외부 결제 게이트웨이와의 통신 규격을 표준화하여 안정적인 데이터 전송이 가능하도록 설계합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| RESTful API 설계 | HTTP 기반의 RESTful API 정의 | 온라인 뱅킹, 모바일 결제 |
| SOAP API 설계 | 금융 전송을 위한 SOAP 프로토콜 정의 | 보험 조회, 대출 심사 연계 |
| 데이터 포맷 정의 | JSON, XML 표준 포맷 적용 | 핀테크 서비스 연동, 국제 송금 |
| 인터페이스 보안 | OAuth2.0, JWT 인증 구조 설계 | 오픈 API, 클라우드 데이터 통신 |
| 상호연동 최적화 | 이기종 시스템 간 데이터 호환성 확보 | ERP 연동, 회계 시스템 통합 |

## 테스트 계획서

### 개요

* 개발된 시스템이 요구사항에 부합하는지 검증하기 위한 테스트 계획서입니다.
* 금융권 프로젝트에서는 대량 트랜잭션, 실시간 데이터 처리, 다단계 인증 등 고도화된 테스트가 요구됩니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 부하 테스트 | 최대 거래량에 대한 시스템 처리 성능 검증 | 대량 이체, 정산 프로세스 |
| 침투 테스트 | 보안 취약점을 탐색하고 보완 | 웹 애플리케이션, API Gateway |
| 회귀 테스트 | 업데이트 후 기존 기능의 정상 작동 여부 확인 | 정기 업데이트, 보안 패치 |
| 통합 테스트 | 각 모듈 간의 통신과 데이터 연동 검증 | 결제 시스템 ↔ 은행 시스템 연동 |
| 사용자 수용 테스트 (UAT) | 최종 사용자 환경에서 요구사항 충족 여부 확인 | 모바일 뱅킹 앱, 인터넷 뱅킹 시스템 |

## 운영 계획서

### 개요

* 시스템의 배포 및 유지보수 전략을 정의한 문서입니다.
* 금융 서비스의 특성상 24시간 무중단 운영이 요구되며, 장애 대응 및 복구 절차가 명확히 정의되어야 합니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 배포 계획 | 서비스 무중단 배포 전략 및 스케줄 정의 | 클라우드 배포, 롤링 업데이트 |
| 장애 대응 계획 | 장애 발생 시 빠른 복구를 위한 절차 | 이중화 구성, 장애 전이 전략 |
| 백업 및 복구 | 데이터 손실 방지를 위한 백업 및 복구 방안 | 일일 백업, 주간 증분 백업 |
| 유지보수 계획 | 정기 점검 및 업데이트 계획 | 금융 시스템 보안 패치, DB 최적화 |
| 모니터링 계획 | 실시간 시스템 상태 감시 및 알림 | TPS 모니터링, 장애 알림 시스템 |

## 보안 계획서

### 개요

* 시스템 보안을 위한 인증, 접근 제어, 데이터 암호화 계획을 정의한 문서입니다.
* 금융 거래의 안정성을 높이고, 고객 데이터를 보호하기 위한 체계적인 보안 정책이 필수적입니다.

### 주요 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 설명 | 적용 사례 |
| 인증 설계 | 사용자 접근을 제어하기 위한 다단계 인증 | OTP, 생체 인증, MFA |
| 접근 제어 정책 | 사용자 권한에 따른 접근 관리 | 고객, 관리자, 협력사 구분 |
| 데이터 암호화 | AES, RSA 기반의 민감 정보 암호화 | 금융 거래 데이터, 고객 정보 |
| 통신 보안 | SSL/TLS를 통한 전송 데이터 보호 | 인터넷 뱅킹, 모바일 결제 |
| 로그 및 감사 관리 | 모든 거래 기록을 추적하고 감사 | 금융 거래 감사 로그, 보안 이벤트 추적 |

## 마무리말

응용 아키텍처 산출물 정의는 프로젝트의 명확한 목표 설정과 진행 상황을 체계적으로 관리하기 위한 필수 요소입니다. 금융 및 공공기관 프로젝트에서는 데이터 무결성과 시스템 확장성 보장을 위한 구조적 설계가 필요하며, 이를 위한 정확한 산출물 정의가 프로젝트 성공의 기반이 됩니다.

## 시사점

|  |  |
| --- | --- |
| 시사점 | 설명 |
| 산출물 관리 최적화 | 명확한 문서 정의로 프로젝트의 목표와 진행 상황을 체계적으로 관리 |
| 데이터 무결성 및 보안 확보 | 데이터 모델링과 보안 계획을 통해 금융 거래의 신뢰성 향상 |
| 인터페이스 표준화 | 다양한 시스템 간 연동을 최적화하여 데이터 손실 방지 |