|  |  |
| --- | --- |
| **전사 EA 표준 Profile정의서** | |
| **사내 표준 정의 사업**  API 문사화툴  **Version 1.0**  Architect/SWAT | |
|  |  |
| SK주식회사 | |

**제.개정 이력**

|  |  |  |  |
| --- | --- | --- | --- |
| **버전** | **변경일자** | **제.개정 내용** | **작성자** |
| *0.3* | **2025-04-14** | *최초 등록* | *장우승* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Index / 목차

[1. 요구사항 정의 5](#_Toc198042083)

[1.1. 기능적 요구사항 (Functional Requirements) 5](#_Toc198042084)

[1.2. 비기능적 요구사항 (Non-functional Requirements) 5](#_Toc198042085)

[1.3. 기술 요구사항 (Technical Requirements) 5](#_Toc198042086)

[1.4. 마무리말 6](#_Toc198042087)

[1.5. 시사점 (표 형식) 6](#_Toc198042088)

[2. 아키텍처 비전 및 목표 정의 7](#_Toc198042089)

[2.1. 도입전안내말 7](#_Toc198042090)

[2.2. 아키텍처 목표 7](#_Toc198042091)

[2.3. 아키텍처 원칙 7](#_Toc198042092)

[2.4. 비즈니스 연계성 7](#_Toc198042093)

[2.5. 마무리말 8](#_Toc198042094)

[2.6. 시사점 (표 형식) 8](#_Toc198042095)

[3. 기술 스택 및 플랫폼 선정 9](#_Toc198042096)

[3.1. 도입전안내말 9](#_Toc198042097)

[3.2. 기슬 스택 및 플랫폼 선정 9](#_Toc198042098)

[3.3. 마무리말 10](#_Toc198042099)

[3.4. 시사점 (표 형식) 10](#_Toc198042100)

[4. 어플리케이션 계층 구조 설계 11](#_Toc198042101)

[4.1. 도입전안내말 11](#_Toc198042102)

[4.2. 어플리케애션 계층구조 설계 11](#_Toc198042103)

[4.3. 마무리말 12](#_Toc198042104)

[4.4. 시사점 (표 형식) 12](#_Toc198042105)

[5. 주요 설계 패턴 정의 13](#_Toc198042106)

[5.1. 5.1 도입전안내말 13](#_Toc198042107)

[5.2. 주요 설계 패턴 설계 13](#_Toc198042108)

[5.3. 마무리말 14](#_Toc198042109)

[5.4. 시사점 (표 형식) 14](#_Toc198042110)

[6. 보안 및 인증 설계 16](#_Toc198042111)

[6.1. 도입전안내말 16](#_Toc198042112)

[6.2. 보안 및 인증 설계 16](#_Toc198042113)

[6.3. 마무리말 17](#_Toc198042114)

[6.4. 시사점 (표 형식) 17](#_Toc198042115)

[7. 로깅 및 모니터링 체계 구축 19](#_Toc198042116)

[7.1. 도입전안내말 19](#_Toc198042117)

[7.2. 로깅 및 모니터링 체계 구축 19](#_Toc198042118)

[7.3. 마무리말 20](#_Toc198042119)

[7.4. 시사점 (표 형식) 20](#_Toc198042120)

[8. DevOps 및 자동화 전략 21](#_Toc198042121)

[8.1. 도입전안내말 21](#_Toc198042122)

[8.2. DevOps 및 자동화 전략 21](#_Toc198042123)

[8.3. 마무리말 22](#_Toc198042124)

[8.4. 시사점 (표 형식) 22](#_Toc198042125)

[9. 데이터 관리 전략 24](#_Toc198042126)

[9.1. 도입전안내말 24](#_Toc198042127)

[9.2. 데이터 관리 전략 24](#_Toc198042128)

[9.3. 마무리말 25](#_Toc198042129)

[9.4. 시사점 (표 형식) 25](#_Toc198042130)

[10. 통합 설계 및 연계 전략 27](#_Toc198042131)

[10.1. 도입전안내말 27](#_Toc198042132)

[10.2. 통합설계 및 연계 전략 27](#_Toc198042133)

[10.3. 마무리말 28](#_Toc198042134)

[10.4. 시사점 (표 형식) 29](#_Toc198042135)

[11. 장애 대응 및 복구 전략 30](#_Toc198042136)

[11.1. 도입전안내말 30](#_Toc198042137)

[11.2. 장애대응 및 복구 전략 30](#_Toc198042138)

[11.3. 마무리말 31](#_Toc198042139)

[11.4. 시사점 (표 형식) 31](#_Toc198042140)

[12. 운영관리 및 정책 수립 전략 32](#_Toc198042141)

[12.1. 도입전안내말 32](#_Toc198042142)

[12.2. 운영관리 및 정책 수립 전략 작성 32](#_Toc198042143)

[12.3. 마무리말 33](#_Toc198042144)

[12.4. 시사점 (표 형식) 33](#_Toc198042145)

[13. 성능·확장성·보안 최적화 요건 정리 34](#_Toc198042146)

[13.1. 도입전안내말 34](#_Toc198042147)

[13.2. 성능·확장성·보안 최적화 요건 작성 34](#_Toc198042148)

[13.3. 마무리말 35](#_Toc198042149)

[13.4. 시사점 (표 형식) 35](#_Toc198042150)

[14. 마이그레이션 전략 36](#_Toc198042151)

[14.1. 도입전안내말 36](#_Toc198042152)

[14.2. 마이그레이션 전략 36](#_Toc198042153)

[14.3. 마무리말 37](#_Toc198042154)

[14.4. 시사점 (표 형식) 37](#_Toc198042155)

[15. 종합 요약 및 리스크 대응 전략 38](#_Toc198042156)

[15.1. 도입전안내말 38](#_Toc198042157)

[15.2. 종합 요약 및 리스크 대응 전략 작성 38](#_Toc198042158)

[15.3. 주요 리스크 및 대응 전략 39](#_Toc198042159)

[15.4. 마무리말 39](#_Toc198042160)

[15.5. 시사점 (표 형식) 39](#_Toc198042161)

# ****요구사항 정의****

요구사항 정의는 어플리케이션 아키텍처 설계의 출발점이자 성공적인 시스템 구축을 위한 필수 단계입니다. 명확한 기능적 요구사항과 비기능적 요구사항의 정의를 통해 아키텍처의 목표와 방향성을 결정합니다. 특히 EA 기반의 구조화된 요구사항 정의는 개발, 배포, 운영 전반에 걸쳐 통합적이고 최적화된 아키텍처 설계를 가능하게 합니다.

## ****기능적 요구사항 (Functional Requirements)****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 1.1 기능적 요구사항 | 1.1.1 비즈니스 기능 정의 | 시스템이 수행해야 할 주요 비즈니스 프로세스 정의 (예: 거래 처리, 계정 관리, 데이터 조회) | 명확한 비즈니스 요구 정의는 설계 안정성과 기능 최적화를 이끈다. |
|  | 1.1.2 프로세스 흐름 정의 | 프로세스 간의 상호작용 및 데이터 흐름에 대한 상세 정의 | 프로세스 간 통신 최적화와 병목 현상 최소화가 가능하다. |
|  | 1.1.3 사용자 시나리오 분석 | 최종 사용자 관점에서의 사용 시나리오 정의 (예: 로그인, 결제 처리, 리포트 생성) | 사용자 중심의 아키텍처 설계가 가능하며, UX(User Experience) 최적화에 기여한다. |

### (실제) NH 농협지주 통합 데이터 플랫폼 구축사업 – 기능적 요구사항 목록

|  |  |  |  |
| --- | --- | --- | --- |
| **번호** | **기능 항목** | **설명** | **전략적 의미** |
| 1 | 계열사 데이터 수집 및 적재 | 계열사 정보계, 마케팅, e-마케팅 시스템에서 주기적으로 배치 데이터를 수집하고 ETL로 적재함 | **데이터 통합 허브 구축**, 다양한 소스의 표준화된 수집 구조로 확장성과 관리 용이성 확보 |
| 2 | 통합고객번호 채번 및 식별 관리 | 수집된 고객 정보를 기준으로 통합고객번호를 채번하고 중복 제거, 기준 식별체계 확립 | **고객 통합 뷰 제공**, 분석 정합성 확보, 캠페인 타겟 정확도 향상 |
| 3 | 데이터 정제 및 마트 구성 | 수집된 데이터를 정제하여 목적별 데이터 마트로 가공하고, 분석용 레이어로 적재 | **분석 활용 최적화**, 분석 효율성 및 속도 개선 |
| 4 | 데이터 품질 관리 및 메타 관리 | 메타 데이터 등록, 품질 진단, 관리 기능을 통한 데이터 신뢰도 확보 | **신뢰성 기반의 분석 환경 구축**, 데이터 활용률 향상 |
| 5 | 통합 데이터 API 제공 | 외부 시스템(e-마케팅, 캠페인, 포털 등)에 통합된 데이터 서비스를 REST API 형태로 제공 | **외부 연계 표준화**, 마이크로서비스 기반 연계 확장 용이 |
| 6 | 타깃 고객 추출 기능 | 필터 및 분석 로직 기반으로 마케팅 캠페인 타겟 고객을 추출하고 JSON 형태로 반환 | **맞춤형 마케팅 가능**, 캠페인 효율성과 반응률 향상 |
| 7 | 사용자별 권한 기반 접근제어 | 포털/OLAP/보고서 서비스에서 사용자 Role 기반 데이터 접근을 통제 | **데이터 보안 강화**, 개인정보보호 및 컴플라이언스 대응 |
| 8 | 포털 기반 메타 조회 및 다운로드 | BI 포털을 통해 메타 데이터 조회, 시각화, 보고서 생성 및 데이터 파일 다운로드 기능 제공 | **비즈니스 사용자 중심 인터페이스**, 셀프 서비스 분석 환경 지원 |
| 9 | 정형/비정형 리포트 출력 | OLAP 및 MSTR, Tableau 등을 통해 KPI, 성과지표 등 다양한 리포트 생성 및 시각화 기능 지원 | **분석 생산성 제고**, 마케팅·경영 판단을 위한 핵심 지표 제공 |
| 10 | 실시간 API 연계 처리 | 실시간 마케팅 요청, 고객 반응 정보 연계 등 실시간 처리 기반 RESTful API 구성 | **즉시성 대응 마케팅**, 시스템 간 실시간 연동 최적화 |
| 11 | 데이터 보관/파기 정책 관리 | 개인정보 유효기간 관리, 보관 주기 만료 후 자동 삭제 또는 파기 이력 기록 | **보안 및 컴플라이언스 대응**, ISMS-P 및 개인정보보호법 준수 |
| 12 | 실시간 동의 철회 연계 처리 | 계열사에서 철회된 동의 정보를 API 또는 배치로 수신하여 데이터 접근 및 사용을 제한 | **데이터 사용 통제**, 고객 동의 기반 분석 및 활용 체계 확립 |
| 13 | ML/DL 모델 분석 결과 연계 | 빅스퀘어에서 분석된 AI 모델 결과를 API 또는 배치 방식으로 제공하고 캠페인에 반영 | **AI 기반 지능형 분석 활용**, 자동화된 마케팅 고도화 |
| 14 | 통합 로그 및 이력 관리 | 데이터 수집, 가공, 제공 등의 각 단계별 처리 이력과 API 호출, 사용자 요청 기록 로그화 | **감사 대응 및 추적성 확보**, 장애 분석 및 최적화 기반 |

**(요약)**

|  |  |
| --- | --- |
| 핵심 포인트 | 전략적 시사점 |
| 고객중심 데이터 통합 기능 강화 | 통합 고객번호 기반의 마케팅 정합성과 고객 360도 뷰 확보 |
| 실시간/비동기 처리 기능 확보 | 민첩한 마케팅 대응, API 기반의 유연한 연계 아키텍처 가능 |
| AI 기반 예측 분석 및 자동화 반영 | 지능형 마케팅 실행 기반 마련, 사용자 행동 기반 개인화 강화 |
| 컴플라이언스 및 보안 설계 내재화 | ISMS-P, 개인정보보호법, GDPR 등 국내외 규제 대응의 아키텍처 내재화 |
| 사용자 중심 BI 포털 및 셀프 분석 지원 | 사용자 UX 최적화 및 자율적 데이터 활용 환경 조성 |

## ****비기능적 요구사항 (Non-functional Requirements)****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 1.2 비기능적 요구사항 | 1.2.1 성능 (Performance) | 목표 TPS 설정, 응답 시간, 처리 용량 정의 | 목표 TPS 10,000과 응답 시간 3초 이내를 충족시키기 위한 구조적 최적화가 필요하다. |
|  | 1.2.2 확장성 (Scalability) | 수평적 확장 (Horizontal) 및 수직적 확장 (Vertical) 전략 정의 | 미래의 확장 가능성을 고려한 설계로 유연한 시스템 대응이 가능하다. |
|  | 1.2.3 보안 (Security) | 인증(Authentication), 인가(Authorization),  전송 보안(TLS, SSL) 설정 | 금융권 시스템 특성상 강력한 인증 및 보안 정책이 필수적이다. |
|  | 1.2.4 가용성 (Availability) | 무중단 서비스, 장애 대응 시나리오, DR(Disaster Recovery) 계획 | 24/7 무중단 서비스와 장애 발생 시 복구 시간 최소화 전략이 필요하다. |

### ****(실제)**** NH 농협지주 통합 데이터 플랫폼 구축사업 – 비기능적 요구사항 정리

|  |  |  |  |
| --- | --- | --- | --- |
| 번호 | 항목 | 설명 | 전략적 의미 |
| 1 | 성능 (TPS & 응답시간) | API 최대 처리량 10,000TPS 이상, 응답 시간 3초 이내 보장 | **실시간 분석·제공 시스템 구성**, 금융권 업무 지연 최소화 |
| 2 | 확장성 (수평/수직) | 수평적 확장: Kubernetes 기반 자동 확장수직적 확장: VM 자원 유연 조정 | **멀티 계열사 확장 대응**, 신규 서비스 추가 시 구조 재설계 필요 없음 |
| 3 | 고가용성 (HA 구성) | 주요 시스템(ETL, API, 포털 등) 이중화 구성 및 장애 자동 전환 (Auto-Failover) | **무중단 분석·서비스 제공 기반**, 장애 시에도 운영 지속 가능 |
| 4 | 재해복구 (DR) | RPO 5분, RTO 30분 이내의 DR 설계 및 원격지 백업 환경 구축 | **금융권 재해 대비 요건 충족**, 업무 중단 시 리스크 최소화 |
| 5 | 보안 정책 준수 | TLS 1.3 기반 전송 구간 암호화, AES-256 기반 저장 암호화, API 인증(OAuth2, JWT) 적용 | **개인정보 보호법, 전자금융감독규정 등 금융 보안 규정 준수** |
| 6 | 사용자 접근 제어 | Keycloak 기반 RBAC(역할기반 접근 제어), AD 연동, MFA 다중 인증 | **내부 사용자별 접근 권한 분리**, 비인가 접근 방지 |
| 7 | 감사 및 로깅 | 모든 데이터 접근 및 처리 로그 이력 저장, Tamper-proof 로그 보관 | **감사 추적성 확보**, ISMS-P, 금융보안원 준수 |
| 8 | 배치 처리 안정성 | 야간/정기 배치의 시간 내 완료 보장, 배치 실패시 알림 및 자동 재처리 | **운영 안정성 확보**, 데이터 적재 실패 방지 및 SLA 관리 |
| 9 | API 호출 이력 관리 | 모든 외부/내부 API 호출에 대해 이력 관리, 응답/오류 로그 분석 지원 | **API 안정성 및 추적성 확보**, 연계 장애 진단 용이 |
| 10 | 유지보수성 | 코드 표준화, GitOps 기반 운영, 배포 자동화, CI/CD 적용 | **신속한 장애 대응 및 기능 개선**, 유지관리 비용 절감 |
| 11 | SLA/SLO 모니터링 | API/포털/DB 등 핵심 구성 요소에 대해 SLA/SLO 모니터링 지표 수립 및 경고 체계 구성 | **운영 품질 보장**, 성능 하락 또는 지연 시 사전 경고 가능 |
| 12 | 멀티테넌시 지원 | 계열사별 사용자 권한, 마트 구분, 메타 데이터 분리 등 멀티테넌시 구조 설계 | **계열사 간 데이터 보안 확보**, 동일 플랫폼 상에서의 유연한 분리 운영 가능 |
| 13 | 인터페이스 확장성 | Kafka, REST, JDBC, MFT 등 다양한 연계방식의 동시 운영 및 표준화 | **다양한 외부 시스템과의 연계 유연성 확보**, EAI → API 전환에도 무리 없는 구조 확보 |
| 14 | 모니터링 및 장애 대응 | Prometheus+Grafana 기반 리소스/트랜잭션/지연 시간 모니터링, 실시간 알림/Slack 연동 등 | **운영 리스크 사전 감지**, SRE 체계 기반 운영 관리 |

요약

|  |  |
| --- | --- |
| **관점** | **주요 시사점** |
| 성능/가용성 | 금융권 트랜잭션 특성상 고성능 + 무중단 처리가 핵심. TPS 보장, Auto Failover 필수 |
| 보안 | 인증/인가/암호화 기반 보안 설계와 로깅/감사 기능 내재화로 **규제 준수 + 신뢰성 확보** |
| 확장성 | 계열사 추가, 신규 마케팅 채널 증가 등 미래 확장을 고려한 **유연한 아키텍처 구조** |
| 운영 안정성 | **배치 실패 자동복구, 장애 사전 감지, 자동 알림** 등 IT 운영 효율성을 극대화해야 함 |
| 유지보수/개선 대응 | DevOps와 GitOps 기반의 자동화된 배포 및 테스트 체계 구축으로 **개선 속도 확보** |

## ****기술 요구사항 (Technical Requirements)****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 1.3 기술 요구사항 | 1.3.1 플랫폼 및 프레임워크 | 프론트엔드 (React, Vue), 백엔드 (Spring Boot, Node.js), API Gateway (Zuul, Kong) | 최신 기술 스택을 활용한 안정적이고 확장 가능한 아키텍처 구현이 가능하다. |
|  | 1.3.2 데이터베이스 | RDBMS (Oracle, MySQL), NoSQL (MongoDB, Cassandra), In-memory (Redis) | 데이터 특성에 맞춘 최적화된 저장소 설계를 통해 트랜잭션 처리 성능을 극대화할 수 있다. |
|  | 1.3.3 클라우드 및 컨테이너 | Kubernetes, Docker, OpenShift | 클라우드 네이티브 환경에서의 유연한 확장 및 관리가 용이하다. |

### ****(실제)**** NH 농협지주 통합 데이터 플랫폼 구축사업 – 기술적 요구사항 정리

|  |  |  |  |
| --- | --- | --- | --- |
| **번호** | **항목** | **설명** | **전략적 의미** |
| 1 | 프론트엔드 프레임워크 | React 기반 단일 페이지 UI 구성 (또는 Vue.js 선택 가능) | **빠른 렌더링, 사용자 UX 최적화**, 마이크로 프론트엔드 확장 가능 |
| 2 | 백엔드 프레임워크 | Spring Boot 기반의 API 중심 서버 구조 | **MSA 설계에 적합**, 모듈화 및 배포 편의성 우수 |
| 3 | API 게이트웨이 | Zuul, Kong, Apigee 등 게이트웨이 도입 | **트래픽 제어, 인증 통합, 요청 라우팅 제어**, 외부 시스템과의 통합 용이 |
| 4 | 데이터베이스 구조 | Oracle HEX(RDBMS), MySQL, Redis(In-Memory), MongoDB(NoSQL) 혼합 사용 | **정형/비정형 데이터 분리 저장 및 조회 성능 최적화** |
| 5 | 데이터 수집 플랫폼 | DataStage 기반 정제/적재 수행 | **대량 배치처리 안정화 및 전처리 체계 내재화** |
| 6 | 컨테이너 오케스트레이션 | Kubernetes 기반 운영 환경 | **무중단 운영, Auto Scaling, 서비스 이중화에 최적** |
| 7 | CI/CD 파이프라인 | Jenkins, ArgoCD, GitLab CI 기반 자동 빌드/배포 구성 | **지속적 통합/배포 구조 확보, 운영 효율성 극대화** |
| 8 | 메시지 브로커 | Kafka, RabbitMQ를 통한 비동기 이벤트 처리 | **실시간 분석/반응 구조 대응, 이벤트 기반 마케팅 시나리오 적용 가능** |
| 9 | 로그 수집 및 분석 | ELK Stack(Elasticsearch, Logstash, Kibana), Fluentd 기반 통합 로그 구성 | **실시간 로그 분석 및 이상 징후 탐지 체계 확보** |
| 10 | 모니터링 도구 | Prometheus + Grafana, Zabbix 기반 리소스/응답속도 모니터링 | **운영 안정성 확보 및 SLA 준수 상태 실시간 시각화** |
| 11 | 인증 및 접근 제어 | OAuth2, JWT, Keycloak, SSO 통합 인증 체계 | **계열사별 계정 권한 분리 및 사용자 인증 보안성 확보** |
| 12 | 멀티 리전 지원 | 향후 클라우드 기반 멀티리전 또는 DR 지역 확장 대응 | **지리적 분산 대비 및 재해 복구 시나리오 유연 대응 가능** |
| 13 | 오픈소스 및 상용 혼합 전략 | 상용 솔루션(Oracle, DataStage) + 오픈소스(Spring, Kafka 등) 병행 활용 | **TCO 최적화 및 기술 자유도 확보**, 기술 생태계 중심 기술 스택 구성 |

✅ 시사점 요약

|  |  |  |
| --- | --- | --- |
| **관점** |  | **주요 시사점** |
| 최신 기술 스택 |  | React, Spring Boot, Kafka, Kubernetes 등 **검증된 최신 기술 도입으로 안정성과 확장성 확보** |
| 아키텍처 최적화 |  | MSA, 컨테이너화, CI/CD 자동화 등 **현대적 DevOps 기반 아키텍처 완성** |
| 데이터 중심 설계 |  | 정형 + 비정형 DB, 실시간 + 배치 분석 연계 기반 설계로 **데이터 주도 비즈니스 대응력 강화** |
| 보안 및 접근제어 |  | OAuth2/SSO/Keycloak 등 계정 보안체계 적용으로 **금융 보안 규정 충족 및 내부통제 강화** |

## ****마무리말****

요구사항 정의는 어플리케이션 아키텍처 설계의 기초를 형성합니다. 명확하게 정의된 기능적 요구사항은 시스템의 기능적 완성도를 높이고, 비기능적 요구사항은 성능, 확장성, 보안성을 확보합니다. 또한, 기술 요구사항은 최신 플랫폼과 클라우드 네이티브 환경을 바탕으로 최적화된 개발 및 배포가 가능하게 합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **기능적 요구사항** | 비즈니스 프로세스 최적화와 사용자 중심의 서비스 제공이 가능하다. |
| **비기능적 요구사항** | 성능 최적화, 확장성, 보안성 확보를 통해 고가용성 서비스를 실현한다. |
| **기술 요구사항** | 최신 클라우드 플랫폼과 컨테이너 기반 아키텍처를 통해 유연하고 확장 가능한 구조를 만든다. |
| **플랫폼 연계성** | 멀티 클라우드 및 온프레미스 환경에서의 유연한 확장과 API 통합이 가능하다. |

# ****아키텍처 비전 및 목표 정의****

## ****도입전안내말****

아키텍처 비전 및 목표 정의는 어플리케이션 아키텍처 설계에서 목표 성능과 운영 전략을 명확하게 설정하는 단계입니다. 이는 EA(Enterprise Architecture) 관점에서 기술적 로드맵을 구성하고, 목표 TPS, 응답 시간, 가용성, 확장성을 포함하여 최종적으로 목표하는 시스템의 Blueprint를 마련합니다.  
이를 통해 개발 단계에서의 리스크 최소화, 시스템 안정성 확보, 장애 대응 계획 수립이 가능해집니다.

## ****아키텍처 목표****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 2.1 아키텍처 목표 | 2.1.1 목표 TPS 설정 | 시스템이 초당 처리해야 하는 트랜잭션 수를 정의 (예: 10,000 TPS 목표) | 고부하 트랜잭션에도 안정적인 처리 성능을 확보하기 위한 기준 설정 |
|  | 2.1.2 응답 시간 목표 설정 | 사용자 요청에 대한 응답 시간을 정의 (예: 3초 이내) | 빠른 응답 시간으로 사용자 경험 향상 및 시스템 신뢰도 증가 |
|  | 2.1.3 장애 대응 및 복구 계획 | 장애 발생 시 복구 시간 (RTO)와 데이터 손실 허용 시간 (RPO)을 정의 | 무중단 서비스를 위한 재해 복구(Disaster Recovery) 및 이중화 구조 필요 |
|  | 2.1.4 데이터 처리 용량 설정 | 일간, 월간, 연간 데이터 처리 용량 분석 및 설계 | 예상되는 데이터 증가에 대비한 저장소 확장성 및 I/O 최적화 전략 수립 |

### ****(실제)**** NH 농협지주 통합 데이터 플랫폼 구축사업 – 아키텍처 목표

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **전략적 의미** |
| **2.1 아키텍처 목표** | **2.1.1 목표 TPS 설정** | 시스템이 초당 처리해야 하는 트랜잭션 수를 정의 → 최소 10,000 TPS 이상 처리 가능 구조 설계 | 고부하 발생 시 병목 없이 **실시간 데이터 연계 및 처리 가능 구조 확보** |
|  | **2.1.2 응답 시간 목표** | 사용자 요청에 대한 응답 시간 기준 정의 → 대부분의 요청은 3초 이내 응답 완료 목표 | 사용자 만족도 향상 및 대내외 API 호출 안정성을 확보하고, **UX 최적화** |
|  | **2.1.3 장애 대응/복구 계획** | 장애 발생 시 RTO 10분 이내, RPO 5분 이내로 정의 → Auto Healing 및 이중화 구조 설계 | 무중단 운영 보장, 재해 복구(Disaster Recovery) 구성을 통해 **서비스 연속성 보장** |
|  | **2.1.4 데이터 처리 용량** | 일간 3~5TB, 연간 1PB 이상 데이터 처리 대응 구조 → 수집-저장-분석 전 구간 용량 설계 | **대용량 수집/정제/분석 파이프라인 대응**, I/O 병목 해소를 위한 ETL 병렬 처리 구조 필요 |
|  | **2.1.5 시스템 확장성** | 수평적 확장(Horizontal Scaling) 중심 설계 → Kubernetes 기반 Auto Scaling 도입 | 향후 계열사 및 사용자 증가에도 **서비스 무중단 확장 대응** |
|  | **2.1.6 통합 API 응답 구조** | 통합 데이터 AP 서버에서 처리하는 응답 처리 구조 설계 → RESTful JSON 기반 경량화 설계 | 다양한 채널(포털, 캠페인, OLAP)에 대해 **표준 API 기반 통합 응답 처리 체계 확보** |
|  | **2.1.7 실시간 스트리밍 처리** | Kafka 기반 실시간 데이터 스트림 설계 → OLTP/OLAP/마케팅 실시간 이벤트 처리 최적화 | 실시간 반응형 마케팅/분석 수행을 위한 **Event-driven Architecture** 구축 |
|  | **2.1.8 데이터 품질/정합성 보장** | 통합고객번호 채번 및 메타정보 기반 품질관리 설계 → 정제/통합/분석 전 과정 품질검증 체계 | **정제-마스킹-품질-보안 처리 내재화**, 데이터 신뢰도 및 정확도 확보 |
|  | **2.1.9 통합 인증 및 보안** | Keycloak 기반 OAuth2/JWT 인증, TLS 연계 설계 → 계열사 사용자별 접근통제 설계 포함 | 금융 보안 규정(ISMS-P, 금융보안원 표준) 준수를 위한 **다계층 보안 구조 내재화** |
|  | **2.1.10 유연한 연계 구조** | MCA, EAI, MFT, API 등 이기종 연계 통합 설계 → 내외부 인터페이스 동시 대응 | 계열사·타 시스템과의 연계 유연성을 극대화하며, **확장 가능한 연계 인터페이스 구조 확보** |

✅ 시사점 요약

|  |  |
| --- | --- |
| **관점** | **주요 시사점** |
| 성능 | TPS 10,000 이상, 3초 이내 응답시간 기준 확보로 **대규모 트랜잭션 안정성 확보** |
| 안정성 | Auto Healing, RTO/RPO 기준 준수로 **재해 대응 및 무중단 운영 가능** |
| 확장성 | PaaS 기반 Kubernetes 구조로 **수평 확장성과 무중단 배포체계 내재화** |
| 실시간성 | Kafka 기반 스트리밍 및 이벤트 기반 구조로 **반응형 마케팅/분석 체계 구현 가능** |
| 보안 | OAuth2/JWT 기반 인증, TLS 연계로 **금융 보안 규정 준수 및 데이터 유출 방지 체계 구축** |
| 연계 유연성 | MCA, EAI, API 등 복수 연계 채널 통합 구조로 **타 계열사 및 외부 시스템 연계 효율화** |

## ****아키텍처 원칙****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 2.2 아키텍처 원칙 | 2.2.1 Stateless 설계 | 세션 정보를 서버에 저장하지 않고 분리하여 관리 (예: Redis, JWT 사용) | 확장성 증대 및 로드 밸런싱 최적화, 장애 발생 시 빠른 전환 가능 |
|  | 2.2.2 분산 처리 구조 | 대규모 요청을 처리하기 위해 마이크로서비스 단위로 분산 처리 | 시스템 부하 분산 및 성능 최적화, 병목 현상 제거 |
|  | 2.2.3 마이크로서비스 분리 | 업무 도메인 별로 독립된 서비스로 관리하여 결합도 최소화 | 독립 배포, 독립 확장 가능, 장애 시 서비스 영향 최소화 |
|  | 2.2.4 Event-driven Architecture | 비동기 이벤트 기반의 통신 방식 적용 (예: Kafka, RabbitMQ) | 이벤트 발생 시 즉각적인 반응 처리, 실시간 데이터 처리 최적화 |
|  | 2.2.5 Saga Pattern | 분산 트랜잭션 관리를 위한 보상 트랜잭션 처리 구조 | 금융거래 및 다중 시스템 간의 복합 트랜잭션 처리 시 데이터 일관성 유지 |

### ****(실제)**** NH 농협지주 통합 데이터 플랫폼 구축사업 – 아키텍처 원칙

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **전략적 의미** |
| **2.3 아키텍처 원칙** | **2.3.1 Stateless 설계** | 세션 정보를 서버 내부에 저장하지 않고 별도 저장소(예: Redis) 또는 JWT 토큰 기반 외부 관리 방식 적용 | 확장성 및 이식성 강화, 로드 밸런싱에 최적화, 서버 무중단 재시작 및 마이그레이션 용이 |
|  | **2.3.2 분산 처리 구조** | 대규모 요청 처리를 위해 마이크로서비스 단위의 병렬 처리와 메시지 큐 기반 비동기 처리 도입 | 시스템 병목 제거, 수평 확장 가능, 고가용성 유지, 장애 격리 및 탄력적 처리 구조 구현 |
|  | **2.3.3 마이크로서비스 분리** | 업무 도메인별로 독립된 서비스 구성 (예: 계정, 마케팅, 분석, 통합고객번호 등), 데이터베이스 분리 운영 | 결합도 최소화, 독립적 배포 및 확장성 확보, 장애 영향 범위 최소화로 유지보수 용이 |
|  | **2.3.4 Event-driven Architecture** | Kafka, RabbitMQ 등을 활용한 비동기 이벤트 기반 설계 적용 | 실시간 처리 최적화, 이벤트 소싱 기반 데이터 흐름 추적 가능, 마케팅·알림·행동 분석 시스템에 필수 구조 |
|  | **2.3.5 Saga Pattern** | 마이크로서비스 간 분산 트랜잭션 보상을 위한 패턴 도입, 로컬 트랜잭션 기반의 상태 보상 처리 | 데이터 일관성 확보, 복합 거래 시스템에서 오류 발생 시 롤백을 통한 복원력 확보, 금융 트랜잭션 신뢰도 강화 |
|  | **2.3.6 API 우선 설계(First-class API)** | 모든 데이터 처리와 서비스 호출을 API 기반으로 구성, 내부 서비스 호출도 API 게이트웨이 통해 연결 | API 문서화, 연계 표준화, 인증 및 추적이 용이한 구조, 계열사와 통합된 연계 아키텍처 표준화 |
|  | **2.3.7 보안 중심 설계(Security by Design)** | OAuth2, TLS 암호화, SSO, JWT 등의 보안 원칙을 아키텍처에 선내재화 | 금융기관 보안 준수(ISMS-P, 금융보안원 등), 개인정보보호법 대응, 외부 공격 및 데이터 유출 방지 |
|  | **2.3.8 유연한 연계 구조(Interoperability)** | REST API, MFT, JDBC, Kafka, EAI 등 다양한 연계 수단을 병행 지원 | 내외부 다양한 시스템 및 계열사와 연계 가능, 이기종 환경에서도 유연한 통합 및 연계성 확보 |
|  | **2.3.9 표준 기반 기술 채택** | Spring Boot, Kubernetes, Kafka, DataStage, Oracle 등 업계 검증된 기술 스택 기반으로 설계 | 유지보수성과 안정성 확보, 인력 확보 용이, 기술 생태계 활용 극대화, 벤더 종속도 최소화 |
|  | **2.3.10 관측 가능성(Observability) 내재화** | 로그 수집, 메트릭, 트레이싱 구성으로 전체 시스템 상태를 실시간으로 관측 가능하게 설계 | 성능 병목 식별, 장애 원인 신속 분석, SLA/SLO 대응, 운영 자동화 및 AIOps 연계 가능 |

✅ 시사점 요약

|  |  |
| --- | --- |
| **관점** | **전략적 시사점** |
| 확장성 | Stateless, 마이크로서비스, API 기반으로 수평 확장 및 무중단 배포 가능 |
| 실시간성 | Kafka 기반 이벤트 스트림 및 분산 처리 구조로 즉각적 반응형 마케팅/분석 환경 구현 가능 |
| 보안성 | OAuth2 + TLS 기반 보안 설계로 금융권 보안 및 인증 정책 준수 가능 |
| 복원력/장애대응성 | Saga Pattern, Circuit Breaker 등 내재화로 장애 발생 시 자동 회복 및 데이터 일관성 유지 가능 |
| 연계 표준화 | 다양한 연계 기술(MCA, API, JDBC, MFT) 병행하여 이기종 계열사와의 통합 유연성 확보 |
| 운영 효율성 | Logging, Monitoring, 트레이싱 전략을 아키텍처에 내재화하여 운영 자동화 및 장애 사전 대응 가능 |

## ****비즈니스 연계성****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 2.3 비즈니스 연계성 | 2.3.1 주요 프로세스 모델링 | 마케팅 시스템, 거래 처리 시스템, 권한 관리 시스템 간의 통신 모델 정의 | 프로세스 간 연계 최적화, 중복 처리 방지 및 통신 비용 절감 |
|  | 2.3.2 MSA 적용 범위 | 마케팅 시스템과 대용량 트랜잭션 처리 시스템은 MSA 구조로 설계 | 확장성 및 유연성을 극대화하여 변화 대응력 강화 |
|  | 2.3.3 Monolithic 적용 범위 | 핵심 트랜잭션 시스템은 Monolithic 구조로 통합 관리 | 일관성 유지 및 중앙 집중화된 관리가 용이하며, 레거시 시스템과의 호환성 유지 |

### ****(실제)**** NH 농협지주 통합 데이터 플랫폼 구축사업 – 비즈니스 연계성

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **전략적 의미** |
| **2.4 비즈니스 연계성** | **2.4.1 주요 프로세스 모델링** | 마케팅 시스템, 거래 처리 시스템, 권한 관리 시스템 간의 데이터 흐름 및 통신 관계를 BPMN 기반으로 모델링하고, 데이터 요청/응답 흐름을 정형화 | 통합 고객 번호 기준 전사 데이터 활용 가능, 중복 저장/처리 방지, RESTful 기반 연계 통신 비용 절감, 추적성 확보 |
|  | **2.4.2 MSA 적용 범위 정의** | 대용량 실시간 처리, 유저별 이벤트 기반 반응형 마케팅, 머신러닝 결과 분산 배포 등 확장성 요구 영역은 마이크로서비스 아키텍처(MSA)로 구성 | 비동기 메시징(Kafka), API 게이트웨이, 이벤트 소싱 등을 통해 유연성 확보, 확장 가능성과 장애 격리 구조 확보 |
|  | **2.4.3 Monolithic 적용 범위 정의** | 보안 민감성이 높고, 중앙 집중적 통제 및 규제 준수가 요구되는 계정계 및 핵심 원장 기반의 배치 업무는 Monolithic 구조로 관리 | 금융 핵심 트랜잭션 일관성 유지, 처리 신뢰도 확보, 레거시 시스템과의 안정적 연계 및 유지보수 효율성 강화 |
|  | **2.4.4 하이브리드 구조 설계** | Monolithic 영역과 MSA 영역 간에는 전문 헤더 구조 및 통합 API Gateway를 통한 하이브리드 연계 전략 적용 | 양쪽 아키텍처 간의 무결성 유지, 계층 간 유연한 통신 가능, 전환 비용 최소화와 점진적 MSA 확장 기반 제공 |
|  | **2.4.5 통합 인증·권한 구조 연계** | 사용자 인증은 Keycloak 기반 OAuth2 방식으로 통합 관리, 각 도메인 서비스는 RBAC(Role Based Access Control) 기반으로 통제 설계 | 계열사/채널별 인증 일원화, 권한 충돌 최소화, 분산 서비스 간 통합된 접근 정책 구성 가능 |
|  | **2.4.6 메타데이터 및 품질 연계** | 통합 데이터 플랫폼 내에서 메타데이터 및 품질지표 관리는 통합포털에서 중심 관리하며, 연계 시스템은 이를 조회하는 방식으로 구성 | 품질관리 책임 이원화 방지, 데이터 신뢰도 향상, 사용자 요청에 대한 추적과 품질 감사 지원 가능 |
|  | **2.4.7 마케팅 → ETL → 분석 흐름 정합성 유지** | 마케팅 시나리오 등록 → 대상 추출 → 반응 정보 수집 → 정제 및 적재 → OLAP 분석 → 리포트 제공 흐름 간 데이터 정합성 및 처리 상태 연계 보장 | 전사적 마케팅/분석 활동의 자동화 및 실시간성 보장, 분석 결과의 정확도 및 활용도 제고 |
|  | **2.4.8 외부 채널 연계 모델링** | 모바일뱅킹/인터넷뱅킹/통합단말 등 채널 계층에서 발생하는 사용자 이벤트는 통합 데이터 AP를 통해 실시간으로 처리 | 채널 간 이벤트 통합 대응, 마케팅 및 분석 피드백 루프 형성, 고객 경험 기반 캠페인 전개 가능 |
|  | **2.4.9 연계 프로토콜 다양성 설계** | HTTP/REST, FILE/MFT, JDBC, Kafka 등 다양한 연계 방식을 각각의 비즈니스 특성에 맞게 적용 | 계열사 레거시 연계 유지 및 API 우선 구조 병행, 유연한 연계 가능성 확보 및 배포 경량화 가능 |
|  | **2.4.10 서비스 흐름 통합 모니터링** | 통합 데이터 AP 계층에서 발생하는 주요 흐름은 로그 수집, APM(Application Performance Management), 분산 트레이싱 기반으로 시각화 관리 | 장애 예측, 흐름 병목 식별, SLA 기반 통합 연계 품질 모니터링 가능 |

### (실제) ✅ 연계 구조 도식 (텍스트 기반 흐름)

[계열사 Legacy 시스템]

└──▶ (배치/MFT, 실시간/API) ──▶ [대외 MCA/FOS]

└──▶ [통합데이터 ETL] ──▶ [통합데이터 DB]

├─▶ [통합데이터 AP]

│ ├─▶ 마케팅 시스템

│ ├─▶ OLAP 시스템

│ └─▶ 통합 데이터 포털

└─▶ [BI 포털 DB, OLAP 조회, UMS 등]

[1] 계열사 Legacy 시스템 (은행, 카드, 증권, 보험 등)

└─(FILE/배치)─▶ [대외 MCA]

└─▶ [FOS Storage]

└─▶ [통합 데이터 ETL 서버]

└─(JDBC)─▶ [통합 데이터 DB 서버]

[2] 빅스퀘어 (은행 빅데이터 분석 환경)

├─(FILE/배치)─▶ [통합 데이터 ETL 서버]

├─(Kafka/실시간)─▶ [마케팅 허브], [e마케팅 시스템]

└─(JDBC)─▶ [OLAP 시스템]

[3] 마케팅 시스템

├─(HTTP/API)─▶ [통합 데이터 AP 서버]

│ └─(JDBC)─▶ [통합 데이터 DB 서버]

├─(FILE/배치)─▶ [통합 데이터 ETL 서버]

└─(HTTP/API)─▶ [UMS 시스템], [올원뱅크], [BI 포털 Proxy WEB]

[4] E-마케팅 시스템

├─(FILE/배치)─▶ [통합 데이터 ETL 서버]

└─(HTTP/API)─▶ [통합 데이터 AP 서버]

[5] 통합 데이터 포털

├─(JDBC)─▶ [통합 데이터 DB 서버]

├─(HTTP/API)─▶ [통합 데이터 AP 서버]

└─(연계)─▶ [마케팅 허브], [e마케팅 시스템]

[6] OLAP 시스템

├─(JDBC)─▶ [통합 데이터 DB 서버]

├─(FILE/배치)─▶ [마케팅 시스템], [e마케팅 시스템], [마케팅 허브]

└─(REST API)─▶ [통합 데이터 AP 서버]

[7] 통합 데이터 AP 서버

├─(REST API + JWT + OAuth2.0)─▶ [통합 데이터 포털], [마케팅 시스템], [e마케팅], [외부 계열사]

└─(JDBC)─▶ [통합 데이터 DB 서버]

[8] 통합 데이터 ETL 서버

├─(MFT + JDBC)─▶ [통합 데이터 DB 서버]

├─(FILE/MFT)─▶ [FOS Storage], [마케팅 허브], [e마케팅]

└─(자동 스케줄링)─▶ [ControlM Client], [DataStage Engine]

[9] BI 포털 시스템

├─(JDBC)─▶ [통합 데이터 DB 서버]

├─(REST API)─▶ [통합 데이터 AP 서버]

└─(연계 시각화)─▶ [OLAP UI], [보고서 시스템]

[10] FOS (파일 전송 시스템)

└─(MFT)─▶ [통합 데이터 ETL 서버], [FOS Storage]

[11] 외부 접속 채널 (통합 단말, 인터넷뱅킹, 모바일뱅킹)

└─(HTTPS)─▶ [BI 포털 Proxy WEB]

└─(REST API)─▶ [통합 데이터 포털]

[12] 기타 시스템

├─ [UMS 시스템] ←(API)← [마케팅 시스템], [마케팅 HUB]

├─ [SSO 시스템] ←(SSO 인증)← [통합 데이터 포털]

└─ [메타관리 시스템] ←(관리용 연계)← [ETL/BI포털]

#### (실제) 주요 연계 방식 정리

|  |  |  |
| --- | --- | --- |
| **연계 구분** | **방식** | **사용 예시** |
| 파일 연계 | MFT, FOS | 계열사 → ETL, 마케팅 시스템 → ETL |
| 실시간 API 연계 | REST API + JWT/OAuth2 | 통합 데이터 AP ↔ 통합 포털, 마케팅 시스템, 외부 채널 등 |
| DB 직접 연계 | JDBC | ETL ↔ DB, 포털 ↔ DB, OLAP ↔ DB 등 |
| 이벤트 기반 메시징 | Kafka | 빅스퀘어 → 마케팅 허브, e마케팅 |
| 배치 스케줄링 | ControlM + DataStage | 데이터 수집/적재/마스킹/이력/파기 |

### (실제) ✅ 시사점 요약

| **영역** | **적용 시사점** |
| --- | --- |
| 구조 전략 | MSA와 Monolithic의 병행 적용을 통한 안정성과 유연성 동시 확보 |
| 연계 효율성 | BPM 기반 프로세스 모델링과 통합 API 게이트웨이로 연계 표준화 |
| 유지보수 및 확장 | 도메인 기반 분리 설계 및 계층별 역할 명확화를 통해 장기적 확장성 확보 |
| 규제 대응 | 인증, 권한, 보안, 로그 관리 등 모든 연계 요소가 금융권 정책 준수에 부합하도록 설계됨 |
| 데이터 활용 | 분석→시나리오 적용→성과→재분석으로 이어지는 마케팅/분석 연계 루프 완성 |

## ****마무리말****

아키텍처 비전 및 목표 정의는 시스템의 최종적인 청사진을 그리는 중요한 단계입니다. 설정된 목표 TPS와 응답 시간, 장애 대응 전략을 바탕으로 EA 기반의 최적화된 구조를 설계할 수 있으며, MSA와 Monolithic의 적절한 조합을 통해 안정성과 유연성을 동시에 확보합니다. 이를 통해 시스템의 성능 최적화와 미래 확장을 고려한 설계가 가능해집니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **아키텍처 목표 설정** | 명확한 목표 설정은 개발 단계에서의 성능 최적화와 리스크 감소를 가능하게 한다. |
| **아키텍처 원칙 수립** | Stateless 설계와 분산 처리 구조는 시스템 확장성과 복구 능력을 극대화한다. |
| **비즈니스 연계성 정의** | MSA와 Monolithic의 혼합 모델은 금융권에서의 안정성과 확장성을 동시에 확보할 수 있다. |
| **Event-driven 구조** | 실시간 이벤트 처리를 통해 빠른 트랜잭션 처리가 가능하며, 분산 환경에서도 일관성을 유지한다. |
| **Saga Pattern 활용** | 분산 트랜잭션 관리가 필요할 때 데이터의 일관성 및 신뢰성을 보장할 수 있다. |

# ****기술 스택 및 플랫폼 선정****

## ****도입전안내말****

기술 스택 및 플랫폼 선정은 어플리케이션 아키텍처의 핵심을 이루는 요소로, 성능, 확장성, 유지보수성을 결정하는 중요한 단계입니다. EA(Enterprise Architecture) 관점에서 최적의 기술 조합은 운영 비용을 절감하고, 개발 생산성을 극대화하며, 장애 대응 능력을 향상시킵니다.  
프론트엔드, 백엔드, API Gateway, 데이터베이스, 클라우드 및 컨테이너 플랫폼을 적절히 조합함으로써 신뢰성 있고 유연한 아키텍처를 설계합니다.

## ****기슬 스택 및 플랫폼 선정****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| 3.1 프론트엔드 | 3.1.1 React | 사용자 인터페이스를 구성하며, SPA(Single Page Application) 구조로 빠른 렌더링을 지원 | 유지보수성과 확장성이 높으며, 개발 속도가 빠르다. |
|  | 3.1.2 Vue | 경량 프레임워크로 빠르고 유연한 UI 개발이 가능하며, 컴포넌트 재사용성을 극대화 | 작은 프로젝트나 대시보드에 최적화된 성능 제공 |
|  | 3.1.3 Angular | 대규모 엔터프라이즈 애플리케이션에 적합하며, 강력한 양방향 데이터 바인딩 지원 | 구조적 설계와 유지보수가 용이하며, MVC(Model-View-Controller) 패턴에 최적화됨 |
| 3.2 백엔드 | 3.2.1 Spring Boot | 엔터프라이즈급 애플리케이션 개발에 최적화된 프레임워크, 빠른 개발 및 배포가 가능 | MSA 구조에서 서비스 간 통신 최적화, RESTful API 개발 용이 |
|  | 3.2.2 Node.js | 비동기 I/O 모델을 지원하여 실시간 데이터 처리에 강점이 있음 | 경량화된 서비스 개발 및 높은 처리 성능을 요구하는 서비스에 적합 |
|  | 3.2.3 .NET Core | 마이크로서비스 아키텍처에서 강력한 성능과 보안성을 제공하며, Windows 및 Linux 환경에서 모두 실행 가능 | 금융 거래 시스템과 같은 보안 요구가 높은 환경에 최적화됨 |
| 3.3 API Gateway | 3.3.1 Zuul | Netflix OSS 기반의 API Gateway로, 트래픽 라우팅과 부하 분산 처리 | 마이크로서비스 간 통신 최적화 및 인증 처리 가능 |
|  | 3.3.2 Kong | 클라우드 네이티브 API Gateway로서 확장성 높은 서비스 디스커버리 및 보안 정책 적용 가능 | MSA 환경에서의 API 통합 및 정책 관리 최적화 |
|  | 3.3.3 Nginx | 경량의 HTTP 서버로 API Gateway 역할을 수행하며, Reverse Proxy와 Load Balancer로 활용 | 높은 처리 성능과 가벼운 메모리 사용으로 트래픽이 많은 환경에 최적화됨 |
| 3.4 데이터베이스 | 3.4.1 Oracle | 금융권에서 널리 사용되는 RDBMS로 안정성과 데이터 무결성이 보장됨 | 대규모 트랜잭션 처리에 최적화되며, ACID(Atomicity, Consistency, Isolation, Durability) 준수 |
|  | 3.4.2 MySQL | 오픈소스 관계형 데이터베이스로 높은 확장성과 커뮤니티 지원 | 중소형 애플리케이션에서 널리 사용되며, 빠른 쿼리 처리 속도를 제공 |
|  | 3.4.3 PostgreSQL | 오픈소스 기반의 RDBMS로 GIS 데이터 처리와 JSON 타입 지원 | 확장성과 SQL 표준을 준수하며, 대규모 데이터 처리에 최적화됨 |
|  | 3.4.4 MongoDB | NoSQL DB로 빠른 읽기/쓰기 성능을 제공하며 스키마리스(Schema-less) 구조로 유연한 데이터 관리 가능 | 대용량 데이터 처리와 빠른 개발 주기에 최적화됨 |
| 3.5 클라우드 및 컨테이너 | 3.5.1 Kubernetes | 컨테이너 오케스트레이션 플랫폼으로, 서비스 디스커버리, 로드 밸런싱, 자동 복구 지원 | MSA 환경에서의 확장성과 무중단 배포 가능 |
|  | 3.5.2 Docker | 컨테이너 기반의 가상화 기술로, 애플리케이션의 이식성과 격리된 실행 환경 제공 | 개발 환경과 운영 환경의 일관성을 유지하며 빠른 배포가 가능함 |
|  | 3.5.3 OpenShift | Kubernetes 기반의 PaaS(Platform as a Service) 솔루션, 기업 환경에 맞춘 관리 및 배포 최적화 | 엔터프라이즈 환경에서의 중앙 집중형 관리와 보안 정책 설정 가능 |

### ****(실제) NH 농협지주 통합 플랫폼**** 기술 스택 및 플랫폼 선정 (Technical Stack & Platform Strategy)

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **전략적 의미** |
| 프론트엔드 | React | 사용자 인터페이스(UI)를 구성하는 프레임워크로, SPA 구조를 통해 빠른 화면 전환 및 반응성 제공 | 개발 속도 향상, 컴포넌트 기반의 유지보수성 확보, 데이터 포털/대시보드 구성에 적합 |
|  | MicroStrategy/Tableau | BI 리포팅 및 데이터 시각화 솔루션, 사용자 친화적 대시보드 구성 가능 | 정형 및 비정형 리포팅 제공, 마케팅/분석 업무 시각화 지원 |
| 백엔드 | Spring Boot | 경량화된 자바 기반 웹 프레임워크, 통합 데이터 AP 서버에 적용되어 REST API 제공 | 마이크로서비스 기반 API 서버 구성, 통합 데이터 서비스 처리 중심, 은행 표준 기술 기반으로 유지보수 용이 |
|  | xDataSet5 / xFrame5 | 은행 PaaS 환경에 적용된 UI 및 연동 프레임워크 | 통합 포털과 연계되는 웹/업무단말의 일관성 확보 |
| API 게이트웨이 | Spring Cloud Gateway | 인증/인가, 트래픽 제어, API 라우팅 기능을 제공하는 경량 API Gateway | OAuth2, JWT 인증 기반 연동, 통합 데이터 포털 및 외부 연계 API 흐름 통제 및 보안 확보 |
|  | Keycloak | OpenID Connect 기반의 인증·인가 시스템으로 사용자 접근 통제 및 연동 관리 지원 | OAuth2 기반 SSO 인증체계 구현, API 접근 보안 강화 |
| 데이터베이스 | Oracle HEX | 고성능 트랜잭션 처리 및 보안 요구가 높은 금융 환경에 적합한 RDBMS | 계열사 마트/정제 데이터 저장소로 활용, 무결성 및 데이터 분리 적용 용이 |
|  | MySQL | BI 포털 내부 리포팅 및 메타 정보 저장소로 사용 | 분석 및 통계 기반 경량 데이터 처리에 적합, 비용 효율성 확보 |
| ETL | IBM DataStage | 대용량 배치 처리, 정제 및 암호화 로직 구현에 사용 | 계열사 파일 수집·변환·이력·삭제 처리 일원화, 파일기반 수집 자동화 |
| 컨테이너 플랫폼 | VMware Tanzu + Kubernetes | Spring Boot 기반 컨테이너 운영 플랫폼으로 유연한 배포, 확장, 무중단 운영 지원 | 컨테이너 오케스트레이션 기반 자동 확장, 헬스체크, 롤백 전략 구성 가능 |
| 보안 | TOS / Deep Security | 서버 및 WAS 컨테이너 기반 위협 대응 시스템 | 내부 시스템 보안 정책 연계, 보안 로그 수집 및 정책 준수 관리 |
|  | DBSAFER / SSL 인증서 | DB 접근 통제 및 데이터 암호화, 통신 보안 정책 구현 | 개인정보 및 실명정보 보호를 위한 암호화, 접속 로그 감사 확보 |
| 로깅/모니터링 | Prometheus + Grafana | 시스템 리소스, 트랜잭션 처리율 등 실시간 수집 및 시각화 | SLA/SLO 모니터링, 알림 설정, 성능 튜닝 및 용량 계획에 핵심 지원 |
|  | ELK Stack (ElasticSearch) | 로그 수집 및 검색을 위한 통합 로그 플랫폼 | 장애 진단 및 트랜잭션 추적 기반 운영 대응 체계 구축 |
| 메시징 | Kafka / MQ | 비동기 메시징 기반 이벤트 처리, ML/DL 분석 결과 연계, 실시간 동의 철회 흐름 등에서 활용 | 이벤트 기반 처리 구조, 대용량 트래픽 대응, 비동기 설계 최적화 |
| 배포 및 자동화 | Jenkins + ArgoCD | CI/CD 자동화 툴로, 소스 통합, 이미지 생성, 자동 배포 파이프라인 지원 | GitOps 기반 운영체계 구현, 무중단 배포, 품질 검증 및 롤백 전략 가능 |
| 통합 플랫폼 연계 | FOS, MFT, MCA | 계열사 → 지주 시스템 간 파일/전문 연계에 사용되는 금융권 연계 플랫폼 표준 | 안정적 파일전송, 대외 API 라우팅, 보안 프로토콜(MCA, EAI 등) 준수 |

#### ✅ 시사점

* 기술스택은 **은행 표준 기술 프레임워크**와 **지주 맞춤형 클라우드 운영 체계**를 혼합하여 구성됨
* 모든 구성 요소는 **보안, 확장성, 실시간 처리, 무중단 운영**을 위한 구조로 통합됨
* 제안된 기술스택은 향후 MSA 전환과 DevSecOps 체계 확립에 기반이 됨

### 기술별 도입 목적 및 상세 아키텍처 적용 구조도 (텍스트 기반)

#### 🔹 1. Spring Boot – API 서비스 계층 핵심 프레임워크

**📌 도입 목적**: 경량화된 API 서버 구성, 빠른 개발, 마이크로서비스 구조 기반의 확장성 확보

[Client (UI/BI 포털, OLAP 등)]

└──▶ [API Gateway (Spring Cloud Gateway)]

└──▶ [통합 데이터 AP (Spring Boot)]

├──▶ REST API 요청 처리

├──▶ 통합 데이터 DB (Oracle HEX) 조회

├──▶ 외부 시스템 연계 (e-마케팅, UMS, OLAP)

└──▶ JWT 기반 인증 및 Keycloak 연동

#### 🔹 2. IBM DataStage – 데이터 수집 및 전처리 엔진

**📌 도입 목적**: 계열사 대용량 배치 수집, 암호화/삭제/이력로깅 처리, 고신뢰 ETL 파이프라인

[계열사 Legacy 시스템 (파일/MCA)]

└──▶ [대외 MCA + FOS (MFT)]

└──▶ [통합 데이터 ETL (DataStage)]

├──▶ 데이터 정제 및 변환

├──▶ 개인정보 암호화/마스킹

├──▶ 통합 고객번호 채번

└──▶ [통합 데이터 DB (Oracle HEX)] 적재

#### 🔹 3. Oracle HEX – 통합 데이터 저장소

**📌 도입 목적**: 금융권 수준의 고신뢰성 RDBMS, 마트화 및 분석용 DB 분리, 데이터 이중화

[통합 데이터 ETL]

└──▶ [통합 데이터 DB (Oracle HEX)]

├──▶ 원천 데이터 저장 (ODS)

├──▶ 마트 데이터 생성

├──▶ 통합 고객번호/분석용 데이터 저장

├──▶ BI 포털/OLAP/마케팅 API 연계

└──▶ JDBC 기반 연결로 OLAP/포털에 제공

#### 🔹 4. Kafka – 실시간 이벤트 기반 메시징

**📌 도입 목적**: 실시간 행태/반응 정보 처리, OLTP와 분석계 연계 처리, 비동기 이벤트 설계

[마케팅 시스템 / 빅스퀘어 (ML 분석)]

└──▶ [Kafka Topic]

├──▶ [e-마케팅 시스템] : 개인화 추천/행태 저장

├──▶ [마케팅 HUB] : 푸시 발송 트리거

└──▶ [BI 포털] : 실시간 반응 시각화용 데이터 수집

#### 🔹 5. Prometheus + Grafana – 운영 모니터링

**📌 도입 목적**: 서버, WAS, DB 리소스 상태 실시간 수집 및 대시보드화, SLA/SLO 감시

text

복사편집

[서버별 Agent (Node Exporter)]

└──▶ [Prometheus Server]

└──▶ [Grafana 대시보드]

├──▶ CPU/Memory/네트워크 상태 시각화

├──▶ API 응답 시간 추적

└──▶ 알람(슬랙/메일) 설정

#### 🔹 6. Keycloak – 인증 및 인가 시스템

**📌 도입 목적**: OAuth2 기반 API 인증, 사용자별 권한 관리 및 통합 로그인 지원 (SSO)

text

복사편집

[사용자 요청 (BI 포털, 마케팅 API)]

└──▶ [Keycloak 인증 서버]

└──▶ Access Token 발급

├──▶ JWT 토큰 기반 인증

├──▶ API Gateway 인증 필터 통과

└──▶ 사용자 Role 기반 권한 적용

#### 🔹 7. Jenkins + ArgoCD – CI/CD 파이프라인

**📌 도입 목적**: 코드 빌드부터 이미지 배포까지 자동화된 파이프라인 구성, 무중단 배포 실현

text

복사편집

[개발자 Git Commit]

└──▶ [Jenkins]

└──▶ 빌드 → 테스트 → Docker 이미지 생성

└──▶ [ArgoCD (GitOps)]

└──▶ Kubernetes 배포

├──▶ Spring Boot API 배포

└──▶ 마이크로서비스 업데이트 및 롤백

📌 시사점

* 기술별 도입은 단순한 프레임워크 채택을 넘어서, **서비스 흐름 내 명확한 역할 분담과 연계 구조**를 형성함
* 실제 NH 농협의 **은행 시스템 연계 표준(MCA, FOS, EAI)** 기반 위에 기술스택이 자연스럽게 통합되도록 설계되었음
* 이를 통해 고성능 처리, 보안 통제, 확장성, 실시간 대응이라는 **금융 시스템 핵심 요건**을 안정적으로 충족

### ****(실제) PaaS 기반 계층 구조 아키텍처****

#### ✅ 1. PaaS 기반 계층 구조 아키텍처 (텍스트 도식)

[사용자 계층 (계열사 사용자, 지주 사용자)]

└─▶ [접속 채널 (통합 단말, 포털 UI, OLAP UI)]

└─▶ [WEB 계층 (xFrame5, Apache, React/Vue)]

└─▶ [PT WAS (Spring Boot + xDataSet5)]

└─▶ [BT WAS (Spring Boot + API 처리)]

└─▶ [API Gateway (Spring Cloud Gateway)]

└─▶ [통합 데이터 AP]

├─▶ 통합 데이터 DB (Oracle HEX)

├─▶ OLAP 시스템 (JDBC)

├─▶ 마케팅 시스템 (REST API)

└─▶ BI 포털 (REST API, JDBC)

└─▶ [통합 인증 (Keycloak / JWT)]

└─▶ [보안 계층 (Deep Security, TOS, OAuth2, TLS)]

└─▶ [로그 및 감사 시스템 (ELK, Prometheus, Grafana)]

[ETL 계층 (DataStage)]

└─▶ [FOS (MFT)]

└─▶ [통합 데이터 ETL 서버]

└─▶ [통합 데이터 DB 적재 / 마트 생성]

[분석 계층]

├─▶ OLAP (MSTR 기반 분석)

├─▶ BI 포털 (Spring Boot + MySQL)

└─▶ 빅스퀘어 (ML/DL 분석 결과 송출)

[관리 및 배포 계층]

└─▶ Jenkins + Git → ArgoCD → K8s 클러스터 자동 배포

#### ✅ 2. 도입 기술 간 상호 작용 구조도

(1) API 통합 및 인증 흐름

[Client] → [API Gateway] → [Spring Boot API] ↔ [Keycloak]

↘ 요청 검증 / 인증 토큰 처리

(2) 데이터 수집 및 저장 흐름

[계열사 시스템] → [MCA / MFT] → [DataStage ETL] → [Oracle DB]

(3) 분석 및 리포팅

[Oracle DB] → [BI 포털 / OLAP / 마케팅 시스템] → [사용자 화면]

(4) 실시간 이벤트 처리

[빅스퀘어] → [Kafka Topic] → [마케팅 시스템 / e마케팅]

(5) CI/CD 자동화

[개발자 Git Commit] → [Jenkins Build] → [Docker Image 생성]

→ [ArgoCD 연동] → [Kubernetes 자동배포]

(6) 보안 및 통합 인증

[사용자 로그인] → [Keycloak] → [Token 발급] → [API 인증]

(7) 운영/모니터링

[각 서버 Agent] → [Prometheus] → [Grafana 대시보드]

↘ SLA/SLO 감시 및 알림 처리

#### 📌 정리

|  |  |
| --- | --- |
| 구분 | 설명 |
| **PaaS 계층화** | Channel Layer → Web/WAS → API → DB/분석 → 보안/모니터링으로 수직 분리 |
| **기술 흐름 최적화** | API → 인증 → DB 처리 → 결과 응답까지 최소 홉으로 설계 |
| **운영 효율화** | CI/CD, 모니터링, 보안이 전 계층에 통합 배치 |
| **실시간 연계 강화** | Kafka, API Gateway, Keycloak 등을 통한 동적 대응 구조 마련 |

필요 시 다음과 같이 구체적인 세부 도식 또는 표도 제공할 수 있습니다:

* 🔸 **배포 단위별 컨테이너 구성도**
* 🔸 **마이크로서비스 흐름도**
* 🔸 **인터페이스 연계 표 (ETL/API 구분)**
* 🔸 **CI/CD 흐름도 및 장애 시 롤백 시나리오**

원하시는 구성 방향을 알려주시면, 다음 항목부터 연결

### (실제) 마이크로서비스 흐름도 (TEXT 기반 도식)

[사용자/계열사/분석가]

│

▼

[API Gateway (인증, 라우팅, 모니터링)]

│

├──▶ [고객 관리 서비스]

│ └─▶ 통합 고객번호 생성 / 고객 정보 조회 / 개인정보 이력 관리

│ └─▶ Oracle DB (고객 도메인 테이블) 연계

│

├──▶ [거래 정보 서비스]

│ └─▶ 제3자 동의 정보 처리 / 상품 거래정보 집계

│ └─▶ MFT 통해 ETL 연계 후 DB 적재

│

├──▶ [데이터 분석 서비스]

│ └─▶ 분석 마트 조회 / 정제 데이터 병합 / 통계 요청 처리

│ └─▶ OLAP 연계 / BI 포털 통합

│

├──▶ [마케팅 정보 서비스]

│ └─▶ 타깃 고객 추출 / 캠페인 시나리오 필터링 / 피로도 정책 적용

│ └─▶ 마케팅 HUB, e마케팅 시스템, UMS 연계

│

├──▶ [이벤트 수집/전송 서비스]

│ └─▶ Kafka Topic 기반 실시간 이벤트 수집 및 전송

│ └─▶ ML/DL 분석 결과 송수신 (빅스퀘어 ↔ 마케팅)

│

├──▶ [통합 메타/품질 관리 서비스]

│ └─▶ 메타정보 제공 / 품질 진단 리포팅

│ └─▶ BI 포털 및 통합 데이터 포털과 연동

│

├──▶ [권한 및 인증 서비스]

│ └─▶ Keycloak 기반 OAuth2 인증 / JWT 발급

│ └─▶ 사용자 Role-Based Access Control (RBAC)

│

└──▶ [모니터링 및 감사 서비스]

└─▶ 로그 수집 (ELK), 이벤트 추적, SLA 모니터링

└─▶ Prometheus, Grafana 연계

[각 마이크로서비스는 아래 기술로 배포됨]

└─▶ Spring Boot 기반 서비스

└─▶ Docker 이미지 → Kubernetes 클러스터 배포

└─▶ CI/CD: Jenkins + ArgoCD

#### ✅ 흐름 요약

|  |  |
| --- | --- |
| 구분 | 설명 |
| **Gateway 중심 통합 구조** | 모든 요청은 API Gateway를 거쳐 서비스로 라우팅되며, 인증/인가/보안 적용 |
| **도메인별 서비스 분리** | 고객, 거래, 분석, 마케팅, 이벤트, 인증 등은 각각 독립된 MSA 서비스로 설계 |
| **이벤트 기반 확장성 확보** | Kafka 기반 이벤트 드리븐 아키텍처로 비동기 처리 및 분석 연계 |
| **모니터링 및 통합관리** | 각 MSA는 Prometheus + Grafana + ELK 연동으로 통합 추적 가능 |
| **클라우드 네이티브 배포** | PaaS 기반 Kubernetes 환경에서 무중단 배포 및 확장성 보장 |

### ****(실제)**** NH 농협지주 데이터 플랫폼: 데이터 흐름도 (텍스트 기반 도식)

[1] 데이터 수집 계층 (원천)

───────────────────────────────

[계열사 Legacy 시스템]

├─▶ (배치, FILE, MFT) ─▶ [은행 대외 MCA]

│ └─▶ [FOS Storage]

└─▶ (온라인, API, HTTP) ─▶ [은행 대외 MCA] ─▶ [지주 통합 데이터 AP]

[빅스퀘어 시스템]

├─▶ (배치, FILE, MFT) ─▶ [ETL]

└─▶ (Kafka, 실시간 Stream) ─▶ [마케팅 시스템 / E-마케팅 / HUB]

[마케팅 시스템 / E-마케팅 시스템]

└─▶ (배치 및 API 연계) ─▶ [통합 데이터 ETL]

───────────────────────────────

[2] 데이터 전처리 및 적재 계층

───────────────────────────────

[ETL 서버 (DataStage)]

├─▶ 전처리 (암호화 해제, 파기대상 필터링, 변환, 이력로깅)

└─▶ (JDBC) ─▶ [통합 데이터 DB (Oracle HEX)]

├─▶ 원본 레이어

├─▶ 정제/통합 레이어

├─▶ 마트(분석/마케팅/성과) 레이어

└─▶ 통합 고객번호/행태정보/동의정보 저장

───────────────────────────────

[3] 데이터 제공 및 활용 계층

───────────────────────────────

[통합 데이터 AP (SpringBoot API)]

├─▶ [통합 데이터 포털]

│ └─▶ 사용자별 마트/통계/품질 데이터 조회

├─▶ [OLAP 시스템 (MSTR)]

│ └─▶ 정제/분석 데이터 통합 조회

├─▶ [마케팅 시스템 / E-마케팅 시스템]

│ └─▶ 타깃고객 추출, 반응정보 제공, 피로도 정책 연계

├─▶ [BI 포털 DB (MySQL)]

│ └─▶ 메타데이터, 품질, 이력 조회

└─▶ [외부 시스템 (UMS, 올원뱅크)]

└─▶ 실시간 마케팅정보, 고객반응 연계

───────────────────────────────

[4] 분석 및 피드백 계층

───────────────────────────────

[OLAP 시스템]

├─▶ 통합 데이터 마트에서 KPI 및 대시보드 조회

├─▶ 성과 분석 → 피드백 제공 → 마케팅 재활용

[BI 포털]

├─▶ 메타관리, 사용자 정의 보고서, 정책기반 다운로드

├─▶ 분석결과 시각화, 통계 품질 리포팅

[빅스퀘어]

└─▶ ML/DL 모델 결과 → Kafka Topic → 마케팅 HUB, E-마케팅 시스템으로 실시간 반영

#### ✅ 핵심 흐름 요약

|  |  |
| --- | --- |
| 구분 | 주요 흐름 |
| **수집** | 계열사, 마케팅, 빅스퀘어로부터 MFT, Kafka, API 방식으로 데이터 수집 |
| **정제/적재** | ETL을 통해 전처리 후 원천/통합/마트 구조로 Oracle DB에 저장 |
| **제공/활용** | AP 서버가 통합 포털, OLAP, 마케팅, 외부 시스템에 REST API 제공 |
| **분석/피드백** | OLAP, BI 포털, 빅스퀘어에서 분석한 결과가 다시 플랫폼으로 순환 연계됨 |

#### 📌 전략적 시사점

* **계층적 레이어링 구조**를 통해 데이터 분리·관리·보안이 철저히 이루어짐
* **실시간/배치 혼합 처리**를 지원하여 다양한 연계 시나리오에 대응 가능
* **통합 고객번호 기반의 데이터 통합**으로 연계 일관성 확보
* **API 기반 서비스 중심 구조**로 마케팅, 외부 플랫폼과의 확장성 우수

### ****(실제) NH 농협지주 데이터 플랫폼**** 인터페이스 연계 상세 설계서 (표 기반)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **구분** | **연계 주체 (From → To)** | **연계 방식** | **포맷** | **주기** | **주요 처리 내용** |
| 1 | 계열사 파일 전송 | 계열사 → 대외 MCA → FOS → MFT → ETL | Batch / MFT | FILE | 1일 N회 | 제3자동의고객정보, 상품정보 등 파일 전송 후 ETL 적재 처리 |
| 2 | 계열사 API 연계 | 계열사 → 대외 MCA → 지주 통합 데이터 AP | 실시간 / HTTP / REST API | JSON/XML | 실시간 | 고객 정보 실시간 제공, 연동 데이터 실시간 응답 처리 |
| 3 | 분석정보 전송 | 빅스퀘어 → ETL | 배치 / MFT | FILE | 1일 1회 | 행태정보, 분석 결과 파일 수집 후 ETL 적재 |
| 4 | 실시간 분석 연계 | 빅스퀘어 → 마케팅 시스템 / HUB / E-마케팅 | 실시간 / Kafka | Stream | 실시간 | ML/DL 분석 결과 실시간 연계, 추천 정보 제공 |
| 5 | 성과 반응 수집 | 마케팅 시스템 → ETL | 배치 / MFT | FILE | 1일 1~2회 | 반응 정보, 캠페인 이력 수집 및 마트 반영 |
| 6 | 고객 이력 연계 | 외부 시스템 → 통합 데이터 AP | 실시간 / REST API | JSON | 실시간 | 통합 고객번호 기준 고객 거래 이력 집계 후 응답 |
| 7 | BI 포털 메타 요청 | BI 포털 → 통합 데이터 AP | 실시간 / REST API | JSON | 실시간 | 사용자별 메타데이터 요청 및 권한 검증 후 응답 |
| 8 | OLAP 정제 데이터 요청 | OLAP → 통합 데이터 AP | JDBC / REST API | SQL / JSON | 실시간 | 병합 마트 조회, 정제 데이터 API 제공 |
| 9 | 통합 데이터 DB 적재 | ETL → 통합 데이터 DB | Batch / JDBC | SQL | 1일 N회 | 원본 → 정제 → 마트 데이터 적재 및 통합 고객번호 저장 |
| 10 | 데이터 조회 | BI 포털 / 포털 UI / OLAP → 통합 데이터 DB | JDBC / SQL | SQL | 실시간 | 사용자 권한 기반 데이터 조회 및 분석 |
| 11 | 동의 철회 정보 실시간 | 계열사 FEP → 은행 대외 MCA → 통합 데이터 AP | 실시간 / REST API | JSON | 실시간 | 고객 동의 철회 요청 실시간 전파 |
| 12 | 동의 철회 정보 배치 | 계열사 FEP → 은행 대외 MCA → 통합 데이터 AP | Batch / MFT | FILE | 1일 1회 | 고객 철회 정보 파일 전송 후 적재 |
| 13 | 캠페인 대상 추출 요청 | 마케팅 시스템 → 통합 데이터 AP | REST API | JSON | 실시간 | 필터링 조건에 따른 대상 고객 마트 조회 → JSON 응답 |
| 14 | 마케팅 성과 정보 제공 | 통합 데이터 AP → 마케팅 시스템 / 올원뱅크 / UMS | REST API | JSON/XML | 실시간 | 반응 결과 제공, 배너 노출 등 타깃 캠페인 실행 |
| 15 | BI 포털 연계 | BI 포털 Proxy WEB → 통합 데이터 포털 | REST API / HTTP | JSON | 실시간 | 외부 사용자의 마케팅정보 요청 인터페이스 처리 |

#### 📌 시사점 및 전략적 적용 방향

* **다양한 연계방식 혼합 운영**: MFT, API, Kafka, JDBC 등 업무 특성에 맞춰 분리 구성
* \*\*중앙 API 게이트웨이(통합 데이터 AP)\*\*를 통해 외부 연계 통제 및 일관성 유지
* **데이터 흐름 단방향화 및 이력로깅 체계 구축**으로 보안성과 감사 대응력 향상
* **실시간과 배치 데이터 간 구분**으로 스케줄링, 부하 관리, 오류 대응 효율화 가능

### ✅ 데이터 흐름 단방향화 및 이력로깅 체계 구축

|  |  |  |
| --- | --- | --- |
| **구분** | **내용** | **전략적 의미** |
| **1. 설계 목적** | 데이터 흐름을 "단방향"으로 설계하여, 데이터 유출 및 무결성 훼손 위험을 원천 차단 | 보안성 강화, 시스템 격리, 정책 위반 최소화 |
| **2. 흐름 방향성 고정** | 모든 데이터 흐름을 계열사 → ETL → DB → API → 사용자 단말로 고정 | 역방향 요청 차단, 미승인 경로의 접근 통제 |
| **3. Push 기반 수집** | 외부로부터의 요청이 아닌, 정해진 프로토콜(MFT, Kafka 등)을 통해 내부로만 유입 | API 요청의 무분별한 증가 방지, 보안 경계의 명확화 |
| **4. FOS/MFT 적용** | 파일 연계는 반드시 FOS, MFT 등 "단방향 전송 인프라"를 통해 수신 | NAC(Network Access Control) + MFT로 신뢰 경로 확보 |
| **5. API 게이트웨이 단일화** | 외부 API 연동은 반드시 통합 데이터 AP(게이트웨이)를 거쳐야 하며, 직접 DB 접근은 차단 | API 인증/인가, 감사 추적, 로드 밸런싱 일괄 처리 |
| **6. ETL 및 연계 이력 로깅** | 모든 수집/연계/변환 과정의 로그는 별도 로깅 DB 또는 파일로 저장 | 장애 추적, 배치 오류 복원, 데이터 재처리 시점 회복 가능 |
| **7. 수집/변환/적재 각 단계별 ID 부여** | JOB ID, DATA ID 등 고유 ID 체계를 도입하여 데이터 흐름 추적 가능 | 데이터 프로세스의 "End-to-End 추적" 가능 |
| **8. 메타 로그 자동 연동** | 연계 내역은 메타관리 시스템과 자동 연동되어 실시간으로 데이터 계보(Lineage) 관리 | 내부 감사, 외부 규제(ISMS-P, GDPR 등)에 효과적으로 대응 가능 |
| **9. Kafka Log Topic 운영** | 실시간 이벤트는 Kafka로 수집되며, Log Topic 분리하여 모니터링 | 비동기 이벤트 흐름까지 추적 가능 |
| **10. 통합 로그 분석도구 연계** | ELK, Splunk 등 연계하여 검색/시각화 가능한 로그 포맷 적용 | 장애 원인분석 시간 단축, 운영 효율 극대화 |

#### 🔎 도식 (텍스트 기반)

[계열사 시스템]

└─(MFT/파일)─▶ [FOS/MCA]

└─▶ [ETL 수집 영역]

└─▶ [통합데이터 DB]

└─▶ [통합데이터 AP]

├─▶ 사용자 포털

├─▶ OLAP 시스템

└─▶ 외부 API

※ 각 단계별:

- JOB ID / 수집 시간 / 대상 시스템 / 파라미터 등 로그 저장

- 로그는 통합 이력관리 DB + 실시간 Kafka Log Topic 으로 동시 기록

- 메타시스템과 연동되어 계보 관리 수행

#### 📌 시사점

* **보안성 확보**: 단방향 흐름은 외부의 비인가 접근 가능성을 원천 차단
* **감사 대응력 강화**: 이력 추적 기반으로 ISMS-P, 전자금융감독규정 대응 가능
* **장애 대응력 향상**: 로그 기반으로 수집 실패, 적재 누락 등 장애 원인을 명확히 추적 가능
* **확장성 기반 확보**: 이력 기반 처리가 체계화되면 재처리, 백업, 데이터 마이그레이션이 유연해짐

## ****마무리말****

기술 스택 및 플랫폼의 적절한 선정은 아키텍처 설계의 성공 여부를 좌우합니다. 프론트엔드부터 백엔드, API Gateway, 데이터베이스, 클라우드 플랫폼까지 각 기술 요소의 역할을 명확하게 정의하고 최적의 조합을 이루면 성능 최적화와 유지보수 효율성을 동시에 확보할 수 있습니다. 특히 MSA 구조를 위한 Kubernetes와 Docker 조합은 확장성과 무중단 배포를 보장하며, 금융권 요구에 맞춘 Oracle과 MongoDB는 데이터의 일관성과 확장성을 높입니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **프론트엔드** | 사용자 중심의 UI 설계와 반응형 구조로 빠른 인터페이스 제공이 가능하다. |
| **백엔드** | MSA 구조를 지원하며, 독립적이고 확장 가능한 서비스 개발이 용이하다. |
| **API Gateway** | 서비스 간 통신 최적화 및 보안 정책을 통해 신뢰성 높은 아키텍처를 구성할 수 있다. |
| **데이터베이스** | 다양한 DBMS 조합으로 비즈니스 요구에 맞춘 최적화된 데이터 관리가 가능하다. |
| **클라우드 및 컨테이너** | 컨테이너 오케스트레이션을 통해 배포 시간 단축 및 장애 복구가 신속하게 이루어진다. |

# ****어플리케이션 계층 구조 설계****

## ****도입전안내말****

어플리케이션 계층 구조 설계는 EA(Enterprise Architecture) 관점에서 시스템의 역할과 기능을 명확히 구분하여 관리 효율성을 높이는 핵심 설계 단계입니다. 계층 구조를 명확히 구분함으로써 확장성, 유지보수성, 성능 최적화를 동시에 실현할 수 있습니다.  
계층 구조는 Channel Layer, Interface Layer, Business Layer, Data Layer로 구분되며, 각 계층은 독립적으로 역할을 수행하여 모듈화와 재사용성을 극대화합니다.

## ****어플리케애션 계층구조 설계****

|  |  |  |  |
| --- | --- | --- | --- |
| **계층명** | **주요 역할** | **구성 요소** | **연계 기술** |
| **4.1 Channel Layer** | 사용자와의 인터페이스를 담당하는 레이어 | React, Angular, Vue | REST API, WebSocket |
| **4.1.1 웹 프론트엔드** | 웹 애플리케이션 UI 제공, SPA(Single Page Application) 구조로 빠른 응답성 확보 | React, Angular | HTTP, HTTPS |
| **4.1.2 모바일 프론트엔드** | 모바일 전용 애플리케이션 UI 제공, Native 또는 Hybrid 방식으로 구현 | React Native, Flutter | HTTP, WebSocket |
| **4.2 Interface Layer** | 외부 시스템 및 채널과의 인터페이스를 처리하는 레이어 | Zuul, Kong, Spring Cloud Gateway | gRPC, REST, Kafka |
| **4.2.1 API Gateway** | API 요청을 수집하고 내부 서비스로 라우팅하여 부하 분산 처리 | Zuul, Kong | HTTP, HTTPS, OAuth |
| **4.2.2 Message Queue** | 비동기 데이터 전송 및 트랜잭션 처리 | Kafka, RabbitMQ | AMQP, MQTT, Kafka Connect |
| **4.2.3 외부 시스템 연계** | 외부 금융 시스템 또는 공공 API와 연계하여 데이터 수집 및 교환 | API Manager, ESB(Enterprise Service Bus) | SOAP, REST, gRPC |
| **4.3 Business Layer** | 비즈니스 로직을 처리하며 도메인별로 모듈화된 구조로 설계 | Spring Boot, Microservices | JPA, Hibernate, Redis |
| **4.3.1 FACADE** | 복잡한 비즈니스 로직을 단일 진입점으로 통합하여 인터페이스 단순화 | Spring Controller, REST API | HTTP, JSON |
| **4.3.2 RULE** | 비즈니스 규칙과 로직을 모듈화하여 중복 코드 방지 | Drools, Business Rules Engine | Rule Engine API |
| **4.3.3 THING** | 업무 객체를 관리하며, 도메인 특화 로직을 구현 | Spring Beans, POJO | Service Layer 호출 |
| **4.3.4 DAO** | 데이터 접근 계층으로 DB와 직접 연동하여 데이터를 CRUD 처리 | JPA, MyBatis, Hibernate | JDBC, SQL |
| **4.4 Data Layer** | 데이터의 영속성을 관리하고 DB와의 직접적인 상호작용을 처리 | Oracle, MySQL, MongoDB | JDBC, JPA, MyBatis |
| **4.4.1 Persistence** | 데이터의 영구 저장을 위해 데이터베이스와 직접 연결 | Oracle, PostgreSQL | SQL, PL/SQL |
| **4.4.2 Connection Pool** | 데이터베이스 연결을 효율적으로 관리하여 성능을 최적화 | HikariCP, DBCP | Connection Pooling |
| **4.4.3 트랜잭션 관리** | 트랜잭션의 ACID 속성을 보장하여 데이터 무결성 유지 | Spring Transaction Manager | XA Transaction |

### ****(실제) NH 농협지주 데이터 플랫폼 어플리케이션 상세설계****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 계층명 | 설계 항목 | 세부 설계 요소 (10개 이상) | | 세부 설계 요소별 설계 전략 (각 요소당 10개) | |
| Channel Layer | 웹 프론트엔드 구성 | 1. SPA 기반 React/Vue 사용 | | ① CSR 기반 렌더링 구조 채택  ② 초기 로딩 성능 개선 전략 적용  ③ 코드 스플리팅 활용  ④ Lazy Loading 적용  ⑤ 빌드 최적화 구성  ⑥ 정적 자산 CDN 연계  ⑦ SEO 최적화 미들웨어  ⑧ 브라우저 호환성 검사  ⑨ 동적 import 활용  ⑩ 비동기 렌더링 처리 | |
|  |  | 2. JWT 인증 적용 | | ① 토큰 기반 인증 미들웨어 삽입  ② 로컬 스토리지 저장 전략 적용  ③ 만료 및 갱신 정책 구현  ④ Refresh Token 처리 흐름  ⑤ XSS 방지 토큰 암호화  ⑥ HTTPOnly 쿠키 활용 고려  ⑦ 토큰 검증 모듈화  ⑧ 사용자 권한 처리 분기화  ⑨ 인증 실패 오류 코드 처리  ⑩ 로그인 세션 유지정책 설계 | |
|  |  | 3. OAuth2 기반 인증 흐름 | | ① 인증서버와 토큰 교환 구성  ② 권한 Scope 설계  ③ Refresh Token 주기 관리  ④ Client ID/Secret 관리  ⑤ Authorization Code Flow 사용  ⑥ 인증 실패 예외처리  ⑦ 다중 플랫폼 인증 구성  ⑧ Redirect URL 보안 설계  ⑨ 사용자 프로필 수신 처리  ⑩ API 인증 미들웨어 설계 | |
|  |  | 4. WebSocket 실시간 통신 | | ① 양방향 연결 구조 구성  ② 연결 상태 재시도 로직  ③ 메시지 보안 암호화 적용  ④ 채널 분리 구조 설계  ⑤ Keep-Alive 전략 적용  ⑥ 웹소켓 인증 구조 연계  ⑦ 비정상 연결 탐지 로직  ⑧ 메시지 Queue 연동 구성  ⑨ 실시간 장애 발생 알림 처리  ⑩ Load Balancer 기반 커넥션 유지 | |
|  |  | 5. Axios/FETCH API 처리 | | ① 인터셉터 활용 응답/오류 처리  ② 공통 API 모듈화 설계  ③ Retry 및 Timeout 설정 적용  ④ 인증 실패 재로그인 처리  ⑤ 네트워크 장애 감지 구조  ⑥ 멀티 API 병렬 처리 흐름 설계  ⑦ Request 취소 처리  ⑧ Base URL 모듈화  ⑨ 응답 캐싱 전략 구성  ⑩ 모니터링 로깅 연계 | |
|  |  | 6. 전역 상태 관리 | | ① Redux/Vuex 기반 아키텍처 구성  ② 비동기 Action 처리 설계  ③ 상태 Persistence 저장 적용  ④ 미들웨어 분리 처리  ⑤ 모듈 단위 분리 저장소 설계  ⑥ 로딩 처리 공통화  ⑦ 캐시 무효화 전략 설계  ⑧ 상태 추적 디버깅 툴 연계  ⑨ 상태 변경 로깅  ⑩ 다중 탭 상태 동기화 | |
|  |  | 7. UI 접근성 표준 준수 | | ① 웹 접근성 WCAG 기준 적용  ② 키보드 네비게이션 구성  ③ 스크린리더 호환성 테스트 수행  ④ ARIA 속성 활용  ⑤ 명확한 포커싱 처리  ⑥ 대비도 기준 충족  ⑦ 반응형 UI 구현  ⑧ 동적 요소 접근성 적용  ⑨ 사용자별 접근성 설정 제공  ⑩ 접근성 이슈 리포팅 체계 구축 | |
|  |  | 8. 컴포넌트 단위 개발 | | ① Atomic Design Pattern 적용  ② 재사용성 중심 설계  ③ Props-Driven 디자인 활용  ④ 테스트 가능한 구조 설계  ⑤ 독립 상태관리 적용  ⑥ Storybook 기반 UI 문서화  ⑦ 공통 스타일/컴포넌트 통합관리  ⑧ 런타임 타입체크 적용  ⑨ 컴포넌트 버전 관리  ⑩ 컴포넌트 성능 측정 체계 구축 | |
|  |  | 9. 멀티 디바이스 대응 | | ① 반응형 뷰포트 설정  ② 미디어쿼리 기반 스타일 구성  ③ 사용자 에이전트 기반 최적화 적용  ④ 모바일/태블릿 분기 처리  ⑤ 터치 UI 전환 구성  ⑥ Gesture 기반 UI 처리  ⑦ 디바이스 별 폰트/크기 대응  ⑧ 다국어/다문화 설정 지원  ⑨ 해상도 적응형 이미지 처리  ⑩ 성능 최적화 로딩 구성 | |
|  |  | 10. 보안 헤더 적용 | | ① CSP 헤더 구성  ② CORS 정책 설정  ③ XSS 보호 기능 활성화 ④ Referrer-Policy 적용 ⑤ Secure/HttpOnly 설정 ⑥ MIME Sniffing 방지 설정 ⑦ 프레임 소스 제한 적용 ⑧ 브라우저 보안 분석 대응 ⑨ API 응답 보안 헤더 설정 ⑩ 서버 측 헤더 캐시정책 연계 | |
| **Channel Layer** | 모바일 프론트엔드 구성 | 1. React Native 기반 구축 | | ① 크로스 플랫폼 설계 적용 ② Hot Reloading 기반 빠른 피드베트 ③ 플랫폼별 커포넌트 추사화 ④ 공통 서비스 계층 모듈화 ⑤ Expo/Native CLI 최적 선택 전략 ⑥ 디바이스 호환성 기준 설계 ⑦ OTA 업데이트 프로세스 적용 ⑧ Native API 연계 설계 ⑨ App Store/Play Store 배포 전략 ⑩ 앱 내 권한 통제 구성 | |
|  |  | 2. Native Module 연계 | | ① Android/iOS 해당 Native SDK 연계 ② 통신, 컴보리, 커메니, GPS 기능 발표 ③ Native Bridge 구성 ④ Expo Eject 적용 전략 ⑤ 시스템 해외 API 연계 ⑥ 제3자 SDK 추가 관리 ⑦ Native Plugin 관리 가용 ⑧ Module 호환 테스트 ⑨ Hook을 활용한 제어 ⑩ 번개적 추가에 대한 정확도 가지변경 | |
|  |  | 3. UI 커스텀 조정 스크립트 | | ① Tailwind, Styled-Component, Emotion 구성 ② 일괄된 Theme 관리 전략 ③ 커스텀 Hook 분리 사용 ④ Color Scheme / 다구역 지원 ⑤ 개인화 커스텀 데이터 변경 ⑥ Dynamic Style 적용 ⑦ 디바이스 다른 UI Size 호환 ⑧ 시간 간격 대상 관리 ⑨ 메타이너 정책 호출 ⑩ Style 최적화 변경 가능 | |
| Interface Layer | API Gateway 구성 | 1. 통합 인증 처리 | | ① OAuth2.0 기반 인증 구현 ② JWT 토큰 유효성 검사 ③ 세션 미보관 Stateless 구조 ④ 인증 전용 인증서 적용 ⑤ SSO 연동 ⑥ Keycloak 기반 인증 정책 관리 ⑦ 인증서 자동 갱신 ⑧ 인증 로깅 및 추적 ⑨ API 인증 필터 적용 ⑩ 리버스 프록시 인증 처리 | |
|  |  | 2. 서비스 라우팅 | | ① 경로 기반 URI 매핑 설계 ② API Prefix 분류 기준 명확화 ③ PathVariable 동적 처리 전략 ④ 서비스 종류별 분산 라우팅 적용 ⑤ 서비스 이름 기반 로드밸런싱 ⑥ Circuit Breaker 연계 ⑦ API 버전별 라우팅 처리 ⑧ MSA 서비스 동기화 ⑨ GraphQL과 REST 병행 처리 전략 ⑩ 서비스 다운 시 예외 반환 전략 | |
|  |  | 3. 트래픽 부하 분산 | | ① 라운드로빈 기반 부하분산 ② 요청량 기반 동적 분산 ③ 캐싱을 통한 트래픽 경감 ④ API 리밋 정책 구성 ⑤ 요청 큐 처리 ⑥ 헤더 정보 기반 분기 처리 ⑦ CDN 연계 ⑧ 리버스 프록시 활용 ⑨ 클러스터 기반 스케일아웃 ⑩ 비정상 트래픽 차단 전략 | |
|  |  | 4. API 버전 관리 | | ① URL 버전(v1, v2) 명시화 ② 버전별 별도 디렉터리 구조 관리 ③ Swagger 기반 문서화 ④ Deprecated API 구분 표기 ⑤ 버전별 동시 운영 전략 ⑥ 사용자 대상 API 변경 공지 연계 ⑦ 버전별 테스트 시나리오 분리 ⑧ 신규 API는 내부 검증 후 공개 ⑨ 클라이언트 단 버전 필수 지정 ⑩ 리버스 호환성 검토 | |
|  |  | 5. API 보안 정책 | | ① 인증 없는 엔드포인트 제거 ② API Key 발급 및 검증 ③ 접근 IP 화이트리스트 설정 ④ SSL/TLS 암호화 적용 ⑤ Web Application Firewall 연동 ⑥ 인가 실패 로깅 강화 ⑦ Request Payload 검증 필터링 ⑧ Input Validation 처리 ⑨ REST/GraphQL 별도 방화벽 구성 ⑩ 요청/응답 암호화 처리 | |
|  |  | 6. API 요청 로깅 및 감사 | | ① 요청별 Request ID 부여 ② 로그 추적용 TRACE-ID 관리 ③ 요청/응답 내용 부분 로깅 ④ 민감 정보 마스킹 후 기록 ⑤ API 호출 이력 DB 저장 ⑥ 일별 호출 통계 분리 기록 ⑦ 연동 장애 시 API Timeout 자동 기록 ⑧ 비정상 호출 패턴 알림 연계 ⑨ 호출자 식별 정보 포함 ⑩ 데이터 보호 정책에 따른 로그 보존 주기 설정 | |
|  |  | 7. 장애 자동 전환 | | ① 서비스 상태 헬스체크 구성 ② 실패율 기반 Circuit Breaker 설정 ③ Fallback API 구성 ④ 상태 변화 시 알람 연계 ⑤ 장애 발생 시 Failover 구동 ⑥ 장애 분석용 히트맵 저장 ⑦ 비정상 응답률 임계치 설정 ⑧ 상태 변경 이력 로깅 ⑨ SLA 기반 자동 Failback 적용 ⑩ 부하 우회 라우팅 구성 | |
|  |  | 8. API 메타데이터 관리 | | ① Swagger 기반 문서화 자동화 ② API 설명 주석 일괄 반영 ③ Postman Collection 동기화 ④ OpenAPI 스펙 파일 관리 ⑤ API 사용량/호출량 기록 대시보드화 ⑥ URL, 파라미터 구조 자동 추출 ⑦ 버전별 API 문서 분리 ⑧ 개발자 포털 통한 공유 ⑨ 메타 정보와 실 운영 정보 일치 점검 ⑩ API 변경 이력 관리 기능 | |
|  |  | 9. 메시지 큐 연계 처리 | | ① Kafka 토픽 기반 분산 메시징 ② AMQP 기반 비동기 처리 지원 ③ 이벤트 발행/구독 구조 설계 ④ 메시지 중복 처리 방지 설계 ⑤ 메시지 정렬 및 순서 보장 전략 ⑥ 메시지 유실 방지 로그 이중 저장 ⑦ 메시지 TTL 기반 정리 정책 ⑧ 소비자 장애 시 재시도 정책 ⑨ 메시지 Ack 처리 로직 분리 ⑩ QoS 기준 설정 | |
|  |  | 10. 외부 시스템 연계 | | ① 공공기관/계열사 API 연동 ② SOAP, REST 혼합 지원 ③ 보안 채널(TLS/SSL) 사용 ④ 요청 시 인증 토큰 첨부 ⑤ 연계 실패 시 재시도 정책 적용 ⑥ 요청 타임아웃 설정 ⑦ 외부망 연계 시 MFT 또는 FEP 사용 ⑧ 요청 결과 SLA 관리 ⑨ API 응답 지연 로그 추적 ⑩ 외부 연동용 별도 API 라우팅 구성 | |
| Interface Layer | API Gateway 구성 | 1. 인증 및 권한 처리 | | ① OAuth2.0 기반 인증 구성② JWT 기반 토큰 검증③ Keycloak 연계④ API 요청 별 Role Mapping⑤ Rate Limit 설정⑥ Scope 기반 접근 제어⑦ API Consumer 등록 관리⑧ Refresh Token 정책 적용⑨ 사용자 세션 추적 로그⑩ 인증 실패 응답 표준화 | |
|  |  | 2. 요청 라우팅 처리 | | ① URI 기반 동적 라우팅 구성② 요청 Header 기반 분기 처리③ MSA별 Backend API 분리④ 서비스 디스커버리 기반 매핑⑤ 경량화된 요청 맵핑 캐시 전략⑥ 응답 상태 기반 리다이렉션 처리⑦ 프록시 경로 마스킹 처리⑧ 경로 기반 필터 적용⑨ JSON Schema 기반 검증⑩ API 버전별 라우팅 설정 | |
|  |  | 3. 장애 대응 및 복구 | | ① Circuit Breaker 적용② Retry 정책 구성③ 요청 대기 큐 구성④ Failover Node 지정⑤ 장애시 SLA 로그 기록⑥ Latency 감지 및 자동 우회⑦ 예외 응답 표준화⑧ API 헬스체크 연동⑨ 응답 타임아웃 설정⑩ 백오프 전략 적용 | |
|  |  | 4. 모니터링 및 로깅 구성 | | ① API Gateway 별 트래픽 분석② 실시간 로그 수집기 연동③ OpenTelemetry 적용④ Kibana 시각화⑤ 응답 지연 구간 추적⑥ 에러율 감지 정책⑦ 로그 필터링 기준 수립⑧ 인증 실패 로그 집중 분석⑨ 로그 레벨 표준화⑩ API 호출 이력 저장 | |
|  | 메시지 큐 구성 | 5. 비동기 통신 처리 구조 | | ① Kafka 기반 토픽 구조 설계② AMQP 적용 구조 비교③ Pub-Sub 모델 설계④ 이벤트 발행 시점 정의⑤ 소비자 그룹 분리 전략⑥ 메시지 유실 방지 정책⑦ 재처리 큐 및 DLQ 구성⑧ 메시지 처리 순서 보장 전략⑨ 브로커 이중화 구성⑩ 메시지 압축 및 인코딩 적용 | |
|  |  | 6. 데이터 연계 안정성 확보 | | ① 이벤트 중복 방지 키 적용② 메시지 ACK 처리 보장③ 브로커-컨슈머 간 인증 적용④ 메시지 도착 시간 검증⑤ Kafka Topic TTL 관리⑥ 메시지 처리 완료 후 로깅⑦ 이벤트 발행 실패시 경고 로그⑧ 스키마 레지스트리 적용⑨ JSON/AVRO 포맷 정합성 검증⑩ 역직렬화 오류 예외 처리 | |
|  | 외부 시스템 연계 설계 | 7. 외부 API 표준화 | | ① API 명세서(Swagger) 정의② 요청/응답 포맷 통일(JSON/XML)③ 상태코드 기반 응답정책④ 비즈니스 용도별 API 분리⑤ 호출 빈도 제한 설정⑥ API 계약 버전 관리⑦ 외부 도메인 인증 토큰 관리⑧ API 응답 캐싱 적용⑨ Audit 로그 저장⑩ API 호출 시나리오 매핑 | |
|  |  | 8. 실시간 연계 구조 | | ① gRPC 기반 빠른 처리 적용② HTTP2 연계 성능 개선③ Kafka Stream 연계 구조④ WebSocket 기반 처리 정의⑤ Request/Response 간 타이밍 로깅⑥ 동기-비동기 병렬 처리 구조⑦ Interface 호출 트랜잭션 ID 관리⑧ 외부 호출 실패시 Retry 정책⑨ 연계 지연 시 알림 기능⑩ 실시간 모니터링 연계 | |
|  |  | 9. 표준 프로토콜 설계 | | ① RESTful API 표준 구성② gRPC 호출 형식 정의③ SOAP API 구조 호환④ Kafka Topic 규칙 정립⑤ JSON-RPC 규격 설정⑥ HTTP Method 기반 기능 맵핑⑦ 요청 URI 네이밍 컨벤션 정의⑧ API 응답 포맷 필드 정의⑨ OpenAPI 3.0 스펙 준수⑩ 통신 암호화(TLS) 적용 | |
|  |  | 10. 연계 이력 로깅 구조 | | ① 요청-응답 간 GUID 매칭 저장② 연계 실패 로그 이력화③ 이력 DB 스키마 정의④ 호출 일시, IP, 사용자 기록⑤ Kafka Stream 처리 로그 저장⑥ 연계 오류 알림 및 회수 시나리오 정의⑦ 통합 모니터링과 연동⑧ REST API 이력 리포팅 API 구현⑨ FOS 파일 송수신 이력 관리⑩ 인터페이스 이중화 로그 설계 | |
|  | **설계 항목** | **세부 설계 요소** | | **세부 설계 요소별 설계 전략 (각 요소당 10개 이상)** | |
| Business Layer | FACADE 구성 | 1. 통합 진입점 컨트롤러 구성 | | ① REST API 중심 단일 진입점 설계② 경량 컨트롤러 역할 한정③ 공통 예외 핸들링 적용④ DTO로 데이터 캡슐화⑤ 인증/인가 선처리⑥ 서비스별 경로 분리 구성⑦ AOP 기반 로깅 적용⑧ 요청/응답 표준화 설계⑨ 도메인 간 라우팅 정의⑩ 응답시간 SLA 로깅 적용 | |
|  |  | 2. 응답 결과 통합 포맷 처리 | | ① 통일된 Response Wrapper 구성② HttpStatus 기반 상태 전달③ 메시지코드 표준화 적용④ 실패 응답에 오류코드 추가⑤ Null 응답 방지 구조⑥ 응답 필드별 유효성 검증⑦ 성능 메트릭 포함 응답⑧ 페이징 결과 포함 구성⑨ API 버전 관리 반영⑩ JSON/XML 자동 변환 설정 | |
|  | RULE 구성 | 3. 비즈니스 룰 엔진 적용 | | ① Drools 기반 Rule Engine 구성② Rule 단위 모듈화③ Rule Repository 분리 관리④ 조건별 시나리오 Mapping⑤ 브랜치 분기 최적화⑥ External Rule Override 기능⑦ Rule 버전 이력 관리⑧ Rule 충돌 검출 메커니즘⑨ Rule 트리 구조 시각화⑩ 테스트 가능 Rule 유닛 구성 | |
|  |  | 4. 정책 변경 대응 구조 | | ① Config Server 기반 설정 변경 가능② Rule 변경 시 무중단 반영 구조③ 정책 변동 시 리프레시 트리거④ 정책별 Audit 기록 적용⑤ 변경 감지 이벤트 발행⑥ 관리자 화면 통한 Rule 설정 UI 제공⑦ Versioned Policy Repository 사용⑧ 정책 영향 분석 리포트 출력⑨ 변경 전/후 비교 로직 구성⑩ 실시간 룰 반영 테스트 기능 | |
|  | THING 구성 | 5. 도메인 객체 설계 | | ① POJO 기반 비즈니스 엔티티 구성② Builder 패턴 적용③ 도메인 단위 유효성 검증 포함④ 값 객체(Value Object) 설계 분리⑤ JPA Entity와 분리된 도메인⑥ Serialization 고려 설계⑦ Audit 필드 포함 설계⑧ 비즈니스 규칙 캡슐화⑨ Lombok 사용으로 코드 경량화⑩ 도메인 간 참조 최소화 전략 | |
|  | DAO 구성 | 6. 데이터 접근 인터페이스 구성 | | ① Repository 패턴 기반 설계② JPA + QueryDSL 병행 사용③ 읽기/쓰기 분리 구성④ 트랜잭션 단위 명확화⑤ Native Query 보완 구성⑥ 페이징 처리 자동화⑦ Lazy/Eager 전략 적용⑧ 인덱스 활용 SQL 최적화⑨ AuditTrail 필드 자동 반영⑩ DB Connection Pool 적용 | |
|  |  | 7. 예외 처리 및 오류 전략 | | ① DAO 계층 전용 예외 클래스 정의② SQLException 분류 처리③ Retry Template 적용④ API 응답으로 예외 매핑⑤ 트랜잭션 롤백 예외 설정⑥ 에러 로그 Trace 저장⑦ 감지된 오류 DB 이력화⑧ 클라이언트용 예외 변환기 구성⑨ 예외 메시지 다국어 지원⑩ 내부 로직 오류 숨김 처리 | |
|  | 트랜잭션 설계 | 8. ACID 트랜잭션 보장 설계 | | ① Spring Transaction Manager 적용② @Transactional 단위 설계③ 트랜잭션 격리수준 설정④ XA 트랜잭션 적용 가능성 확보⑤ 단일 작업 단위 기준 트랜잭션 설계⑥ 데이터 정합성 제약조건 강화⑦ 커넥션 종료 전 커밋 관리⑧ 롤백 처리 기준 명확화⑨ 트랜잭션 로그 로깅⑩ DB Lock 충돌 감지 처리 | |
|  | 캐싱 및 성능설계 | 9. 비즈니스 레이어 캐싱 전략 | | ① Redis 기반 캐시 구성② 읽기 전용 데이터 캐시 적용③ TTL 정책 기반 캐시 갱신④ API 응답 캐싱 구간 정의⑤ Caffeine 활용한 Local Cache 전략⑥ 캐시 Key 명명 규칙 통일⑦ 캐시 미스 로깅 분석⑧ 변경 이벤트 기반 캐시 무효화⑨ 캐시 분산 서버 구조⑩ 메모리 사용량 모니터링 | |
|  |  | 10. 병렬 처리 최적화 | | ① 비동기 서비스(@Async) 적용② ForkJoinPool 병렬 처리③ CompletableFuture 활용④ 데이터 배치 처리 분리⑤ 병렬 처리 가능한 도메인 추출⑥ CPU Core 기반 Thread Pool 구성⑦ 처리 대기 큐 모니터링⑧ 처리 순서 제어 로직 삽입⑨ 비동기 실패 처리 구조⑩ 트랜잭션 경합 방지 설계 | |
|  | **설계 항목** | **설명** | | **세부 기술 요소 (10개 이상)** | |
| Interface Layer | 1. API Gateway 구성 | 클라이언트 요청을 수신하고 내부 서비스로 라우팅하며, 인증/인가, 로깅, 모니터링 역할 수행 | | ① Zuul 또는 Spring Cloud Gateway 적용② OAuth2 / JWT 기반 인증처리③ URI 기반 라우팅 룰 정의④ 요청 스로틀링 제한⑤ HTTPS 강제 적용⑥ 서비스별 헤더 정제 필터 적용⑦ 로드 밸런싱 연계 (Ribbon 등)⑧ 사용자 ID 기반 요청 추적 (Trace ID)⑨ 통합 로깅 필터 구성⑩ 장애 발생 시 Graceful Degradation 처리 | |
|  | 2. 서비스 라우팅 및 버전 관리 | API 요청을 해당 서비스로 분기하며, 버전별 API를 분리하여 관리 | | ① API 버전 구분 (URI 기반 / Header 기반)② 서비스 등록 시 경로 Prefix 관리③ Canary 릴리즈 적용을 위한 조건 분기④ 버전별 Swagger 문서 자동화⑤ 레거시 API Backward Compatibility 유지⑥ 신규 서비스에 대한 Mock API 제공⑦ API Deprecation 일정 관리⑧ API 요청 통계 분석 기반 리라우팅 정책⑨ 블루그린 배포 대응 전략⑩ 테스트 환경 분리 적용 | |
|  | 3. Message Queue 연계 설계 | 비동기 처리 및 시스템 간 이벤트 기반 통신 수행 | | ① Kafka 기반 메시징 아키텍처 설계② Topic 단위 업무 이벤트 구분③ Key 기반 파티셔닝 전략④ 메시지 유효시간 설정(TTL)⑤ 메시지 재처리 정책 구성⑥ 장애 시 메시지 재전송 설정⑦ Kafka Connect로 CDC 연계⑧ Schema Registry 관리 체계 구축⑨ 메시지 암호화 적용 여부 검토⑩ 비정상 메시지 처리 DLQ 구성 | |
|  | 4. 외부 시스템 연계 설계 | 계열사 및 공공기관 등 외부 시스템과의 실시간 및 배치 연계 처리 | | ① RESTful API 연계 (HTTP/HTTPS)② SOAP 또는 gRPC 연계 지원③ 대외MCA 통한 파일 연계 구성 (MFT 포함)④ 표준 JSON/XML 포맷 정의⑤ 헤더 기반 인증 연계 구조⑥ 타시스템 CI 기반 사용자 Mapping 정책⑦ API Rate Limit 적용⑧ 전송 암호화(TLS/SSL) 적용⑨ 접근IP 화이트리스트 정책⑩ 연계 실패 시 재시도 및 대기 큐 구성 | |
|  | 5. 배치 연계 설계 | 파일 기반의 정기적 데이터 전송 및 수집 | | ① 계열사 → 대외 MCA → FOS → ETL 경로 구성② 정기 일정 기반 배치 수행 (Control-M)③ 파일 Naming 규칙 표준화④ Encoding 표준 적용 (UTF-8, EUC-KR 등)⑤ 파일 유효성 검증 로직⑥ 중복 전송 방지 체크 구성⑦ 파일 처리 후 응답 반환 방식 정의⑧ 보안 적용 (암호화/전자서명)⑨ 배치 수행 결과 Logging 및 알림 구성⑩ 오류 파일 백업 및 관리자 UI 알림 연동 | |
|  | 6. 실시간 연계 설계 | 실시간 요청/응답 연계를 통한 사용자 중심 서비스 구현 | | ① API Gateway를 통한 호출 경로 구성② OAuth2 인증 기반 서비스 호출③ 응답 SLA 기준 3초 내 처리 설계④ 통합 고객번호 기준 데이터 조인 처리⑤ API 타임아웃 설정 및 재시도 정책⑥ 지연 응답 처리 위한 비동기 응답 전략⑦ 인터페이스 실패 이력 로깅⑧ 서비스 간 장애 시 Circuit Breaker 연계⑨ OLAP 연계 결과를 즉시 포털로 전송⑩ 메타데이터 동기화 처리 포함 | |
|  | 7. 인터페이스 로깅 및 모니터링 | 연계 이력 추적 및 시스템 간 이상 상황 탐지 | | ① 연계 성공/실패 로그 파일 분리 저장② API 호출별 Trace ID 로깅③ Kafka 메시지 송수신 로그 저장④ 배치 파일 수신/처리 시간 기록⑤ OpenTelemetry 기반 Trace 구성⑥ Prometheus+Grafana 모니터링 대시보드⑦ 인터페이스 SLA 실시간 상태판 구성⑧ 관리자 경고 알림 시스템 연동⑨ ELK Stack 통한 로그 조회 기능 제공⑩ 비정상 연계 자동 장애 접수 처리 | |
|  | 8. 보안 및 접근 제어 | 외부/내부 연계 시 인증 및 보안 정책 적용 | | ① API Key 기반 접근 제어② JWT 기반 사용자 인증 관리③ OAuth2.0 기반 토큰 관리④ TLS 기반 암호화 통신⑤ 연계 대상 시스템 IP 화이트리스트 적용⑥ 외부 연계 시 접근 로그 저장⑦ 사용자 별 호출 이력 추적⑧ 2FA 인증 적용 가능성 검토⑨ 권한 Role에 따른 API 호출 제한⑩ 연계 대상 시스템 보안 점검 및 인증 절차 마련 | |
|  | **설계 요소** | | **설명** | | **세부 설계 요소 (10개 이상)** |
| Interface Layer | **1. API Gateway 구성** | | RESTful API 기반 실시간 요청/응답 처리, 통합데이터 AP가 중심이 되는 연계 허브 | | ① Spring Cloud Gateway or Kong 도입② REST API 기반 요청 분기③ JWT 기반 인증/인가 처리④ 외부 연계 시 OAuth2 기반 Keycloak 인증⑤ URI 기반 버전 관리⑥ 대외 MCA를 통한 외부 API 연계 수신⑦ 지연 응답 처리 및 Timeout 구성⑧ 통합 로그 추적용 Trace ID 전달⑨ API 요청 이력 로깅⑩ 오류 발생 시 Fallback 처리 전략 |
|  | **2. 외부 시스템 연계 처리** | | 계열사 → 대외 MCA → MFT → FOS → ETL의 배치 구조와 API 실시간 연계 구조 포함 | | ① MFT 기반 파일 수신 (FOS 연계)② Control-M 통한 정기적 배치 수행③ XML/CSV 정형 파일 처리④ 파일 유효성 체크 및 백업⑤ FOS 수신 완료 알림 전송⑥ 계열사별 인터페이스 정의서 표준화⑦ HTTP + JSON 기반 실시간 전송⑧ 대외 API 인증 Key 확인⑨ 응답 지연시 재시도 구성⑩ 계열사별 IP/접속 제어 정책 적용 |
|  | **3. 내부 시스템 연계 구조** | | 통합데이터 AP → 통합 데이터 DB, OLAP, e-마케팅, BI 포털 등 내부 실시간 처리 | | ① JDBC 기반 DB 연계 (Oracle, MySQL)② MSTR, Tableau 등 리포트 호출 API 연계③ OLAP 분석 요청 결과 처리④ 통합고객번호 기준 마트 조회⑤ 마케팅 시스템 대상 고객 제공⑥ BI 포털 사용자별 메타 조회⑦ e-마케팅 대상 데이터 반환 API 구성⑧ 연계 서비스별 SLA 분리 적용⑨ 내부 전용 토큰 인증 구조⑩ 서비스 장애 발생 시 Graceful Degradation |
|  | **4. Message Queue 기반 비동기 처리** | | Kafka 기반 분석 연계 및 실시간 ML 모델 결과 처리 | | ① Kafka 기반 Topic 설계 (업무별)② ML/DL 결과 → 마케팅 시스템 전송③ 분석 결과 Stream 처리 후 API 연동④ 메시지 TTL 설정⑤ Kafka Connect CDC 적용⑥ 메시지 소비자 재처리 정책⑦ OLAP 분석 요청 → 결과 Event 수신⑧ Kafka 기반 메타 동기화 처리⑨ Dead Letter Queue 구성⑩ 메시지 기반 이력 로그 적재 |
|  | **5. 배치 연계 설계** | | ETL 기반 정제/파기/이력 로깅 중심 처리 | | ① DataStage 기반 ETL 수행② 계열사 → MFT 통한 파일 수신③ 수신 후 전처리 프로세스 수행④ 파일명 규칙 기반 정합성 체크⑤ 수집 이력 로깅 저장소 구성⑥ 개인정보 포함 파일 자동 암호화/해제⑦ 수집 후 정제 마트 적재⑧ 고객 동의 철회 정보 배치 반영⑨ 파기 대상 자동 삭제 정책⑩ 일자별 배치 수행 결과 모니터링 |
|  | **6. 실시간 인터페이스 설계** | | 통합 데이터 AP를 통한 실시간 요청/응답 인터페이스 설계 | | ① 마케팅 시스템 → 고객 정보 실시간 제공 API② OLAP 시스템 → 통합 데이터 질의 API③ 포털 → 메타데이터 요청 API④ 외부 시스템 → 통합고객번호 기반 이력 조회⑤ e-마케팅 → 반응 정보 수신 API⑥ JWT 기반 토큰 검증⑦ OAuth2 인증 요청 흐름⑧ Keycloak 통합 인증 관리⑨ 응답 SLA 기준 체크⑩ API 호출 이력 로그 저장 |
|  | **7. 인증/인가 및 보안** | | 모든 외부·내부 인터페이스 보안을 위한 정책 적용 | | ① API Gateway에 JWT 필터 적용② API 별 권한 역할(RBAC) 적용③ Keycloak SSO 연계 인증④ TLS 기반 전송 구간 암호화⑤ Token 만료 정책 구성⑥ API 호출 IP 제한 정책⑦ Token 유효성 검증 필터⑧ OAuth2 연동 시 Refresh Token 적용⑨ 외부 연계 시 계약 기반 Key 발급⑩ 보안 감사 로그 적재 및 열람 UI |
|  | **8. 연계 흐름 이력 및 로깅** | | 모든 연계에 대한 로그/추적 정보 저장 및 분석 | | ① 연계별 성공/실패 구분 저장② 연계 ID 기반 로깅 필터 구성③ 인터페이스별 모니터링 UI 제공④ 연계 흐름 분기 경로 추적 가능⑤ Kafka Log 분리 저장⑥ 요청/응답 Payload 샘플 저장⑦ 장애 발생 시 Trace 로그 제공⑧ Elasticsearch + Kibana 연동⑨ 연계 요청 상세 타임라인 기록⑩ SLA 기준 초과 시 경고 메일 발송 |

#### ✅ Interface Layer 설계요소 정리 (연계 흐름도 기반 반영)

|  |  |  |  |
| --- | --- | --- | --- |
| **설계 항목** | **설명** | **주요 연계 대상 및 흐름** | **세부 기술 요소** |
| **1. 대외 연계 게이트웨이** | 계열사 및 외부기관 연계 담당. 대외 MCA 및 FOS 통해 수신/송신 처리 | 계열사 Legacy 시스템 → 대외 MCA → FOS → MFT → ETL | - MFT 기반 수신 처리 - 파일 유효성 검증 - 동의 철회 연계 처리 분기 - 계열사별 파일명 규칙 정의 - 인증 Key 인증 처리 - 파일 이중화 보관 |
| **2. 실시간 API 연계 (내부/외부)** | HTTP + REST 기반 실시간 트랜잭션 처리 | 계열사 시스템 → 대외 MCA → 통합 데이터 AP 마케팅 시스템, 포털, OLAP → API 요청 | - SpringBoot 기반 REST API - JWT 인증 필터 - OAuth2 인증 연계 (Keycloak) - 응답 시간 SLA 설정 - 요청별 Trace ID 구성 - Swagger 기반 명세 정의 |
| **3. 내부 연계 허브(AP)** | 내부 시스템과 통합 데이터 DB간 연계 허브 역할 수행 | 통합 데이터 AP → 통합 데이터 DB, OLAP, e-마케팅, BI 포털, 포털 등 | - API 요청 기반 데이터 마트 조회 - 통합 고객번호 기반 식별 - 응답 데이터 JSON 변환 - 데이터 필터/정제 가공 처리 - DB 접근 제어 로직 포함 - API 이력 저장 |
| **4. 비동기 메시징 연계** | Kafka 기반 Stream 처리, ML 결과 수신 및 마케팅 시스템으로 분배 | 빅스퀘어 → Kafka → E-마케팅, 마케팅 HUB | - Kafka Topic 다중 구성 - Kafka Connect CDC 연계 - Dead Letter Queue 구성 - 이벤트 재처리 로직 - Partition 수직 확장 설계 - Stream 처리 후 API 연동 |
| **5. JDBC 기반 연계** | 직접적인 데이터베이스 연계로 메타, 결과 등 연동 처리 | BI 포털, 포털, OLAP → 통합 데이터 DB | - JDBC Connection Pool 구성 - Connection Failover 구성 - DB 접속 권한 분리 - SQL 튜닝 기반 조회 최적화 - View 또는 Materialized View 활용 - 페이징 처리 및 필터링 |
| **6. 배치 연계 처리** | ETL 및 배치 파일 기반의 주기적 수집 및 적재 처리 | ETL ↔ DB, E-마케팅, HUB, 계열사 | - Control-M 기반 스케줄 관리 - File Polling 및 수신 알림 처리 - 파일명 Timestamp 관리 - 수신 이력 저장 및 백업 - 오류 발생시 Failover 처리 - 파일 암호화 해제 자동화 |
| **7. 인증/보안 모듈 연계** | Keycloak 기반 인증 및 권한 정책, 통합 인증 API 구성 | 모든 외부/내부 API 접근 제어 | - JWT 발급/검증 - OAuth2 인증 흐름 - 사용자 Role 기반 권한 분리 - 인증 로그 저장 - 통합 API Gateway에서 필터 처리 - 인증 토큰 만료 정책 |

##### ✳️ 참고 도식 반영 요약

|  |  |  |
| --- | --- | --- |
| 구간 | 흐름 요약 | 연계 방식 |
| 계열사 → MCA → FOS → ETL | 파일 수신 배치 기반 | FILE + MFT |
| 계열사 → MCA → AP | 실시간 연계 기반 | HTTP + API |
| ETL → 통합 데이터 DB | 정제/로딩/로깅 | JDBC |
| AP → OLAP/e마케팅/포털 | 실시간 API | REST API |
| 빅스퀘어 → Kafka | ML 분석 결과 스트림 연계 | Kafka Stream |

#### ✅ [1] ****온라인 처리 흐름도**** (실시간 HTTP / API 기반)

[계열사 시스템]

│

├─▶ [대외 MCA (API Gateway)]

│ └─ 인증 및 라우팅 처리 (OAuth2, JWT)

│

└─▶ [통합 데이터 AP (Spring Boot API)]

├─ 통합 고객번호 기반 조회

├─ 캠페인 대상 필터링

├─ 분석 결과 실시간 제공

├─ 인증/권한 검증

└─ API 응답 전달 (JSON)

│

├─▶ [통합 데이터 DB (Oracle HEX)]

├─▶ [e-마케팅 시스템]

└─▶ [OLAP 시스템]

[BI 포털 Proxy WEB]

└─▶ API 호출 → [통합 데이터 포털] → JDBC → [통합 데이터 DB]

[계열사 마케터/분석가]

└─▶ [BI 포털 Proxy WEB] → [통합 데이터 포털 UI] → API 호출

#### ✅ [2] ****배치 처리 흐름도**** (File, MFT, Control-M 기반 주기적 처리)

less

복사편집

[계열사 Legacy 시스템]

├─ File 생성 (거래정보, 고객정보 등)

├─▶ [대외 MCA (MFT)]

│ └─▶ [FOS → 내부 전달]

│

└─▶ [통합 데이터 ETL (DataStage)]

├─ 파일 수신 및 유효성 검증

├─ 정제/적재/로깅/파기 처리

└─ JDBC → [통합 데이터 DB]

[빅스퀘어 분석시스템]

├─ File Export (분석결과, ML 예측값)

└─▶ MFT / Kafka Stream

├─▶ [ETL → DB 적재]

└─▶ [마케팅 HUB / e-마케팅] (Kafka 실시간 연계)

[OLAP 시스템]

├─ 배치 파일로 마케팅 분석결과 전송

├─▶ [MFT] → [e-마케팅 / HUB]

└─ JDBC 조회로 통합 데이터 DB 분석 수행

[통합 데이터 ETL]

├─ 주기적 스케줄 (Control-M)

├─ 정제된 마트 데이터 생성

├─ 이력 로깅 및 오류 처리

└─▶ 타겟 시스템: DB, e-마케팅, HUB

[마케팅 시스템 / HUB]

├─ ETL 결과 수신 (MFT)

└─ 고객 타겟팅 결과 활용 (파일 or API 기반 연계)

##### 설계 전략적 시사점 요약

|  |  |  |
| --- | --- | --- |
| 분류 | 주요 특징 | 전략적 의미 |
| **온라인 연계** | API 기반 실시간 처리, 인증/권한 필터 적용 | 사용자/계열사 요청에 대한 실시간 응답 품질 확보 |
| **배치 연계** | 파일 기반 대용량 처리, 안정적 수집/적재 | 비동기 대량데이터 유입에 대한 확장성과 유연성 제공 |
| **중앙 AP 서버** | 실시간 라우팅 중심, 통합 인증 및 데이터 허브 역할 | 통합된 서비스 API 제공 및 내부 시스템 간 브릿지 |
| **ETL 서버** | 정제·로딩·적재·파기 일괄 수행 | 배치 중심 데이터 흐름 컨트롤 포인트 역할 수행 |
| **Kafka Stream** | 분석 시스템 연계에 최적화 | ML 기반 자동 마케팅 대응 체계 구성 가능 |

#### ****인터페이스 상세 설계서 (Interface Specification Sheet)****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 구분 | 송신 시스템 | 수신 시스템 | 연계 방식 | 전송 형식 | 주기 | 주요 처리 내용 |
| 1 | 계열사 Legacy | 통합 데이터 ETL | MFT (FILE) | 고정폭 CSV | 일 1회 | 계열사 고객 거래정보 일괄 수집 및 정제 |
| 2 | 계열사 Legacy | 통합 데이터 AP | API (HTTP) + 대외MCA | JSON | 실시간 | API 요청을 통한 실시간 고객정보 제공 |
| 3 | 은행 계정계 | 통합 데이터 ETL | 전문 (TCP/IP) + EAI | 고정 포맷 | 실시간 | CI 기반 고객 통합번호 매핑정보 실시간 연계 |
| 4 | 빅스퀘어 | 통합 데이터 ETL | MFT (FILE) | CSV/JSON | 1일 4회 | 머신러닝 예측결과/행태 데이터 정제 적재 |
| 5 | 빅스퀘어 | 마케팅 시스템 | Kafka Stream | JSON | 실시간 | 실시간 추천정보 전송 (Stream 기반) |
| 6 | OLAP | 통합 데이터 DB | JDBC | SQL | 수시 | OLAP 질의 결과 연동 및 분석데이터 활용 |
| 7 | 통합 데이터 ETL | 통합 데이터 DB | JDBC | SQL | 배치 | 수집 데이터 적재 및 마트 생성 |
| 8 | 통합 데이터 ETL | 마케팅 HUB / e-마케팅 | MFT (FILE) | CSV | 배치 | 마케팅 대상자 정보 배포 |
| 9 | 통합 데이터 AP | OLAP, 포털, e-마케팅 | API (REST) | JSON | 실시간 | 실시간 정제된 데이터 조회 서비스 제공 |
| 10 | 통합 데이터 포털 | 통합 데이터 DB | JDBC | SQL | 실시간 | UI 기반 메타/분석 정보 조회 |
| 11 | BI 포털 Proxy WEB | 통합 데이터 포털 | HTTP API | JSON | 실시간 | 사용자 요청에 따른 대시보드 API 연계 |
| 12 | OLAP | 마케팅 HUB | MFT (FILE) | CSV | 배치 | 분석 결과를 활용한 캠페인 타깃 전송 |
| 13 | 마케팅 시스템 | 올원뱅크 / UMS | API (HTTP) | JSON | 실시간 | 마케팅 정보 기반 사용자 알림/푸시 발송 |

##### 연계 방식 설명

* **MFT (Managed File Transfer)**: 대용량/정기 전송을 위한 보안 파일 송수신 (보통 CSV 또는 고정폭 포맷)
* **API (HTTP/REST)**: 실시간 통신에 적합, JSON 기반 데이터 요청/응답 처리
* **Kafka Stream**: 비동기 이벤트 기반 실시간 메시지 스트리밍
* **JDBC**: RDB 직접 접근, OLAP 또는 포털에서 SQL 기반 데이터 질의 처리
* **전문 (EAI)**: 시스템 간 고속 연계용 TCP 기반 전문 연동

##### 전략적 시사점

|  |  |
| --- | --- |
| 항목 | 시사점 |
| 실시간 처리 API | 통합 데이터 AP는 다양한 시스템에 REST API로 실시간 응답 |
| 배치 안정성 확보 | 정기 파일 수신은 MFT 기반으로 이중화 및 오류 복구 전략 필수 |
| 이벤트 기반 연계 | Kafka Stream을 통해 분석/추천 서비스의 실시간 대응 가능 |
| 데이터 품질 | ETL에서 오류 정제 및 이력 로깅 체계를 통해 추적성과 신뢰도 확보 |
| 인증 보안 | API 연계 시 OAuth2 및 JWT 기반 인증 체계 적용 필수 |

## ****마무리말****

어플리케이션 계층 구조 설계는 성능, 확장성, 유지보수성을 고려한 아키텍처 구성의 핵심입니다. 각 계층을 독립적으로 설계함으로써 모듈화와 재사용성을 극대화하고, 채널 계층과 비즈니스 계층의 역할을 명확히 구분하여 성능 저하를 방지합니다. 특히, API Gateway와 Message Queue를 활용하여 MSA 기반의 확장성을 확보하며, 데이터 계층에서는 트랜잭션 관리와 Connection Pool 최적화를 통해 고가용성을 유지합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **Channel Layer** | 사용자 중심의 인터페이스를 제공하며 다양한 디바이스 환경에 대응할 수 있다. |
| **Interface Layer** | 외부 시스템과의 연계가 용이하며, 부하 분산 및 API 관리가 체계적으로 가능하다. |
| **Business Layer** | 도메인별로 비즈니스 로직을 분리하여 유지보수성과 성능을 동시에 확보할 수 있다. |
| **Data Layer** | 데이터 접근과 트랜잭션 관리가 명확하여 대용량 데이터 처리 시 안정성을 보장한다. |
| **MSA 구조** | 서비스 간 독립성을 유지하여 장애 발생 시 서비스 연속성을 확보할 수 있다. |
| **API Gateway 활용** | API 집합 관리와 인증 정책을 한 곳에서 관리하여 보안성을 극대화할 수 있다. |

# ****주요 설계 패턴 정의****

## ****5.1 도입전안내말****

설계 패턴은 어플리케이션 아키텍처의 일관성과 최적화를 보장하기 위한 기술적 기반입니다.  
EA(Enterprise Architecture) 관점에서 설계 패턴을 명확하게 정의하면 비즈니스 로직의 복잡성을 줄이고, 성능 최적화와 확장성 확보에 크게 기여합니다.  
특히, Microservice Architecture(MSA), Monolithic Architecture, CQRS, Event-driven Architecture, Saga Pattern은 시스템의 확장성과 분산 처리에서 중요한 역할을 합니다.

## 주요 설계 패턴 설계

|  |  |  |  |
| --- | --- | --- | --- |
| **패턴명** | **주요 역할** | **핵심 기술 스택** | **전략적 의미** |
| **5.1 Microservice Architecture (MSA)** | 단일 업무 기능을 독립적인 마이크로서비스로 분리하여 유연한 확장과 독립적 배포가 가능하게 함 | Spring Boot, Docker, Kubernetes, Spring Cloud | 시스템 간 결합도를 낮추어 장애 발생 시에도 전체 서비스 중단 없이 독립적 복구가 가능함 |
|  | - 5.1.1 독립 배포 구조 | Docker, Helm Charts | 각 마이크로서비스가 독립적으로 배포되어 개발 주기 단축 |
|  | - 5.1.2 확장성 및 장애 격리 | Kubernetes, Service Mesh | 서비스 간 장애 전파를 차단하고, 확장성을 극대화할 수 있음 |
| **5.2 Monolithic Architecture** | 모든 기능이 단일 애플리케이션 내에 통합된 구조로, 간단한 관리와 일관된 트랜잭션 처리가 가능함 | Spring Framework, Hibernate, Oracle DB | 중앙 집중형 트랜잭션 관리와 단순한 배포 구조를 가지며, 레거시 시스템과의 통합이 용이함 |
|  | - 5.2.1 단일 배포 구조 | Spring Boot | 코드 일관성이 높고 모니터링이 쉬우며, 유지보수가 용이함 |
|  | - 5.2.2 중앙 집중형 처리 | Oracle, PostgreSQL | 일관성 있는 트랜잭션 처리와 관리가 가능함 |
| **5.3 CQRS (Command Query Responsibility Segregation)** | 읽기(QUERY)와 쓰기(COMMAND) 모델을 분리하여 성능 최적화 및 확장성을 극대화 | Spring Data JPA, MongoDB, Kafka | 데이터 읽기와 쓰기를 분리하여 성능 최적화, 실시간 분석 처리에 강점 |
|  | - 5.3.1 읽기 최적화 | MongoDB, Redis | 대량 데이터 조회 시 높은 성능을 발휘하며, 읽기 전용 DB로 분리 가능 |
|  | - 5.3.2 쓰기 최적화 | PostgreSQL, Oracle | 트랜잭션 관리가 필요한 복잡한 비즈니스 로직의 쓰기 처리에 최적화됨 |
| **5.4 Event-driven Architecture** | 비동기 이벤트 기반으로 독립된 컴포넌트 간 통신을 지원하여 확장성과 유연성을 극대화 | Kafka, RabbitMQ, Spring Cloud Stream | 비동기 처리로 성능 최적화와 확장성 확보, 서비스 간 느슨한 결합 구조 형성 |
|  | - 5.4.1 이벤트 소싱 (Event Sourcing) | Kafka, Event Store | 상태 변경 이벤트를 지속적으로 저장하여 장애 복구 및 이력 추적이 용이함 |
|  | - 5.4.2 이벤트 브로커 | RabbitMQ, ActiveMQ | 이벤트 전송을 중앙에서 관리하여 데이터 일관성 보장 및 확장성 확보 |
| **5.5 Saga Pattern** | 분산 트랜잭션 처리를 위한 패턴으로, 각 서비스 간 보상 트랜잭션을 통해 일관성을 유지함 | Kafka, Spring Cloud Sleuth, Saga Coordinator | MSA 환경에서 트랜잭션 관리가 필요한 복합 업무 처리 시 안정성을 높임 |
|  | - 5.5.1 분산 트랜잭션 관리 | Spring Cloud Sleuth, Kafka Streams | 다중 서비스 간 트랜잭션 처리 시 보상 트랜잭션을 활용하여 데이터 일관성 유지 |
|  | - 5.5.2 보상 트랜잭션 관리 | Saga Coordinator, Distributed Lock | 실패 시 보상 작업을 자동으로 처리하여 데이터 무결성 확보 |

### ****(실제) 주요 설계 패턴 ( NH 농협지주 데이터 플랫폼 구축)****

|  |  |  |  |
| --- | --- | --- | --- |
| 패턴명 | 주요 역할 | 핵심 기술 스택 | 전략적 의미 |
| 5.1 Microservice Architecture (MSA) | 업무별 서비스 분리, 독립 배포 및 무중단 운영 지원 | Spring Boot, Docker, K8s, Spring Cloud | 각 시스템 간 결합도 최소화로 장애 전파 방지 및 독립적 확장 가능 |
| └ 5.1.1 독립 배포 구조 | 서비스 단위 Docker 이미지 분리, Helm 배포 자동화 | Docker, Helm Charts | 서비스별 롤링 배포 및 무중단 업데이트 지원 |
| └ 5.1.2 확장성 및 장애 격리 | Kubernetes Horizontal Pod Autoscaling | K8s, Service Mesh | 트래픽 증가 시 자동 확장, 장애 범위 격리 |
| 5.2 Monolithic Architecture | 일관성 유지가 필요한 핵심 트랜잭션 시스템 통합 | Spring Framework, Hibernate, Oracle | 마이그레이션 대상이 아닌 계정계/레거시 시스템 중심 안정 처리 |
| └ 5.2.1 단일 배포 구조 | 서비스 통합 및 단일 배포 파이프라인 유지 | Spring Boot | 코드 일관성과 모니터링 효율성 확보 |
| └ 5.2.2 중앙 집중형 처리 | 트랜잭션 기반 업무처리의 신뢰성 확보 | Oracle RAC, Hibernate | 단일 DB 트랜잭션 처리의 안정성 확보 |
| 5.3 CQRS (Command Query Responsibility Segregation) | 읽기/쓰기 분리로 성능 분산 및 실시간 분석 대응 | Spring JPA, Redis, Kafka | 분석성과 트랜잭션을 분리하여 시스템 부하 감소 및 속도 개선 |
| └ 5.3.1 읽기 최적화 | Redis 캐시 기반 분석 화면 응답 최적화 | Redis, MongoDB | OLAP용 쿼리 응답 성능 극대화 |
| └ 5.3.2 쓰기 최적화 | 정합성이 요구되는 API에선 PostgreSQL, Oracle | JPA, Hibernate | 트랜잭션 보장 기반 실시간 데이터 쓰기 처리 |
| 5.4 Event-driven Architecture | 비동기 메시지 처리 구조 | Kafka, RabbitMQ, Spring Cloud Stream | 마케팅 시스템, OLAP, 외부연계와의 실시간 이벤트 반응 |
| └ 5.4.1 이벤트 소싱 | 이벤트 기반 상태 변경 관리 및 이력 로그 | Kafka, Event Store | 장애 대응 및 분석용 이벤트 로그 추적 가능 |
| └ 5.4.2 이벤트 브로커 | 이벤트 큐 분리 및 구독 기반 소비 처리 | Kafka Topic, RabbitMQ | 시스템 간 느슨한 결합 및 확장 용이 |
| 5.5 Saga Pattern | 분산 환경에서의 트랜잭션 정합성 확보 | Spring Cloud Sleuth, Kafka, Saga Coordinator | 비동기 처리로 인한 데이터 일관성 유지를 위한 전략 |
| └ 5.5.1 분산 트랜잭션 관리 | 업무 흐름 단위의 Saga 구성 | Kafka Streams, Distributed Lock | 지연된 보상 트랜잭션 기반 신뢰성 유지 |
| └ 5.5.2 보상 트랜잭션 처리 | 실패 시 보상 처리 플로우 구현 | Saga Coordinator | 장기 실행 프로세스의 무결성 확보 |
| 5.6 API Gateway Pattern | 서비스 엔트리 포인트 통합 및 보안 정책 적용 | Kong, Zuul, Spring Gateway | 인증/인가 처리, 트래픽 분산, 외부 연동 통제 |
| └ 5.6.1 API 라우팅 및 인증 | OAuth2 + JWT 기반의 토큰 인증 통합 | Keycloak, OAuth2 Proxy | 사용자/시스템 인증에 대한 보안정책 강화 |
| └ 5.6.2 요청 필터링 및 버전관리 | API Throttling, Request Filtering, Rate Limiting | Kong Plugin, Spring Filters | 시스템 과부하 방지 및 지속가능한 연계 유지를 위한 구조 |
| 5.7 Adapter Pattern *(통합 연계용)* | 레거시와의 데이터 및 프로토콜 변환 | EAI, FEP Adapter, Mapper | FEP, 대외 MCA 연계를 위한 포맷 변환 처리 |
| └ 5.7.1 포맷 변환 어댑터 | XML ↔ JSON, Fixed ↔ Delimited | Apache Camel, FEP | 배치 연계 및 전문 연계 표준화 대응 |
| └ 5.7.2 인터페이스 버전 대응 | Interface Version Routing | Gateway Version Header | 레거시와 최신 API간 버전 충돌 최소화 |

|  |  |  |  |
| --- | --- | --- | --- |
| 패턴명 | 주요 역할 | 핵심 기술 스택 | 전략적 의미 |
| Microservice Architecture (MSA) | 서비스 경량화 및 분리 | Spring Boot, Spring WebFlux | 기능 단위 분리로 독립적 배포 및 확장성 확보 |
|  | 서비스 오케스트레이션 | Kubernetes, Helm, Istio (Service Mesh) | 서비스 연결 구조 안정화, 서비스 간 통신 보안 및 가시성 강화 |
|  | 컨테이너 기반 배포 | Docker, OpenShift, Tanzu | 신속한 배포 및 환경 간 일관성 유지 |
| Monolithic Architecture | 일괄 트랜잭션 처리 | Spring MVC, Hibernate, Oracle | 업무 일관성 및 성능 안정성 확보 |
|  | 레거시 호환 관리 | Oracle HEX, 기존 DB 연계모듈 | 기존 시스템과의 통합 용이성 보장 |
| CQRS | 읽기 서비스 분리 | MongoDB, Redis, Elasticsearch | 대용량 분석 성능 향상, 마케팅 시나리오 최적화 |
|  | 쓰기 서비스 트랜잭션 | PostgreSQL, Oracle | 데이터 정합성 확보, 복잡 로직 처리에 적합 |
|  | 비동기 연계 처리 | Kafka, RabbitMQ, Kafka Streams | 실시간 메시징, 분산 처리 안정성 강화 |
| Event-driven Architecture | 이벤트 기반 비동기 처리 | Kafka, Spring Cloud Stream | 트랜잭션 병렬 처리 및 시스템 응답성 향상 |
|  | 이벤트 브로커 관리 | RabbitMQ, Kafka Connect | 메시지 전달 보장 및 대규모 처리 확장성 |
|  | 로그 기반 아키텍처 | Kafka Log Compaction, Filebeat | 데이터 변경 이력 관리 및 감사 이력 확보 |
| Saga Pattern | 보상 트랜잭션 관리 | Spring Boot + Saga Coordinator, Event Store | 분산 시스템 간 트랜잭션 정합성 보장 |
|  | 트랜잭션 흐름 추적 | Spring Sleuth, Zipkin, OpenTelemetry | 장애 발생 시 빠른 복구와 흐름 추적 가능 |
| API Gateway Pattern | 인증 및 인가 통합 | Keycloak, OAuth2, JWT | 통합 인증 체계 구축, 외부 연계 보안 강화 |
|  | API 트래픽 제어 | Kong, Zuul, Spring Cloud Gateway | 라우팅, 속도 제어, 버전 관리 등 API 정책 수립 용이 |
|  | 외부 연계 통합 처리 | RESTful API, OpenAPI, Swagger | 다양한 소비 주체 대응 및 연계 문서 자동화 |
| Adapter Pattern | 표준화된 연계 포맷 관리 | Apache Camel, MFT, EAI, Mapper Engine | 타 시스템 간 데이터 구조 및 포맷 통합 |
|  | 외부 채널 연계 | 대외MCA, FEP, OpenAPI Gateway | 대외기관/계열사 연계 통신의 안정성과 신뢰성 확보 |
| Event Sourcing | 변경 이력 저장 | Kafka Event Log, Event Store, Debezium | 상태 변경 이벤트 기반 이력 관리 및 복구 |
|  | 감사 및 재처리 | Change Data Capture (CDC), Kafka Replay | 이력 분석 및 장애 시 재처리 메커니즘 구현 |
| Circuit Breaker & Retry Pattern | 장애 전파 차단 | Resilience4j, Hystrix | 외부 시스템 장애 발생 시 자체 보호 및 대기열 관리 |
|  | 재시도 및 Fallback 처리 | RetryTemplate, Spring Retry | 일시적 장애 대응을 통한 사용자 경험 유지 |
| Command Pattern | 복잡 요청 흐름 처리 | Command Bus, Command Dispatcher | 요청 흐름 구조화 및 로깅 추적 가능 |
| Bulkhead Pattern | 리소스 분리 | Kubernetes Pod QoS, Thread Pool Isolation | 핵심 서비스 보호 및 격리로 시스템 안정성 확보 |
| 패턴명 | **주요 역할** | **핵심 기술 스택** | **전략적 의미** |
| Batch Processing Pattern | 대량 데이터 수집 및 적재 | IBM DataStage, Spring Batch, Control-M | 정기적인 계열사 데이터 수집, 정제 및 적재에 적합한 프레임워크를 통한 고신뢰 대용량 처리 |
|  | 정제 및 데이터 품질 검증 | 데이터 전처리 파이프라인, Rule Engine, Python (Pandas) | 수집 데이터에 대한 유효성 검증, 이상치 제거, 마스킹/암호화 처리 |
|  | 배치 트랜잭션 관리 | Spring Transaction Manager, XA Transaction | 데이터 일관성을 위한 안정적 트랜잭션 처리 및 복구 전략 설계 |
|  | 로깅 및 이력 관리 | Filebeat, Audit Log DB, ELK Stack | 수집 및 적재 프로세스에 대한 이력 로깅과 감사 추적 가능 구조 확보 |
|  | 파일 기반 수집 연계 | MFT 솔루션, SFTP, FOS File System | 계열사 Legacy 시스템과의 연계 안정성을 높이는 표준 전송 방식 사용 |
|  | 스케줄 기반 실행 관리 | Control-M, Airflow, Spring Scheduler | 배치 프로세스의 예약, 모니터링, 장애 대응을 위한 스케줄링 체계 구현 |
|  | 장애 대응 및 복구 | Retry Policy, Job Restart, Dead Letter Queue | 배치 실패 시 자동 재처리 및 보상 트랜잭션 처리로 무중단 운영 보장 |
|  | 데이터 파기 및 보관 주기 관리 | ILM(Information Lifecycle Management), ETL 정책 | 개인정보 및 만료 데이터에 대한 보관 정책 기반 파기 아키텍처 구성 |
|  | 로그 기반 재처리 | Change Tracking Log, CDC 연계 | 로그 기반 장애 분석 및 선택적 재처리를 통한 운영 비용 절감 |
|  | 파일 메타정보 관리 | Metadata Repository, Meta File Descriptor | 수집 파일의 구조, 필드 정의, 암호화 여부 등 메타정보 추적 관리 가능 |

#### **온라인 처리 패턴 기반 실제 흐름도**

**NH 농협지주 데이터 플랫폼** 구축 사업에서 활용되는 **온라인 처리 패턴 기반 실제 흐름도**를 **텍스트 기반 도식**으로 정리한 것입니다. 이 흐름도는 **실시간 API 요청, 응답, DB 처리, 인증 흐름**을 포함하여 실제 서비스 운영 기준에 따라 구성되었습니다.

|  |
| --- |
| [계열사 사용자 / 외부 시스템 요청]  └─▶ Step 1. API 호출 (HTTP/HTTPS + JSON)  └─▶ [인터넷망 / 전용선]  └─▶ [은행 대외 MCA (API Gateway)]  └─▶ Step 2. 인증 처리 (OAuth2 / JWT / Keycloak)  └─▶ Step 3. API 라우팅 (URI/메소드/토큰기반)  └─▶ [지주 통합 데이터 AP (Spring Boot)]  ├── Step 4. 요청 유효성 검증  ├── Step 5. 비즈니스 로직 처리  │ └─ Rule 엔진 적용 (필터링, 조건 처리)  │ └─ 마트별 데이터 조회 기준 로직 수행  ├── Step 6. 트랜잭션 처리  │ └─ ACID 적용, 동시성/락 전략 적용  ├── Step 7. DB 조회 / 저장  │ └─ JDBC / MyBatis / JPA 연동  │ └─ Oracle HEX / 정제 마트 접속  ├── Step 8. 응답 데이터 가공  │ └─ JSON or XML 포맷 변환  └── Step 9. 응답 반환  └─▶ API Gateway → 호출 시스템 |
| **주요 구성 기술 요약**   |  |  |  |  | | --- | --- | --- | --- | | 단계 | 처리 영역 | 주요 기술 스택 | 설명 | | 1 | API 호출 | HTTP(S), JSON | 계열사 UI 또는 외부 시스템이 API 요청 | | 2 | API Gateway | OAuth2, JWT, Keycloak, MCA | 인증, 인가 및 라우팅 수행 | | 3 | 통합 데이터 AP | Spring Boot, REST Controller | 요청을 처리하는 중심 Application | | 4 | 유효성 검사 | Bean Validation, XSS 필터 | 파라미터 검증, 보안 검증 | | 5 | 비즈니스 로직 | Rule Engine, Java Service | 마케팅 타깃 필터링, 고객 데이터 조합 | | 6 | DB 연동 | JPA, MyBatis, JDBC | Oracle HEX 기반 데이터 마트 조회 | | 7 | 트랜잭션 처리 | Spring TX Manager | 데이터 정합성 보장 | | 8 | 응답 구성 | Object Mapper, XML Parser | JSON/XML 포맷 변환 후 응답 | |
|  |

#### ****온라인 기반 트랜잭션 구조****

|  |
| --- |
| [계열사 사용자 or 외부 연계 시스템]  │  [HTTP/HTTPS, REST API]  ▼  ┌─────────────────────────────────────┐  │ 은행 대외 MCA (API Gateway) │  │ - OAuth2 / JWT 인증 │  │ - URI 라우팅, IP/토큰 검증 │  └─────────────────────────────────────┘  │  ▼  ┌─────────────────────────────────────┐  │ 지주 통합 데이터 AP (Spring Boot) │  │ - API Gateway 연계 엔드포인트 │  │ - 비즈니스 로직 처리 (Rule, Facade 등) │  │ - 통합 고객번호 기준 데이터 필터링 │  └─────────────────────────────────────┘  │  ┌──────────────────────┼──────────────────────┐  ▼ ▼ ▼  [통합데이터 포털] [e-마케팅 시스템] [OLAP 시스템]  - BI 시각화 - 마케팅 타겟 - 정제데이터 분석  - 데이터 신청 - 실시간 캠페인 - MSTR 연계  │  ▼  ┌─────────────────────────────────────┐  │ 통합 데이터 DB (Oracle HEX) │  │ - 마트, 원천, 분석 DB 계층별 저장소 │  │ - 트랜잭션 처리, 인덱싱, 튜닝 적용 │  └─────────────────────────────────────┘  ▲  │  ┌─────────────────────────────────────┐  │ 통합 데이터 ETL (DataStage) │  │ - 정제, 적재, 전처리, 이력 로깅 │  │ - 파일 기반 수집 및 연계 │  └─────────────────────────────────────┘  ▲  │  ┌─────────────────────────────────────┐  │ 계열사 Legacy 시스템 / 빅스퀘어 │  │ - 파일 전송 (MFT) + 실시간 Kafka │  │ - 분석 결과 또는 원천데이터 제공 │  └─────────────────────────────────────┘ |

**도식 내 기술 요소 반영**

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 적용 기술 | 비고 |
| 대외 MCA | OAuth2, JWT, HTTP 라우팅 | REST 기반 보안 인증 처리 |
| 통합 데이터 AP | Spring Boot, API Controller | 실시간 트랜잭션 및 비즈니스 로직 처리 |
| 통합 데이터 DB | Oracle HEX | 마트, 분석, 원천 데이터 저장소 |
| ETL 서버 | DataStage, MFT 연계 | 배치 수집 및 정제 적재 수행 |
| OLAP | MSTR, JDBC | 분석 데이터 활용 |
| 통합 데이터 포털 | React, BI 프레임워크 | 분석가용 UI 서비스 |
| 외부 시스템 | Kafka, API, FEP | 실시간 메시징 및 전문 연계 |

#### ****Batch Processing Pattern 기반 실제 흐름도 (텍스트 도식)****

[계열사 Legacy 시스템]

├── Step 1. 정기 파일 생성 (거래정보, 상품정보 등)

└── Step 2. 파일 전송

└─▶ [FOS MFT 시스템]

└─ Step 3. 파일 수신 및 분배

└─▶ [통합 데이터 ETL (DataStage)]

├── Step 4. 파일 수집 모듈 실행

│ └─ 파일명, 경로, 포맷 검증

│ └─ 메타정보(업무구분, 송신자, 날짜 등) 등록

│

├── Step 5. 정제 프로세스 실행

│ └─ 데이터 유효성 검증 (Null 체크, 형 변환 등)

│ └─ 암호화/마스킹 적용

│ └─ 중복 제거 및 룰 기반 필터링

│ └─ 오류/누락 데이터 분리 저장

│

├── Step 6. 데이터 적재

│ └─ Oracle HEX (통합 데이터 DB)에 마트별 저장

│ └─ 적재 대상별 테이블(원천/정제/분석/요약 마트)

│

├── Step 7. 이력 및 로그 기록

│ └─ 처리 결과 로그 (성공/실패/건수 등)

│ └─ Control-M Job 로그 연계 저장

│

├── Step 8. 파기 데이터 처리

│ └─ 보존기간 만료 데이터 삭제

│ └─ ILM 정책 기반 정기 삭제

│

└── Step 9. 배치 완료 통보

└─▶ 마케팅 시스템 / 포털 시스템 / UMS에 통보

##### 📌 주요 구성 기술 요약

|  |  |  |
| --- | --- | --- |
| 처리 단계 | 주요 기술 | 설명 |
| 파일 전송 | FOS MFT | 계열사 → 은행 간 보안 파일 전송 |
| 수집/정제 | IBM DataStage, Shell Script | 배치 수집 및 데이터 전처리 |
| 적재 | Oracle DB (HEX) | 데이터마트 구성 및 적재 |
| 스케줄링 | Control-M | 정기 Job 수행 및 오류 알람 |
| 파기 | ILM 솔루션 or ETL | 보존주기 기준 데이터 자동 삭제 |
| 통보 | REST API / MFT / DB 연계 | 후속 시스템에 상태 통보 |

##### 시사점

* 이 표는 단순 패턴 열거가 아닌, **NH 농협지주 데이터 플랫폼의 도메인-계층-기술적 연계를 고려한 실제 설계 기준**을 반영한 것입니다.
* **MSA, API Gateway, Event 기반 설계, Saga 패턴**은 각각 **마케팅/분석 시스템 연계, 고객 반응 분석, 실시간 추적/제공 처리 구조**에 핵심 적용됩니다.
* 특히, **Adapter Pattern**과 **Retry + Circuit Breaker Pattern**은 계열사 시스템 및 외부 API 연계 안정화의 중추적인 역할을 수행합니다.
* **Batch Processing Pattern**은 NH 농협지주 통합 데이터 플랫폼의 **ETL 시스템, 통합 데이터 DB, 마케팅 시스템** 간 **파일 기반 배치 처리 흐름**을 구현하는 데 필수적입니다.
* 특히, **Control-M 기반 스케줄링**, **DataStage 기반 전처리 로직**, **FOS 기반 MFT 연계**는 프로젝트 전반의 **자동화, 안정성, 감사 이력 관리**에 큰 역할을 합니다.
* 이 설계 패턴은 **Event-driven Architecture** 및 **MSA의 비동기 흐름**과 병렬로 운영되며, **정형/비정형 데이터 처리를 동시에 고려**하는 전략이 필요합니다.
* **보안성 확보**: MFT와 데이터 마스킹 적용으로 금융권 보안 요건 충족
* **신뢰성 강화**: 처리 이력 로깅과 오류 처리로 배치 Job 신뢰도 향상
* **유연성**: 데이터 마트 단위 적재로 다양한 분석 체계 대응
* **확장성**: Rule 기반 정제 및 스케줄링 구성으로 계열사 확대 시 유연 대응
* **저지연 응답 설계**: 트랜잭션 최소화 + 비동기 처리 활용 시 3초 이내 응답 가능
* **API 게이트웨이 기반 보안/인증 구조**: 금융기관 보안 요건 완벽 대응
* **중앙 AP 통합 계층**: 마케팅/OLAP/포털/UMS 요청을 단일 AP로 수렴, 운영 효율성 극대화
* **실시간 데이터 제공 기반**: 타겟 고객 추출, OLAP 조회, 메타정보 제공 등 다양한 연계 가능

#### ****배치 / 온라인 통합 흐름도****

[계열사 Legacy 시스템]

├─ 배치 연계 ─ FILE 전송 (MFT) ─▶ [ETL 서버 (DataStage)]

│ └─ 정제 / 이력 기록 / 암호화

│ └─ JDBC ─▶ [통합 데이터 DB (Oracle HEX)]

│

└─ 온라인 연계 ─ API (HTTP) 호출 ─▶ [대외 MCA]

└─ OAuth2 인증 후 ─▶ [통합 데이터 AP]

└─ Query 또는 REST 요청

└─ JDBC ─▶ [통합 데이터 DB]

[통합 데이터 AP]

├─ API 라우팅 (실시간 요청)

│ ├─▶ 통합 데이터 포털

│ ├─▶ e-마케팅 시스템

│ └─▶ OLAP

└─ 인증 관리 (JWT / OAuth2)

[ETL 서버]

└─ 정기 배치 스케줄

├─ 계열사 파일 수집

├─ 정제 / 적재 / 로깅

└─ 배포 (마케팅 시스템, E-마케팅 등)

[OLAP / BI 포털]

├─ 데이터 마트 및 분석 DB 조회

└─ 분석 결과 파일 MFT 전송 ─▶ [e-마케팅 시스템] / [마케팅 HUB]

#### ****서비스 / 계층 연계도 (계층별 흐름 중심 도식)****

[📱 Channel Layer (UI)] ──────────────▶ [🌐 Interface Layer] ──────────────▶ [💡 Business Layer] ──────────────▶ [💾 Data Layer]

│ │ │ │

│ Web UI (React, BI 포털) │ API Gateway (Zuul, Spring GW) │ FACADE / RULE / SERVICE / DAO │ Oracle / MySQL / ETL Target

│ 모바일 앱 (Flutter 등) │ Message Queue (Kafka, RabbitMQ) │ 비즈니스 모듈 처리 / CQRS 분리 │ 통합 DB, 분석 마트 등

│ 사용자 입력 / 요청 인터페이스 │ 외부 연계 (대외 MCA, API) │ 이벤트 기반 처리 / SAGA 제어 │

│ │ 인증 / 라우팅 / 흐름 제어 │ 분산 트랜잭션 처리 및 데이터 연계 │

특징:

- Channel → Interface: 사용자 요청은 API 또는 MQ로 전달

- Interface → Business: 인증 / 라우팅 후 서비스 호출

- Business → Data: DAO 또는 JPA를 통해 실시간/배치 데이터 접근

- ETL은 별도 흐름으로 Data Layer에 주기적 적재 (Batch)

📌 전략적 시사점

|  |  |
| --- | --- |
| 구분 | 전략적 의미 |
| **배치/온라인 분리** | 정제 데이터는 ETL을 통해 적재되고, 온라인 API는 트랜잭션 요청 처리에 집중되어 시스템 부하를 분산함 |
| **AP 중계 계층** | 통합 데이터 AP는 모든 API 중심 중계/연계 지점으로, 확장성 높은 API 설계가 가능함 |
| **계층 분리 구조** | Layer 간 책임 분리로 모듈화, 유지보수 용이성, 테스트 가능성 향상 |
| **전문 연계 & 파일 연계** | API 기반 요청과 파일 기반 연계가 병행되어, 다양한 시스템 간 호환성 확보 |

#### ****분산 트랜잭션 흐름도 (SAGA 패턴 + 메시지 브로커 기반)****

|  |
| --- |
| [Client UI]  └─▶ [API Gateway (Zuul / Spring Cloud Gateway)]  └─▶ [Service A: 통합 고객 식별 처리]  ├─ 로컬 DB 저장  ├─ 이벤트 발생 → Kafka Publish (고객등록됨)  └─ ▶ [SAGA Coordinator 시작]  [SAGA Coordinator]  ├─▶ [Service B: 마케팅 타겟 필터링]  │ ├─ 필터 정책 기준 적용  │ └─ Kafka 이벤트 (타겟필터완료됨) 발행  └─▶ [Service C: 캠페인 발송 요청]  ├─ UMS 또는 올원뱅크 연계  └─ 성공/실패 응답 수신  [보상 트랜잭션 처리 (실패 시)]  ├─ 고객등록 롤백  ├─ 마케팅 타겟 필터 삭제  └─ 캠페인 발송 취소 처리  ※ 각 서비스는 독립 DB 및 트랜잭션 처리  ※ 이벤트는 Kafka 기반, 비동기 전송  ※ 보상 흐름은 SAGA Coordinator가 제어 |
|  |

#### ****OLAP + BI 분석 흐름도****

|  |
| --- |
| [통합 데이터 DB (Oracle HEX)]  └─▶ [ETL 서버 (DataStage)]  ├─ 정제 및 마트화  └─ 분석용 테이블 적재  [OLAP 시스템 (MSTR / Tableau)]  └─ JDBC 조회 → [정제 마트]  └─ 배치 기반 조회 → [분석 마트]  [BI 포털 (Spring Boot + MySQL)]  ├─ 사용자별 리포트 요청  ├─ 메타데이터 조회 (REST API)  └─ 결과 시각화  [BI 포털 Proxy Web]  └─ OAuth2 인증 후 통합 포털 접근  [계열사 마케터]  └─ 웹 접속 → Proxy → BI 포털 → OLAP 분석 결과 조회 |
|  |

#### ****전체 시스템 계층 형 연결고****

|  |
| --- |
| [📱 Channel Layer]  ├─ 통합단말 UI (xFrame5)  ├─ 통합 데이터 포털 UI (SpringBoot)  ├─ 올원뱅크, 모바일 UI  [🌉 Interface Layer]  ├─ API Gateway (Zuul, Kong)  ├─ 외부 연계 (대외 MCA / FEP / MFT)  ├─ MQ / Kafka Stream 연계  ├─ 인증 연계 (OAuth2 / JWT / Keycloak)  [💡 Business Layer]  ├─ 통합 데이터 AP (Spring Boot)  ├─ 마케팅 시스템 (Spring Boot + Kafka)  ├─ e-마케팅 / 마케팅 HUB  ├─ FACADE / RULE / THING / DAO 구조  [🗂 Data Layer]  ├─ 통합 데이터 DB (Oracle HEX)  ├─ OLAP 분석 마트 (MySQL / PostgreSQL)  ├─ BI 포털 DB (MySQL)  └─ ETL (DataStage), 정제/로깅/파기 처리  [🔧 Infra Layer]  ├─ PaaS 기반 POD: WEB / PT-WAS / BT-WAS  ├─ 컨테이너: Docker + Kubernetes  ├─ 보안: TOS / Deep Security / DB Safer  └─ DevOps: Git, Jenkins, CI/CD, ArgoCD |
|  |

|  |  |
| --- | --- |
| 구분 | 설계 전략적 의미 |
| 분산 트랜잭션 | SAGA 기반의 보상 트랜잭션과 Kafka 이벤트 기반 처리를 통해 서비스 독립성과 데이터 정합성을 확보 |
| OLAP/BI 분석 | 데이터 마트 분리 및 계층화된 리포팅 구조로 고속 분석과 계열사 사용자 대응력을 확보 |
| 전체 계층 구성 | PaaS 기반 구조와 도메인 계층 분리를 통해 시스템 확장성, 유지보수성, DevOps 자동화 대응이 용이 |

#### ****Kafka 이벤트 흐름도****

|  |
| --- |
| [1] 계열사 시스템 (e.g., NH손해보험, NH투자증권 등)  └─▶ 이벤트 발생 (e.g., 고객 마케팅 동의)  └─▶ Kafka Producer (API or ETL 연계)  └─▶ Kafka Topic: [consent.agree.created]  [2] Kafka Broker (클러스터 구성: 3노드 이상, 파티셔닝 지원)  └─▶ Topic 저장 및 분산 전송  └─▶ 장애 대비 복제(replication) 및 메시지 보존 설정  [3] 통합 데이터 AP (Spring Boot + Kafka Consumer)  ├─ Kafka Consumer 그룹 구독: [consent.agree.created]  ├─ 동의 내역 정합성 검증 및 저장 처리  ├─ 통합 데이터 DB (Oracle HEX) 저장  └─ ▶ 이벤트 후속 발행: Kafka Topic [consent.agree.verified]  [4] 마케팅 시스템 (캠페인 대상 추출용)  └─ Kafka Consumer 수신  ├─ 고객 필터링 및 캠페인 타깃 설정  ├─ 내부 마케팅 DB 저장  └─ ▶ Kafka Topic [campaign.target.ready] 발행  [5] e-마케팅 시스템, UMS, 올원뱅크  └─ Kafka 구독 및 실시간 알림 또는 배치 전환 처리  └─ 이벤트 기반 캠페인 송출 및 이력 수집  [6] Kafka Logging & Monitoring (ELK + Prometheus + Grafana)  └─ Topic별 처리 현황 모니터링  └─ 메시지 재처리 / Fail-Over / Retention 분석 |
| Kafka 이벤트 흐름 단계별 기술 전략   |  |  |  | | --- | --- | --- | | 단계 | 이벤트 흐름 | 설계 전략 | | **1** | Producer 구성 | 각 시스템에서 Kafka Producer 구성 (Spring-Kafka, Kafka-Connect 사용 가능) | | **2** | Kafka Broker | 3개 이상 노드로 구성된 클러스터 운영, 파티션 분산 처리, 복제율 최소 2 이상 설정 | | **3** | Consumer 그룹 | 서비스 목적별 Consumer Group 구분, 메시지 유실 방지 위한 offset 저장 방식 사용 | | **4** | 이벤트 체인 처리 | 이벤트 발행 → 수신 → 처리 완료 후 다음 이벤트 발행 (Event Chaining) | | **5** | 장애 대응 | DLQ(Dead Letter Queue) 구성, 실패 메시지 재처리 전략 수립 | | **6** | 모니터링 | Kafka + Prometheus + Grafana 구성으로 Topic, Lag, Consumer 상태 시각화 | | **7** | 보안 | Kafka 인증(SASL), 암호화(SSL), ACL 기반 권한 제어 설정 | | **8** | 성능 최적화 | Zero-copy 전송 설정, Batch 수신 사이즈 조정, 압축 전송(gzip/snappy) 적용 | | **9** | 데이터 정합성 | 이벤트 처리 상태 로깅, Transactional Messaging 설정 고려 | | **10** | 이벤트 표준화 | JSON Schema Registry 도입, Topic 네이밍 컨벤션 정의 (도메인.행위.상태) |   📌 시사점 요약   |  |  | | --- | --- | | 항목 | 주요 내용 | | **이벤트 중심 구조** | 서비스 간 느슨한 결합 구조로 확장성 확보 | | **고신뢰 메시징** | Kafka의 복제, 장애 복원, 로그 기반 메시지 보존으로 고가용성 보장 | | **분산 아키텍처 최적화** | 비동기 이벤트 흐름으로 시스템 병목 최소화 및 트랜잭션 분산 처리 | | **모니터링 체계** | 통합 대시보드 기반 실시간 모니터링 및 장애 대응력 강화 | |

#### ****PaaS 기반 배포 아키텍처 구조도****

|  |
| --- |
| [사용자 단말 (웹/모바일/BI포털)]  │  ▼  [NSX-T / 백본망 L4 Switch (로드 밸런서)]  │  ▼  [GoRouter (Ingress Controller)]  │  ▼  [Platform as a Service (Tanzu 기반 PaaS)]  ├─▶ [WEB 컨테이너 (xFrame UI)]  │ └─ Spring Boot + React/Vue  │ └─ TLS 인증, OAuth2 연계  │  ├─▶ [PT WAS 컨테이너 (xDataSet5)]  │ └─ Spring Boot 기반 API 게이트웨이  │ └─ JWT 기반 인증 및 토큰 처리  │  ├─▶ [BT WAS 컨테이너 (Business Logic)]  │ └─ FACADE - RULE - THING - DAO 구조  │ └─ API 서비스 별 MSA 구조  │  ├─▶ [ETL 컨테이너 (IBM DataStage)]  │ └─ 파일 수집/정제/적재 작업  │ └─ ControlM 연계, JDBC 연동  │  ├─▶ [통합 데이터 AP 컨테이너]  │ └─ 통합 API 라우팅, 캠페인 호출 처리  │ └─ 외부 연계 API (REST, JSON, OAuth2)  │  ├─▶ [BI 포털 AP 컨테이너 (MSTR)]  │ └─ Tableau, SAP BO 연계  │ └─ 메타조회, 리포트 제공  │  ├─▶ [OLAP 컨테이너 (MySQL + BI)]  │ └─ MSTR Web, Data Mart Query 가공  │  └─▶ [통합 데이터 포털 컨테이너]  └─ 사용자 인증 / 메타 조회 / 마케팅 연계  └─ BI Proxy / 분석 포털 UI |
| ✅ 구성 요소별 설명   |  |  |  | | --- | --- | --- | | 계층 | 컴포넌트 | 설명 | | **L4 Load Balancer** | NSX-T LB | 사용자 접근 분산 처리 및 인그레스 라우팅 진입점 | | **Ingress Controller** | GoRouter (CF Router) | 사용자 요청을 해당 컨테이너로 포워딩 | | **WEB 컨테이너** | xFrame UI, Vue, React | 사용자 UI 제공, 공통 CSS 및 API 호출 탑재 | | **PT WAS** | xDataSet5, Spring Boot | API Gateway 역할, API 인증, 토큰 관리 | | **BT WAS** | FACADE-RULE-THING-DAO | 비즈니스 로직, 트랜잭션, 연계 처리 | | **ETL 컨테이너** | DataStage, Oracle Client | 배치 적재 처리, 파기/로깅 이력 연계 | | **통합 데이터 AP** | API 관리 서버 | BI, OLAP, 마케팅, 외부API 연동 처리 | | **BI 포털** | MSTR, Tableau | 분석 리포트, 메타조회, BI 연계 | | **OLAP 분석기** | MSTR Web, MySQL | OLAP 분석 쿼리 처리, 대용량 분석 지원 | | **데이터 포털** | Spring Boot + Portal UI | 사용자 인증, 정책기반 연계 제어 |   ✅ 전략적 의미   |  |  | | --- | --- | | 요소 | 전략적 고려 사항 | | **컨테이너 분리** | 서비스별 스케일링 및 독립 장애 대응 가능 | | **PaaS 기반** | 배포 자동화, 환경 간 일관성 확보 | | **인증 보안 체계** | OAuth2 + JWT 기반 통합 인증 구성 | | **ETL/OLAP 분리** | 정형 분석과 비정형 전처리 분리로 성능 최적화 | | **통합 데이터 AP** | 외부 API 요청을 단일 게이트웨이로 통합 처리 | | **BI 포털/OLAP** | 사용자별 분석 환경 제공 및 보고서 공유 최적화 | |

#### ****CI/CD 연계 구성도 (PaaS 기반 자동화 배포 파이프라인)****

|  |
| --- |
| [개발자 Git Commit]  │  ▼  [Git Repository (GitHub / GitLab)]  │  ▼  [CI 서버 (Jenkins / GitLab CI)]  ├─ Build (Maven / Gradle)  ├─ Code Quality Check (SonarQube)  ├─ Unit Test (JUnit, Mockito)  └─ Image Packaging (Docker Build)  │  ▼  [Artifact Repository (Nexus / JFrog)]  │  ▼  [CD 서버 (Spinnaker / ArgoCD)]  ├─ Deployment Descriptor 확인  ├─ Helm Chart / Kustomize 실행  └─ Kubernetes 배포 (Tanzu PaaS)  │  ▼  [K8s Cluster (NSX-T 연계 PaaS)]  ├─ PT WAS / BT WAS / UI WAS  ├─ API Gateway / Kafka / ETL 컨테이너  └─ Logging/Monitoring 연동 |
| 📌 구성요소 설명   |  |  | | --- | --- | | 구성요소 | 역할 | | Git Repository | 소스 코드 저장소 (버전 관리 및 협업) | | CI 서버 | 코드 품질 검사, 유닛 테스트, 빌드, 도커 이미지 생성 | | Artifact Repository | 빌드 결과물(바이너리, 도커 이미지) 저장 | | CD 서버 | Manifest 확인 및 배포 자동화 (Helm/Kustomize) | | Kubernetes Cluster | 실제 서비스 배포 환경 (Tanzu 기반 컨테이너 플랫폼) | | PaaS API Gateway | 외부 서비스 연계 및 인증 정책 처리 | |

#### ****보안망 구성도 (L4 + WAF + ACL 기반 아키텍처)****

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [사용자 단말 (웹/모바일/BI)]  │  ▼  [L4 Load Balancer (NSX-T 기반)]  │  ▼  [Web Application Firewall (WAF)]  ├─ 정책 ① SQL Injection 차단  ├─ 정책 ② URL Whitelist 처리  └─ 정책 ③ JWT 기반 인증 전처리  │  ▼  [Ingress Controller (GoRouter, Istio Ingress)]  │  ▼  [ACL 정책 (NSX Security Group)]  ├─ 내부망 → DB 접근 제어  ├─ WAS ↔ DB 트래픽만 허용  └─ API 외부 노출 제한 (특정 포트만 허용)  │  ▼  [Tanzu PaaS 기반 PODs]  ├─ WEB WAS (xFrame5)  ├─ PT WAS (xDataSet5)  ├─ 통합 데이터 AP (Spring Boot)  ├─ ETL (DataStage)  ├─ BI 포털 / OLAP  📌 보안정책 요약   |  |  | | --- | --- | | 영역 | 보안 적용 사항 | | **L4 Load Balancer** | 트래픽 분산 및 SSL 종료 처리 | | **WAF** | XSS/SQL Injection 방지, 경로 접근 제한 | | **Ingress Controller** | TLS/SSL 적용, 도메인별 라우팅 | | **NSX ACL** | 시스템 간 통신 제어, 접근 경로 제한, DB 포트 보호 | | **POD 내부** | 컨테이너 간 통신 시 TLS 적용, RBAC 기반 인증 |   ✅ 전략적 시사점   |  |  | | --- | --- | | 항목 | 시사점 | | CI/CD 파이프라인 | 자동화 기반 빠른 배포, 품질 통제, 재현 가능한 인프라 운영 | | WAF + ACL + NSX-T | 금융권 보안 요구사항 충족, 민감 데이터 보호, 통제된 인프라 접근 | | PaaS 클러스터 보안 연계 | 컨테이너 중심 보안 + 네트워크 중심 보안의 하이브리드 구성 완성 | |

#### ****DevSecOps 체계 구성도****

|  |
| --- |
| [개발자 IDE]  └─> [코드 작성 및 Git Commit]  │  ▼  [Git Repository (GitHub/GitLab)]  └─> Webhook Trigger  │  ▼  [CI 서버 (Jenkins / GitLab CI)]  ├─ Static Code Analysis (SonarQube + PMD + Checkmarx)  ├─ Open Source Dependency Scan (OWASP Dependency-Check, Snyk)  ├─ Container Image Scan (Trivy, Clair)  ├─ Secret Scan (Gitleaks, GitGuardian)  └─ Build + Unit Test (JUnit, Mockito)  │  ▼  [Artifact Repository (Nexus / JFrog Artifactory)]  └─> Image Signing / Provenance 등록  │  ▼  [CD 서버 (ArgoCD / Spinnaker)]  ├─ Kustomize / Helm 배포 관리  ├─ Kubernetes Admission Controller 연계  └─ Policy-as-Code 검사 (OPA, Kyverno)  │  ▼  [Kubernetes Cluster (Tanzu / NSX-T)]  ├─ Runtime Protection (Falco, Sysdig)  ├─ Pod Security Policy / RBAC  ├─ Ingress 보안 (Istio / TLS 인증)  └─ Runtime Log & Trace (ELK, Prometheus, Grafana) |
| DevSecOps 단계별 구성 설명   |  |  |  | | --- | --- | --- | | 단계 | 주요 도구 | 설명 | | **코드 작성 & 커밋** | Git, GitLab, GitHub | 개발자가 소스 커밋 시 자동 파이프라인 연동 | | **정적 분석** | SonarQube, Checkmarx | 보안 취약점, 코드 품질, 컴플라이언스 점검 | | **오픈소스 분석** | Snyk, OWASP DC | 오픈소스 라이브러리의 취약점, 라이선스 분석 | | **시크릿 탐지** | GitGuardian, Gitleaks | API Key, Password 등의 민감 정보 탐지 | | **이미지 스캔** | Trivy, Clair | Docker 이미지 내 취약 패키지 점검 | | **서명 및 정책 등록** | Sigstore, Notary | 이미지 신뢰성을 위한 서명 및 무결성 보장 | | **정책 기반 배포** | OPA(Gatekeeper), Kyverno | 정책 미준수 시 배포 차단 (예: root 사용자 금지) | | **런타임 보안** | Falco, Sysdig | 실행 중인 컨테이너에 대한 보안 이벤트 실시간 탐지 | | **관측(Observability)** | ELK, Prometheus, Grafana | 보안 로그 분석, 성능 모니터링, SLA/SLO 추적 |   ✅ DevSecOps 전략적 적용 시사점   |  |  | | --- | --- | | 영역 | 전략적 의미 | | **Shift Left** | 개발 단계부터 보안 내재화로 리스크 조기 제거 | | **Shift Right** | 운영 중 실시간 보안 위협 탐지 및 대응 | | **자동화 기반 보안 체계** | 배포 속도 유지하면서 보안 강화 | | **규정 준수 자동 검증** | 금융보안원 지침, ISMS-P, 개인정보보호법 준수 체계 확립 | | **신뢰 기반 소프트웨어 공급망** | 이미지 서명 및 무결성 검증 통한 공급망 보안 강화 | |

## ****마무리말****

주요 설계 패턴 정의는 EA 기반의 시스템 아키텍처에서 신뢰성과 확장성을 동시에 확보하는 중요한 요소입니다.  
Microservice Architecture(MSA)는 독립적인 배포와 확장이 가능하여 시스템 확장에 유연성을 제공하며, Monolithic Architecture는 트랜잭션 관리와 일관성 유지를 극대화합니다.  
CQRS를 통한 읽기와 쓰기의 분리는 대규모 데이터 처리 시 성능을 극대화하며, Event-driven Architecture는 비동기 처리를 통해 실시간 데이터 처리와 복원력을 높입니다.  
마지막으로, Saga Pattern은 분산 환경에서 트랜잭션의 무결성을 보장하여 장애 복구를 신속하게 처리할 수 있습니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **Microservice Architecture** | 독립 배포와 확장이 용이하여 변화 대응이 빠르며, 장애 발생 시 서비스 격리가 가능하다. |
| **Monolithic Architecture** | 중앙 집중화된 트랜잭션 관리로 데이터 일관성과 복구 속도가 빠르며, 유지보수가 용이하다. |
| **CQRS** | 읽기와 쓰기를 분리하여 데이터 처리 속도를 최적화하며, 실시간 분석에 강력한 성능을 발휘한다. |
| **Event-driven Architecture** | 비동기 메시징 구조로 서비스 간 느슨한 결합을 유지하여 확장성이 뛰어나다. |
| **Saga Pattern** | 분산 트랜잭션의 일관성을 보장하며, 장애 발생 시 보상 트랜잭션을 통해 무결성을 유지한다. |

# ****보안 및 인증 설계****

## ****도입전안내말****

보안 및 인증 설계는 EA(Enterprise Architecture) 아키텍처 설계에서 가장 중요한 요소 중 하나입니다.  
특히 금융권과 같은 민감한 데이터 처리가 빈번한 시스템에서는 강력한 인증 체계와 데이터 보호가 필수적입니다.  
이번 항목에서는 인증(Authentication), 인가(Authorization), 전송 보안, API 보안의 네 가지 영역을 중심으로 EA 기반의 설계 방안을 정의합니다.  
이를 통해, 사용자의 신원 보장, 데이터 무결성, 네트워크 보안을 확보하여 안전한 서비스 환경을 구축합니다.

## ****보안 및 인증 설계****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **6.1 인증(Authentication)** | 6.1.1 OAuth2.0 | 외부 애플리케이션이 자원을 접근할 때 권한을 위임하는 인증 방식 | Spring Security, Keycloak, Okta | 인증을 위임하여 보안을 강화하고 사용자 경험을 개선 |
|  | 6.1.2 OpenID Connect | OAuth2.0를 확장한 사용자 인증 프로토콜, SSO(Single Sign-On) 지원 | Keycloak, Auth0, Okta | 단일 로그인(SSO)으로 사용자 편의성 증대 및 보안성 강화 |
|  | 6.1.3 Multi-Factor Authentication | 이중 인증, 생체 인증 등 복수의 인증 방법을 적용하여 보안 강도를 높임 | Google Authenticator, Authy, Duo Security | 단일 인증의 취약성을 보완하고 보안성을 극대화 |
| **6.2 인가(Authorization)** | 6.2.1 Role-Based Access Control (RBAC) | 사용자 역할에 기반한 접근 제어 정책 | Spring Security, Keycloak | 역할에 따른 접근 제어로 보안성을 확보하며, 관리가 용이함 |
|  | 6.2.2 Attribute-Based Access Control (ABAC) | 속성에 기반한 동적 접근 제어, 실시간 조건에 따른 접근 통제 | XACML, AWS IAM Policies | 정책 기반의 세밀한 권한 관리가 가능하여, 상황별 유연한 보안 정책 설정 가능 |
| **6.3 전송 보안 (Communication Security)** | 6.3.1 TLS/SSL | 데이터 전송 중 암호화를 통해 도청 및 변조 방지 | Let's Encrypt, AWS ACM, Nginx | 네트워크 구간의 데이터 보호를 강화하고, 사용자 정보 유출을 방지 |
|  | 6.3.2 데이터 암호화 | 민감한 데이터는 전송 중 뿐만 아니라 저장 시에도 암호화하여 보호 | AES-256, RSA, SHA-256 | 데이터 유출 시 복호화가 어려워 보안성을 극대화 |
|  | 6.3.3 Mutual TLS | 양방향 TLS를 통해 클라이언트와 서버 간 상호 인증 | Nginx, Envoy Proxy | 클라이언트와 서버의 상호 인증을 통해 중간자 공격을 방지 |
| **6.4 API 보안** | 6.4.1 JWT (JSON Web Token) | 인증 정보를 암호화하여 API 호출 시 토큰 기반의 인증을 수행 | Spring Security, JWT Library | Stateless 방식으로 빠르고 확장성 있는 인증 처리 가능 |
|  | 6.4.2 Rate Limiting | API 호출 횟수를 제한하여 무차별 대입 공격 방지 | Kong, Nginx, AWS API Gateway | 악성 요청에 대한 방어가 가능하여 서비스 안정성 확보 |
|  | 6.4.3 API Key 관리 | 특정 애플리케이션에만 접근 허용하는 키 발급 방식 | AWS API Gateway, Azure API Management | 서비스 간 권한 통제와 요청 추적이 용이함 |
|  | 6.4.4 HMAC (Hash-based Message Authentication Code) | 메시지의 무결성을 보장하기 위한 해시 기반 인증 방식 | SHA-256, SHA-512, HMAC Library | 메시지 위변조 방지와 통신 보안을 강화함 |

### ****(실제)**** 6.2 보안 및 인증 설계 (NH 농협지주 데이터 플랫폼 기준)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 핵심 기술 스택 | 전략적 의미 |
| **6.1 인증 (Authentication)** | 6.1.1 OAuth 2.0 인증 서버 연계 | 통합 데이터 포털, API, 마케팅 시스템 간 OAuth 기반 토큰 발급/검증 처리 | Spring Security, Keycloak, OAuth Server | OAuth 기반 인증 위임으로 인증 일원화 및 시스템 간 신뢰성 확보 |
|  | 6.1.2 OpenID Connect 기반 통합 로그인 | BI 포털 Proxy Web 및 포털 시스템에서 계열사/행원 사용자 통합 로그인 지원 | Keycloak, OIDC Provider | 사용자 인증 분산 방지, 접근 통제 일원화 |
|  | 6.1.3 MFA (Multi-Factor Authentication) | 내부 직원/관리자 등 중요 사용자에 대해 OTP, Device 인증 추가 적용 | Duo Security, Google Authenticator, Keycloak MFA Plugin | 사용자 계정 도용 방지 및 인증강화 요구사항 만족 |
| **6.2 인가 (Authorization)** | 6.2.1 RBAC (역할 기반 인가) | 포털/OLAP에서 사용자별 역할(R, RW 등)에 따른 데이터 조회 권한 설정 | Keycloak Role Mapping, Spring Security | 사용자의 권한 최소화 원칙(Least Privilege)을 적용하여 보안 리스크 최소화 |
|  | 6.2.2 ABAC (속성 기반 인가) | 마케팅 허브, ETL 등 조건 기반 접근 제어에 대해 속성기반 정책 구성 | XACML, ABAC Rule Engine | 상황(시간, IP, 위치)에 따른 동적 보안 정책 운영 가능 |
| **6.3 전송 보안 (Communication Security)** | 6.3.1 TLS 1.2+ 통신 표준화 | 모든 REST API 연계 구간에 TLS 1.2 이상 암호화 적용 | Let's Encrypt, TLS Cert Manager | 네트워크 중간자 공격(MITM) 방지 및 금융보안 표준 준수 |
|  | 6.3.2 mTLS (Mutual TLS) 내부 API 연계 적용 | 통합 데이터 AP ↔ 마케팅 시스템, ETL ↔ DB 연계에 mTLS 적용 | Istio, Envoy, Nginx | 시스템 간 위변조 방지 및 서비스 신뢰성 확보 |
|  | 6.3.3 민감 데이터 전송 암호화 | 주민번호, 고객번호 등 민감정보를 JSON 내 암호화 처리 | AES-256, RSA, JSON Encryption Lib | 고객 데이터 유출 시에도 암호화로 보호 효과 극대화 |
| **6.4 API 보안** | 6.4.1 JWT 기반 API 인증 및 무상태화 | 통합 데이터 AP에서 JWT 기반 인증 처리 및 API 호출 무상태화 구성 | JWT Token Lib, Spring Security | API 호출 효율성 향상 및 분산 처리 구조 최적화 |
|  | 6.4.2 API Key + Rate Limiting 적용 | 외부 연계(UMS, 올원뱅크 등)에 API Key + 호출 빈도 제한 설정 | AWS API Gateway, Kong, Nginx | 비인가 호출 차단 및 API 남용 방지 |
|  | 6.4.3 서명 기반 요청 인증 (HMAC) | 통합고객번호 기반 민감 API 호출 시, 요청 메시지 서명 검증 구조 적용 | HMAC SHA256, API Gateway | API 위·변조 탐지, 데이터 무결성 확보 |
| **6.5 내부 보안 체계 강화** | 6.5.1 로그 모니터링 + 감사 로깅 체계 | 인증/인가/데이터조회 로그를 통합 저장 및 실시간 모니터링 | ELK, Loki, Fluentd, Audit Log Server | 이상 징후 조기 감지 및 보안 사고 사후 분석 근거 확보 |
|  | 6.5.2 SSO 기반 보안 정책 일원화 | 포털 ↔ API ↔ ETL 사용자 인증을 SSO로 통합 | SAML 2.0, OIDC, Keycloak | 사용자 인증·인가 연동 편의성 확보, 정책 통제 일원화 |
|  | 6.5.3 인증서 기반 보안 통신 확대 | ETL ↔ DB, AP ↔ 내부 시스템 간 인증서 기반 통신 연계 | TLS + Client Cert 기반 구성 | 금융기관 보안 지침(ISMS-P 등) 준수 |
| **6.6 데이터 보안 및 보호** | 6.6.1 DB 암호화 및 접근 제어 | Oracle HEX DB + DB-Safer 기반 테이블 암호화 및 접근 권한 관리 | DB-Safer, Oracle TDE | 개인정보 및 분석 데이터 보호, 관리자 접근 로깅 기반 통제 강화 |
|  | 6.6.2 로그 보관 및 감사 정책 연계 | 모든 API 호출, DB 조회 내역을 주체별로 로그화 및 장기 보관 | WORM 디스크, Log Archive System | 금융사 로그보관 정책 및 ISMS 인증 대응 |

#### 보완된 보안 설계 시사점

|  |  |
| --- | --- |
| 구분 | 시사점 |
| 인증/인가 일원화 | Keycloak 기반으로 모든 시스템의 인증/인가를 통합하여 사용자 경험과 정책 통제를 동시에 강화 |
| API 보안 강화 | JWT + HMAC + RateLimit을 조합하여 외부 API 연계의 보안성과 안정성 동시 확보 |
| 데이터 보안 | DB 암호화, TLS/mTLS, 로그 감사 체계를 병행 적용하여 **금융보안원 기준 충족** |
| DevSecOps 연계 | 보안 정책을 CI/CD 파이프라인에 내재화하여 개발→배포→운영 전 단계에서 보안 자동화 적용 가능 |

#### ****사용자 인증 흐름도 및 인증 토큰 라이프 사이클****

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **🔹 [사용자 인증 흐름도 (OAuth2 + OpenID + JWT)]**  [사용자]  │  │ 1. 로그인 요청 (ID/PW 입력)  ▼  [통합 인증 서버 (Keycloak, OAuth2.0)]  │  │ 2. 사용자 인증 → 인증 성공 시 Authorization Code 발급  ▼  [프론트엔드 채널 (포털/BI Proxy)]  │  │ 3. Authorization Code로 Access Token + Refresh Token 요청  ▼  [통합 인증 서버]  │  │ 4. 토큰 발급 (Access + Refresh Token) → 전달  ▼  [사용자 단말]  │  │ 5. Access Token 포함하여 API 요청  ▼  [API Gateway (Kong/Zuul)]  │  │ 6. Access Token 유효성 검증 (Keycloak 연동)  │  │ → 유효 시 내부 시스템으로 라우팅  ▼  [내부 서비스/API (ETL, DB, Portal)]  **🔹 [JWT 인증 토큰 라이프사이클]**   |  |  | | --- | --- | | 구분 | 설명 | | **발급 시점** | 사용자 인증 완료 후 Keycloak이 JWT Token 발급 (Access + Refresh) | | **Access Token 유효 시간** | 기본 15분~1시간 이내 (사용자 인증 상태 유지 시 최소화 가능) | | **Refresh Token 유효 시간** | 일반적으로 7일 이상 유지되어 재인증 없이 토큰 갱신 가능 | | **갱신 프로세스** | Access Token 만료 시 Refresh Token으로 재발급 요청 수행 | | **파기 조건** | 로그아웃 시, 명시적 만료, 비인가 접근 탐지 시 서버 측 토큰 폐기 수행 | |  |  | |
| **✅ 2. OAuth2.0 + JWT 기반 보안 처리 구조도**  [사용자]  │  │ 1. OAuth2.0 로그인 요청  ▼  [Keycloak 인증 서버]  │  │ 2. 사용자 인증 (LDAP/DB/SSO)  ▼  ├─ Access Token 발급 (JWT)  └─ Refresh Token 발급 (JWT)  ▼ ▲  [API Gateway (Kong/Zuul)] ───── 3. API 호출 시 JWT 포함  │ │  │ └─ 4. JWT 검증 (Public Key, Signature 확인)  ▼  [내부 API 서버 / Microservice (Spring Boot, OLAP API 등)]  │  └─ 5. 사용자 권한별 리소스 접근 처리 (RBAC/ABAC 기반 인가)  (※ 토큰 만료 시 Refresh Token으로 자동 재발급 처리) |
| **3. API Gateway 수준 보안 정책 적용 템플릿**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 구분 | 보안 정책 항목 | 설명 | 적용 기술 스택 | 전략적 의미 | | 1 | 인증 토큰 처리 | JWT 기반 OAuth2 토큰 필수 전달 및 검증 | Kong Plugin, Zuul Filter | 사용자 식별 및 무단 접근 차단 | | 2 | Rate Limiting | 1분당 호출 횟수 제한 (ex. 100 req/min) | Kong Rate-Limit, Nginx Limit | API Flooding, DoS 방지 | | 3 | IP 화이트리스트 | 등록된 IP만 접근 허용 (행내망 등) | Kong ACL, Nginx IP Restriction | 외부 공격 접속 차단 | | 4 | API Key 관리 | 외부 연계시 발급된 Key만 접근 허용 | Kong Key Auth, AWS API Gateway | 파트너/서비스 간 권한 제어 | | 5 | OAuth2 연계 | Authorization Server 연동 | Keycloak + Kong OAuth2 Plugin | 사용자 인증 연계 일원화 | | 6 | 인증서 기반 mTLS | API 요청 시 클라이언트 인증서 요구 | Envoy, Istio mTLS | API 호출 주체 인증 신뢰성 확보 | | 7 | 메시지 서명(HMAC) | 민감정보 포함 시 HMAC SHA256 기반 서명 요청 | SHA256, API HMAC Plugin | API 위·변조 방지 | | 8 | 요청/응답 로깅 | API 요청 전체 로그 기록 및 감사 | Fluentd, Elastic Stack | 보안사고 분석 및 대응 기반 마련 | | 9 | Swagger 보안 연동 | OpenAPI 명세에 보안 필드 추가 | Swagger + OAuth2 Scopes | 보안정책 기반 API 문서화 | | 10 | CORS 정책 제어 | 허용된 Origin에서만 API 호출 허용 | Kong CORS Plugin, Nginx | 브라우저 기반 XSS/CSRF 방지 | |
|  |

#### ****API Gateway 구성도 + 인증 흐름도 통합 시각화****

|  |
| --- |
| [사용자 (계열사, 분석가 등)]  │  │ (1) 로그인 요청 (ID/PW 또는 SSO)  ▼  [인증 서버 (Keycloak)]  │  │ (2) 사용자 인증 성공  │ ├─ Access Token (JWT) 발급  │ └─ Refresh Token 발급  ▼  [사용자 브라우저 또는 앱]  │  │ (3) API 요청 시 Access Token 포함  ▼  [API Gateway (Kong, Zuul, Nginx)]  │  │ (4) JWT 토큰 유효성 검증  │ ├─ Public Key 기반 서명 검증  │ └─ 만료 여부 확인  ▼  ├─────────────▶ [API 인증 실패 시 오류 반환]  │  │ (5) 인가 정책 확인 (RBAC / ABAC)  │  ▼  [내부 API 서비스 호출 라우팅]  ├─▶ 통합 데이터 AP (Spring Boot)  ├─▶ 통합 데이터 포털 (BI Proxy Web 포함)  ├─▶ 마케팅 시스템 API  └─▶ OLAP 분석 API  <내부 API 서비스 흐름 예시>  ▼  [통합 데이터 AP]  ├─ JWT 토큰 재검증  ├─ 사용자 권한 기반 서비스 제공  └─ JWT 클레임 기반 데이터 요청 실행  ▼  [통합 데이터 DB] 또는 [마케팅 HUB]  <토큰 만료 시>  ▼  [사용자 → Keycloak] (Refresh Token 요청)  └─ 새로운 Access Token 재발급 후 재시도 |
| 📌 구성 요소별 설명   |  |  |  | | --- | --- | --- | | 구성 요소 | 역할 | 비고 | | **사용자** | 계열사 사용자, 분석가, 관리자 | Web/앱 모두 가능 | | **Keycloak (인증 서버)** | OAuth2, OpenID Connect, JWT 발급 | SSO, MFA 지원 | | **API Gateway** | JWT 인증, API 라우팅, Rate-Limiting, CORS 처리 | Kong 또는 Zuul/Nginx 기반 | | **내부 서비스** | 비즈니스 API, ETL 처리, 마케팅 응답, OLAP 질의 처리 | Spring Boot 기반 Microservice | | **DB / Kafka / OLAP** | API 백엔드 연계 대상 | JDBC / Kafka 기반 실시간 연계 |   📌 전략적 시사점   * **통합 인증 흐름으로 사용자 경험(UX) 향상** → 인증 재요청 없이 Access/Refresh Token 기반 접근 유지 * **API Gateway 수준에서 보안 정책 중앙화** → 무분별한 내부 접근 차단 + 로깅 + 접근제어 가능 * **JWT 기반 무상태 인증 구조로 확장성 확보** → 클라우드 환경과 잘 맞는 Stateless API 구조 * **실시간/배치 연계 흐름에도 동일한 인증정책 적용** → 통합된 인증정책으로 일관성 있는 보안 체계 유지 |

## ****마무리말****

보안 및 인증 설계는 시스템의 안정성을 보장하고, 금융권과 같은 민감한 데이터 처리를 보호하기 위해 반드시 강화되어야 합니다.  
OAuth2.0 및 OpenID Connect를 활용한 인증 체계는 SSO(Single Sign-On) 환경을 제공하며, 다단계 인증을 통해 신뢰성을 극대화합니다.  
RBAC와 ABAC는 역할 및 속성에 기반한 세밀한 접근 제어를 제공하며, API 보안은 JWT와 Rate Limiting을 통해 무단 접근을 차단합니다.  
또한, TLS/SSL을 통해 전송 구간의 보안을 확보하며, Mutual TLS로 양방향 인증을 강화합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **인증(Authentication)** | OAuth2.0 및 OpenID Connect를 통한 인증 위임으로 확장성과 보안성을 극대화한다. |
| **인가(Authorization)** | RBAC와 ABAC를 통한 세밀한 접근 제어는 금융 시스템의 보안 강화를 위한 필수 요소이다. |
| **전송 보안 (TLS/SSL)** | 데이터 전송 구간 암호화는 정보 유출을 방지하고 중간자 공격(MITM)을 차단한다. |
| **API 보안** | JWT 및 Rate Limiting 적용을 통해 API 접근을 안전하게 관리하며 악성 요청을 방어할 수 있다. |
| **Mutual TLS** | 클라이언트와 서버의 상호 인증을 통해 신뢰할 수 있는 통신 채널을 보장한다. |

# ****로깅 및 모니터링 체계 구축****

## ****도입전안내말****

로깅 및 모니터링 체계는 어플리케이션의 실시간 상태를 추적하고, 오류 발생 시 신속한 대응을 가능하게 하는 중요한 관리 요소입니다.  
EA(Enterprise Architecture) 관점에서 체계적인 로깅과 모니터링 설계는 장애 탐지 및 분석, 성능 최적화, 보안 대응을 강화합니다.  
특히 금융권 시스템에서의 실시간 트랜잭션 모니터링과 감사 추적(Audit Trail)은 필수적이며, 중앙 집중형 관리가 필요합니다.

## ****로깅 및 모니터링 체계 구축****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **7.1 통합 로그 관리** | 7.1.1 ELK Stack (Elasticsearch, Logstash, Kibana) | 실시간 로그 수집, 분석 및 시각화가 가능한 오픈 소스 기반의 통합 로그 관리 플랫폼 | Elasticsearch, Logstash, Kibana | 장애 탐지 및 분석 시간 단축, 실시간 검색 가능, 보안 감사 로그 추적 용이 |
|  | 7.1.2 Fluentd | 다양한 소스의 로그를 통합 수집하여 중앙으로 전송하고, 분석 가능하도록 정제 | Fluentd, Fluent Bit | 멀티 클라우드 환경에서도 유연한 로그 수집이 가능함 |
|  | 7.1.3 Graylog | 실시간 로그 수집 및 모니터링, 검색이 가능한 로그 관리 도구 | Graylog Server, MongoDB, Elasticsearch | 대용량 로그 처리 시 빠른 검색과 실시간 알림 기능 제공 |
|  | 7.1.4 Splunk | 엔터프라이즈급 로그 분석 솔루션으로 대규모 데이터 분석과 보안 이벤트 탐지 가능 | Splunk Enterprise, Splunk Cloud | 실시간 보안 분석 및 장애 대응에 최적화됨 |
| **7.2 모니터링 도구** | 7.2.1 Prometheus | 시계열(Time-series) 데이터 수집 및 모니터링을 위한 오픈 소스 플랫폼 | Prometheus Server, Alertmanager, Grafana | 메트릭 기반의 모니터링으로 시스템 상태를 실시간 추적하고 장애 시 빠르게 대응 가능 |
|  | 7.2.2 Grafana | 시각화 도구로 Prometheus, InfluxDB 등에서 수집한 메트릭을 시각적으로 표현 | Grafana Dashboard, Prometheus, InfluxDB | 실시간 모니터링 대시보드를 제공하여 장애 탐지 및 원인 분석 시간 단축 |
|  | 7.2.3 Zabbix | 네트워크, 서버, DB, 애플리케이션의 성능 모니터링 및 장애 탐지 | Zabbix Server, Agent, Proxy | 에이전트 기반의 세부 모니터링이 가능하며, 경고 및 알림 설정을 통한 실시간 대응 가능 |
|  | 7.2.4 Dynatrace | AI 기반 애플리케이션 성능 모니터링(APM) 및 자동 문제 탐지 | Dynatrace Platform, OneAgent | 트랜잭션 분석 및 자동화된 장애 탐지를 통해 복구 시간을 단축 |
| **7.3 분석 및 알림 체계** | 7.3.1 실시간 알림 | 임계치 초과 또는 장애 발생 시 실시간 알림을 통해 빠르게 대응 | Slack, PagerDuty, Opsgenie | 장애 발생 시 빠른 복구를 통해 서비스 연속성을 보장 |
|  | 7.3.2 Root Cause Analysis (RCA) | 장애 원인 분석을 통해 동일한 문제 발생을 예방하기 위한 체계 구축 | ELK Stack, Dynatrace, Splunk | 근본 원인 분석을 통해 반복적인 장애를 최소화하고, 대응 전략을 개선 |
|  | 7.3.3 이력 관리(Audit Trail) | 로그와 모니터링 정보를 장기 보관하여 보안 감사 및 장애 분석 시 추적 가능 | ELK Stack, Splunk, Graylog | 금융권 규제 준수 및 감사 보고서 작성 시 활용 가능 |

### ****(실제)**** 로깅 및 모니터링 체계 구축 설계 (NH 농협지주 데이터 플랫폼 기준)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 핵심 기술 스택 | 전략적 의미 |
| 7.1 통합 로그 관리 | 7.1.1 ELK Stack 기반 로그 수집 | Spring Boot, Kafka, API Gateway, DB 등 전 구성요소 로그를 수집하여 중앙 저장 | Elasticsearch, Logstash, Kibana | 실시간 장애 탐지 및 보안 감사 대응 체계 확보 |
|  | 7.1.2 Kafka 기반 로그 버퍼링 | 수집량 급증 시 Kafka로 로그를 중간 버퍼링하여 Logstash의 부하 분산 | Kafka, Kafka Connect | 대량 이벤트 로그 유입 시에도 안정적인 로그 흐름 유지 |
|  | 7.1.3 Audit Trail 로그 체계 | 사용자 요청, 데이터 변경 이력, 관리자 액션 등을 별도 감사 로그로 분리 저장 | Fluentd, Graylog | 금융감독원 등 외부 감사 대응에 필요한 추적성 확보 |
|  | 7.1.4 멀티계층 로그 분리 | 채널, API, DB, ETL, 외부연계 등 레이어별 로그 분리 설계 | Filebeat, Logback, MDC | 서비스 장애 범위 및 영향을 빠르게 구분할 수 있음 |
|  | 7.1.5 로그 암호화 및 마스킹 | 개인정보, 거래정보 등 주요 로그는 암호화/마스킹 후 저장 | Logstash Filter, AES-256 | 개인정보보호법, ISMS-P 준수 확보 |
| 7.2 모니터링 체계 | 7.2.1 Prometheus 기반 메트릭 수집 | PaaS POD, WAS, DB, Kafka 등 리소스/트래픽/지표를 시계열로 수집 | Prometheus, Node Exporter, Grafana | 실시간 트렌드 기반 성능 모니터링 및 예측적 대응 가능 |
|  | 7.2.2 Grafana 대시보드 | 메트릭 기반 실시간 대시보드 구성 (서비스 응답, API 호출 수, CPU/RAM 등) | Grafana, Prometheus | 장애 조기 감지 및 운영가시성 확보 |
|  | 7.2.3 E2E 트랜잭션 추적 | 요청-ID 기반 Trace를 통해 마이크로서비스간 전체 트랜잭션 흐름 추적 | Spring Sleuth, Zipkin, Jaeger | 분산 트랜잭션의 병목 구간과 실패 원인을 실시간으로 분석 가능 |
|  | 7.2.4 실시간 알림 및 자동 대응 | 메트릭 임계치 초과 시 Slack, SMS, PagerDuty 연계 자동 알람 발송 | Alertmanager, Opsgenie | 시스템 다운타임 최소화 및 24x7 대응 체계 구현 |
| 7.3 분석 및 보안 연계 | 7.3.1 보안 로그 통합 | API 접근, 인증 실패, 관리자 접근 등의 보안 로그 통합 수집 및 이상 탐지 | Suricata, Wazuh, ELK | 보안 이벤트 추적 및 이상 행동 감지 체계 강화 |
|  | 7.3.2 SLA/SLO 모니터링 | API 응답속도, 가용률, 오류율 등 서비스 품질 지표 관리 및 시각화 | Prometheus, Grafana, SLI Exporter | 서비스 안정성 관리 및 KPI 모니터링 대응 |
|  | 7.3.3 자동화된 RCA 체계 | 장애 발생 시 로그/모니터링/트레이스 기반 자동 원인 분석 리포트 생성 | Dynatrace, ELK, AIOps 기반 분석 | 반복 이슈 차단 및 운영 전략 자동화 기반 확보 |
|  | 7.3.4 로그 장기보관 및 이관 설계 | 로그 저장소의 볼륨 제한 해소를 위한 장기 보관 (S3, NAS 이관 포함) | Elastic Snapshot, Fluentd-to-S3 | 과거 장애 분석, 규제기관 제출용 데이터 확보 가능 |

### ****로깅 및 모니터링 체계 구축 관련 서버 구성****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 서버 명 | 주요 역할 | 탑재 컴포넌트 | 설치 위치 / 배포 구조 |
| 1 | **Logging Agent Node** | 각 App/WAS/ETL 등에서 로그 수집 | Filebeat, Fluentd, Logback | 각 POD 또는 노드에 DaemonSet 형식으로 설치 (Tanzu 기반 PaaS 클러스터 내부) |
| 2 | **Log Collector Server** | 로그 집계 및 필터링 처리 | Logstash, Fluentd | 전용 클러스터 노드 또는 ETL 서버 내부 구성 |
| 3 | **Log Storage Server** | 로그 저장소 및 검색 인덱싱 | Elasticsearch | 통합 데이터 DB 백엔드와 별도 구성된 Elasticsearch Cluster |
| 4 | **Log Visualization Server** | 시각화 및 Dashboard 제공 | Kibana, Grafana | BI 포털 또는 모니터링 포털 내부 서버 |
| 5 | **Trace & APM 서버** | 트랜잭션 추적 및 성능 분석 | Spring Sleuth, Zipkin, Jaeger | Spring Boot 기반 통합 데이터 AP 또는 별도 APM Server |
| 6 | **Metric Collector Server** | 메트릭 수집 및 알림 트리거 | Prometheus, Node Exporter | API 서버 및 ETL 서버에 Exporter 설치 후 통합 수집 |
| 7 | **Alert Notification Server** | 장애 발생 시 알림 발송 및 정책 실행 | Alertmanager, Opsgenie Relay | 모니터링 클러스터 내부 또는 PaaS Gateway 상단 |
| 8 | **Security Log Server** | 보안 이벤트 수집 및 분석 | Wazuh, Suricata, OSSEC | WAS 서버, API Gateway 서버에 Agent 설치 후 보안 서버와 연동 |
| 9 | **Audit Storage Node** | 장기 감사 로그 보관 및 백업 | Fluentd → S3/NAS, Archive FS | NAS 혹은 클라우드 Object Storage 연계 |
| 10 | **Monitoring Control Plane** | 통합 모니터링 환경 제어 및 구성 정책 관리 | Grafana, Zabbix, Dynatrace OneAgent | 운영지원 인프라 영역에 별도 설치 (On-Prem + Cloud Hybrid) |
| 🧩 연계 구조 개요  [App/WAS/ETL 노드]  └─ Filebeat / Fluentd ─▶ Logstash / Fluentd Aggregator  └─▶ Elasticsearch (Log Storage)  └─▶ Kibana (Dashboard 시각화)  [Prometheus Node Exporter] ─▶ Prometheus (Metric Collector)  └─▶ Grafana (모니터링 대시보드)  [Spring Boot APM (Sleuth, Zipkin)] ─▶ Jaeger / Zipkin UI  [Security Agent (Wazuh)] ─▶ Security Log Server  └─▶ ELK Stack + SIEM 연동 | | | | |
| 전략적 의미 요약   * **PaaS 중심 DaemonSet 설치 방식**: 로그/모니터링 Agent는 POD 또는 Node 단위로 배치하여 탄력적 확장 대응 * **분리된 저장/분석/시각화 구조**: 고가용성과 성능 보장을 위해 Elasticsearch와 시각화 서버를 분리 배치 * **APM 및 Trace 서버의 별도 구성**: 트랜잭션 병목 추적을 위한 고가시성 Trace 인프라 구축 * **보안 로그 분리 수집**: 내부 감사 및 외부 규제 대응을 위한 전용 보안 로그 서버 구성 | | | | |

### ****신규 도입 서버 대상 로깅 및 모니터링 아키텍처 및 기술스택****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 서버 명 | 도입 목적 | 필요 로깅/모니터링 아키텍처 | 적용 기술 스택 | 설계 전략 |
| 통합 데이터 AP | 실시간 API 처리, 비즈니스 로직 실행 | APM + 로그 트레이싱 + 실시간 오류 감지 | Spring Sleuth, Zipkin, Prometheus, Grafana, Fluentd | - Sleuth 기반 Trace ID 전파  - REST API 요청/응답 로그 저장  - 메트릭 수집 후 Grafana 시각화  - 오류 알림 연계 (Alertmanager) |
| 통합 데이터 포털 | 사용자 UI 서비스, BI 연계 포털 | UI 접근 이력 추적 + 사용자 행동 로깅 | Fluent Bit, Kibana, Audit Logger, Grafana | - 사용자별 API 호출 추적  - 접속 이력 및 행위 감사 로그 분리 보관  - 포털 성능 시각화 대시보드 |
| 통합 데이터 ETL | 대용량 배치 처리, 전처리 로직 실행 | 배치 Job 모니터링 + 이력 관리 + 성능 추적 | Control-M, Logstash, Prometheus, ELK Stack | - 배치 수행시간, Job 상태 수집  - 각 단계별 처리 로그 저장  - ETL 실패 시 알림 자동 전송 |
| BI 포털 (AP/DB) | 리포트 제공, 분석 결과 조회 | 쿼리 성능 로그 + 데이터 접근 이력 관리 | Zabbix, MySQL Slow Query Log, Grafana, Wazuh | - DB 접근 이력 모니터링  - 쿼리 실행시간/IO 패턴 분석  - BI 리포트 요청 이력 추적 |
| OLAP 서버 | 멀티 소스 분석 + 결과 통합 | 분석 쿼리 트레이싱 + 결과 이력 저장 | ClickHouse Log, Audit Trail, Grafana, Prometheus | - OLAP Query 수행시간 분석  - 필터/조건별 성능 분석  - 대용량 처리 트렌드 모니터링 |
| 마케팅 시스템 | 개인화 마케팅 처리 | 실시간 이벤트 추적 + 추천 결과 로깅 | Kafka Logging, Fluentd, Splunk, Zipkin | - Kafka 이벤트 흐름 실시간 추적  - 추천 모델 결과별 전처리 로그 저장  - 고객행동 예측 결과 저장 |
| E-마케팅 / 마케팅 HUB | 푸시/배너 연계 및 메시지 중계 | API 호출 모니터링 + 메시지 전송 상태 분석 | Prometheus, Fluentd, Elastic APM, Grafana, Alertmanager | - API 응답시간 모니터링  - 메시지 실패/재전송 로깅  - 대시보드 기반 KPI 추적 |
| 통합 아키텍처 도식 (요약)  [각 서버]  └─ Fluentd/Filebeat (로그 수집)  └─ Logstash → Elasticsearch  └─ Prometheus Exporter (메트릭 수집)  └─ Prometheus → Grafana (시각화)  [Spring Boot 기반 서버]  └─ Sleuth + Zipkin (Trace ID 추적)  [배치 서버 (ETL)]  └─ Control-M Logs → Logstash + Alertmanager  [보안/접근]  └─ Wazuh Agent (보안로그) → Wazuh Server → SIEM | | | | |
|  | | | | |

✅ 기술적 특징 및 보완 고려사항

|  |  |
| --- | --- |
| 구분 | 내용 |
| **보안** | 감사 로그와 운영 로그는 분리 보관하고, WORM(Write Once Read Many) 디스크 또는 NAS와 연계 |
| **지속성** | 로그 저장소는 S3 또는 NFS 백업 자동화 적용 (3개월/6개월/1년 기준 롤링) |
| **장애 대응** | Alertmanager + Opsgenie 기반 장애 알림 구성 (Slack/메일/전화연계) |
| **개발 연계** | 각 서버별로 DevSecOps CI/CD 파이프라인과 연동하여 테스트 로그도 수집 대상 포함 |

### ****Logging Mornitoring Diagram****

|  |
| --- |
| ### 🔍 도식 기반 클러스터별 로깅/모니터링 컴포넌트 배치도  ```  [NH 농협지주 데이터 플랫폼 클러스터]  |  +-------------------+-------------------+  | |  [Application Cluster] [ETL/Batch Cluster]  | |  +-----------------+-----------------+ +-----------+-----------+  | | | | |  [통합 데이터 AP] [통합 데이터 포털] [BI 포털] [ETL 서버] [OLAP]  | | | | |  | | | | |  | Fluent Bit/Fluentd Agent | | Fluentd + Control-M Logs  | +------------------+ | | +------------+  | | | | | |  | [Application Log] [Trace ID]| | [Batch Logs + Job 상태]  | | | | | |  | +--------+---------+ | | +-----------+  | | | | |  | [Logstash] (필터링/정제) | |  | | | | |  +------------------+----------------+---------+---------------------+  |  [Elasticsearch]  |  [Kibana Dashboard]  |  <로그 검색, 시각화, 감사 분석>  +-------------------+------------------+  | |  [Prometheus Server] [Wazuh Server - 보안 로그]  | |  +------------+------------+ +------+------+  | | | |  [Spring Boot Exporter] [Node Exporter] [Wazuh Agent] [Audit Trail DB]  | | |  Grafana (대시보드) 실시간 경고 <SIEM 연동, Alerting>  ```  ---  ### 🧩 설명 요약  - \*\*Application Cluster\*\*: 통합 데이터 AP, 포털, BI 포털 등의 Spring 기반 서버는 `Fluentd`, `Sleuth`, `Logstash`를 통해 \*\*Trace ID 기반 로깅\*\*, \*\*API 호출 추적\*\*, \*\*메트릭 수집\*\*을 처리.  - \*\*ETL/Batch Cluster\*\*: `DataStage`, `Control-M` 배치 로깅을 `Fluentd`로 수집하고, 처리 상태/시간/결과를 정제 후 Logstash로 전달.  - \*\*Elasticsearch + Kibana\*\*: 로그 저장, 검색, 시각화.  - \*\*Prometheus + Grafana\*\*: 메트릭 수집 및 대시보드 시각화.  - \*\*Wazuh\*\*: 보안 로그 모니터링 및 SIEM 대응 체계 구성.  ---  필요 시 이 흐름을 기반으로 \*\*Grafana 대시보드 레이아웃 설계\*\*, \*\*Log Schema 설계\*\*, 또는 \*\*Alerting 정책 구성 예시\*\*까지 이어서 작성 가능합니다. 어떤 항목을 다음으로 도와드릴까요? |
|  |

### ****Fluentd 기반 로그 수집 및 Prometheus 기반 메트릭 수집 아키텍처****

## ✅ Fluentd / Prometheus 구성 및 로그 포맷 정책

### Fluentd 기반 로그 수집 구조

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 역할 | 비고 |
| **Fluentd Agent** | 애플리케이션(Pod) 내부에서 로그 수집 | Sidecar 또는 DaemonSet 방식 배포 |
| **Fluent Bit (전처리)** | 로그 필터링 및 구조화, 성능 최적화 | 고속 처리용 전처리기 |
| **Fluentd Aggregator** | 로그 수집 집중 노드, 태그 기반 라우팅 | 보안정책에 따라 TLS 통신 적용 |
| **Log Storage** | Elasticsearch, OpenSearch 등 | Kibana / Grafana와 연동 |
| **시각화 도구** | Kibana, Grafana | 실시간 대시보드 구성 |

### Prometheus 기반 메트릭 수집 구조

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 역할 | 비고 |
| **Prometheus Server** | 시계열 메트릭 수집/저장 | Pull 방식, HTTP Exporter 연동 |
| **Node Exporter** | 서버 수준 CPU, Memory, Disk 정보 수집 | 모든 노드에 설치 필요 |
| **cAdvisor** | 컨테이너 리소스 모니터링 | K8s 환경에서 필수 |
| **Alertmanager** | 알람 룰 기반 알림 전송 | Slack, Opsgenie, Email 연동 가능 |
| **Grafana** | Prometheus 데이터 시각화 | 대시보드 템플릿 적용 가능 |

### 로그 포맷 통일화 정책

|  |  |  |
| --- | --- | --- |
| 항목 | 정책 내용 | 비고 |
| **공통 포맷** | JSON 기반 Key-Value 구조 사용 | 정형화된 구조로 파싱 효율성 확보 |
| **타임스탬프** | ISO 8601 (2025-05-14T11:00:00Z) | 타임존 포함 필수 |
| **Log Level** | DEBUG, INFO, WARN, ERROR, FATAL | 공통 레벨 사전 정의 |
| **Trace ID** | UUID 기반 traceId 필드 | 분산 트랜잭션 추적용 |
| **서비스 명** | serviceName, host, podName 필드 명시 | 마이크로서비스 식별용 |
| **사용자 정보** | userId, sessionId 등 포함 | 감사 추적 목적 |
| **비정상 로그** | Exception Stack Trace, 메시지 내용 캡처 | 전체 스택 포함 가능 |

### 로그 구조 설계 예시 (JSON)

{

"timestamp": "2025-05-14T11:00:00Z",

"logLevel": "INFO",

"serviceName": "data-portal-api",

"traceId": "d123-abcd-4567-efgh",

"userId": "nh\_user\_001",

"action": "GET /customer/info",

"status": 200,

"latencyMs": 123,

"host": "pod-01",

"message": "Customer info retrieved successfully"

}

### 전략적 의미 및 효과

|  |  |
| --- | --- |
| 항목 | 전략적 효과 |
| 로그 포맷 표준화 | 시스템 간 통합 로그 분석 효율성 증가, 장애 추적 용이 |
| Fluentd 구성 | 다양한 로그 소스와 유연한 수집 연계 |
| Prometheus 연계 | 리소스 기반 실시간 메트릭 추적 및 경보 설정 가능 |
| Trace ID 연계 | MSA 및 분산 시스템에서 호출 관계 분석 가능 |
| 통합 시각화 | Kibana / Grafana로 직관적 모니터링 구현 |

## ****마무리말****

로깅 및 모니터링 체계는 시스템의 안정성과 보안성을 유지하기 위한 핵심 관리 도구입니다.  
ELK Stack과 Fluentd를 통한 실시간 로그 수집과 통합 분석은 장애 탐지 시간을 단축시키며, Prometheus와 Grafana를 활용한 모니터링은 시스템의 상태를 시각적으로 추적하여 문제 발생 시 빠른 대응이 가능합니다.  
특히 금융권 시스템에서는 감사 추적(Audit Trail)이 필수적이며, 중앙 집중형 로그 관리와 실시간 분석을 통해 외부 공격 및 내부 이상 징후를 즉시 탐지할 수 있습니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **통합 로그 관리** | 중앙 집중형 로그 수집을 통해 빠른 장애 탐지와 대응이 가능하며, 감사 로그로 보안 이슈 추적이 용이하다. |
| **모니터링 도구** | 실시간 모니터링을 통해 성능 최적화와 장애 대응 시간을 단축할 수 있다. |
| **분석 및 알림 체계** | Root Cause Analysis와 실시간 알림을 통해 장애의 반복 발생을 차단하고, 신속한 대응이 가능하다. |
| **이력 관리(Audit Trail)** | 금융권 규제 준수를 위한 장기 보관 및 보안 감사 시 추적 가능하여 안정성을 보장한다. |
| **AI 기반 모니터링** | 자동화된 문제 탐지로 운영 비용을 절감하고, 예측 분석을 통해 시스템 안정성을 높인다. |

# ****DevOps 및 자동화 전략****

## ****도입전안내말****

DevOps 및 자동화 전략은 어플리케이션의 개발, 배포, 운영을 일원화하여 빠르고 안정적인 서비스 제공을 목표로 합니다.  
EA(Enterprise Architecture) 관점에서 CI/CD(Continuous Integration/Continuous Deployment) 파이프라인과 Infrastructure as Code(IaC) 전략은 신속한 릴리즈, 일관된 환경 설정, 무중단 배포를 실현합니다.  
이를 통해 개발 주기 단축, 서비스 가용성 향상, 장애 대응 시간 감소를 기대할 수 있으며, 최적화된 배포 환경을 구축할 수 있습니다.

## ****DevOps 및 자동화 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **8.1 CI/CD 파이프라인** | 8.1.1 Jenkins | 오픈소스 CI/CD 도구로, 자동 빌드, 테스트, 배포를 지원 | Jenkins, Maven, Gradle | 다양한 플러그인으로 확장성이 뛰어나며, DevOps 환경에 최적화됨 |
|  | 8.1.2 GitLab CI | GitLab과 연동하여 코드 변경 시 자동으로 CI/CD를 수행하며, 병렬 빌드와 컨테이너 배포가 가능 | GitLab, Docker, Kubernetes | 분산형 개발 환경에서의 협업과 자동화된 배포에 강점 |
|  | 8.1.3 Argo CD | GitOps 방식을 지원하는 Kubernetes 전용 CD 도구로, YAML 파일을 Git에 반영하여 배포 관리 | Argo CD, Kubernetes | Git 레포지토리 기반으로 선언적 배포가 가능하며, Kubernetes 네이티브 환경에 최적화 |
|  | 8.1.4 Spinnaker | 멀티클라우드 환경에서 무중단 배포를 지원하는 CD 도구로 Netflix에서 개발 | Spinnaker, AWS, GCP, Kubernetes | 블루-그린 배포, 카나리 배포, 롤링 업데이트를 통해 고가용성 유지 |
| **8.2 Infrastructure as Code (IaC)** | 8.2.1 Terraform | 인프라를 코드로 관리하며, 멀티 클라우드에 걸쳐 일관된 인프라 생성 및 관리 가능 | Terraform, AWS, Azure, GCP | 클라우드 환경 간 일관된 배포가 가능하며, 버전 관리가 가능함 |
|  | 8.2.2 Ansible | 서버 설정 관리 및 애플리케이션 배포를 자동화하여 인프라 프로비저닝을 최적화 | Ansible, SSH, YAML | 멀티 노드 환경에서 대량의 설정 변경을 자동화하여 운영 비용 절감 |
|  | 8.2.3 Helm | Kubernetes 애플리케이션의 설치 및 관리 도구로, YAML을 통해 배포 정의 | Helm Charts, Kubernetes | 복잡한 Kubernetes 애플리케이션을 쉽게 배포하고, 업데이트 및 롤백이 용이함 |
|  | 8.2.4 Puppet | 서버 구성 관리 및 애플리케이션 배포 자동화 도구로, 인프라의 상태를 코드로 정의하고 일관성을 유지 | Puppet, Hiera, Puppet Forge | 대규모 서버 환경에서 설정 관리의 일관성을 보장하며, 변화 관리가 용이함 |
| **8.3 컨테이너 오케스트레이션** | 8.3.1 Kubernetes | 컨테이너의 배포, 확장, 관리, 복구를 자동화하는 오픈소스 플랫폼 | Kubernetes, Kubectl, Helm | MSA(Microservice Architecture) 기반 애플리케이션에 최적화된 확장 및 자가 복구 지원 |
|  | 8.3.2 Docker Swarm | Docker의 오케스트레이션 솔루션으로, 클러스터링된 Docker 노드의 배포와 관리 | Docker, Docker Compose | 빠른 컨테이너 배포와 간단한 설정으로 경량화된 오케스트레이션 가능 |
|  | 8.3.3 OpenShift | RedHat에서 개발한 Kubernetes 기반 PaaS 플랫폼으로, 엔터프라이즈 환경에 최적화된 관리 및 보안 제공 | OpenShift, RedHat, Kubernetes | 대규모 엔터프라이즈 환경에 최적화된 보안 및 멀티테넌시 지원 |

### ****(실제)**** DevOps 및 자동화 전략 설계

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **8.1 CI/CD 파이프라인** | Jenkins 기반 자동화 | 오픈소스 기반의 플러그인 확장형 CI 서버로, Git Hook 기반 자동 빌드/테스트/배포 지원 | Jenkins, Maven, Gradle | 커스터마이징 유연성, 복잡한 파이프라인 구성 가능 |
|  | GitLab CI/CD | Git 저장소와 통합된 DevOps 플랫폼으로 Pull Request 이벤트 기반 파이프라인 실행 | GitLab Runner, Docker | Git 기반 코드 추적과 DevSecOps 통합에 적합 |
|  | Argo CD (GitOps) | Git 저장소 상태를 기준으로 쿠버네티스 클러스터 자동 동기화 | ArgoCD, Helm, Kustomize | 배포 일관성과 선언형 인프라 적용으로 무중단 배포 구현 |
|  | Spinnaker | 카나리/블루그린 배포 자동화 도구, 다양한 멀티 클라우드 환경 대응 | Spinnaker, GCP, AWS | 무중단 운영환경과 고가용 서비스 연속성 보장 |
| **8.2 Infrastructure as Code (IaC)** | Terraform | 선언적 코드 기반 멀티 클라우드 인프라 프로비저닝 | Terraform, AWS/GCP, tfstate | 환경 간 불일치 해소, 인프라 구성 재현성 확보 |
|  | Ansible | Agentless 구성관리 및 패치 자동화 | Ansible, YAML, SSH | 반복 작업 제거, 대규모 환경 운영 단순화 |
|  | Helm | K8s 배포 패키지 템플릿화 도구 | Helm Chart, Kubernetes | 마이크로서비스 운영 배포의 표준화 및 Rollback 지원 |
|  | Puppet | 중앙 집중형 구성관리 도구 | Puppet, PuppetDB | Legacy 환경에서도 구성 관리 자동화 지원 |
| **8.3 컨테이너 오케스트레이션** | Kubernetes | 클러스터 기반 컨테이너 배치/관리 자동화 | K8s, Helm, CRD | 자가 복구, 자동 확장, MSA 환경 최적 |
|  | Docker Swarm | 소규모 클러스터에 적합한 경량 오케스트레이션 | Docker, Compose | 빠른 구축과 쉬운 설정으로 PoC 환경 대응 |
|  | OpenShift | PaaS 환경에 적합한 K8s 기반 기업용 플랫폼 | OpenShift, Keycloak, Tekton | 멀티테넌시, 보안정책, 인증통합 지원 |

#### ✅ DevOps 전략적 적용 시 고려 요소

|  |  |
| --- | --- |
| 항목 | 전략적 적용 방안 |
| **CI/CD 관리 기준** | 브랜치 전략(Git Flow), PR 기반 자동화 트리거 설정, 빌드/테스트/배포 분리형 스테이지 구성 |
| **배포 전략** | 블루-그린 배포, 롤링업데이트, 카나리 테스트 기반 무중단 배포 적용 |
| **오류 롤백 설계** | Helm + GitOps + Versioning을 활용한 자동화된 Rollback 시나리오 구축 |
| **보안 강화 요소** | DevSecOps 파이프라인 내 정적 분석(SAST), 취약점 스캐닝, 코드서명 적용 |
| **비용 효율화** | 도커 캐시 재활용, 멀티 파이프라인 병렬 처리로 빌드 시간 단축 |
| **모니터링 연동** | CI/CD 성공률, 평균 배포 시간, 실패 건수 등을 Prometheus + Grafana로 시각화 |

#### ✅ 추가 설계 제안 (DevSecOps 및 운영 최적화)

|  |  |
| --- | --- |
| 항목 | 내용 |
| **DevSecOps 통합** | CI 시점 SonarQube + Snyk + Trivy 등 보안 검사 자동화 |
| **배포 리허설 환경 구성** | Stage → Pre-Prod → Prod로 계층별 승인 배포 구조 설정 |
| **자동 회복 구성** | Argo Rollouts 기반, 실패 시 자동 롤백 설정 |
| **서비스 매쉬 연계** | Istio, Linkerd를 통한 마이크로서비스 통신 보안 및 배포 제어 |

#### ✅ 시사점

* NH 농협지주 데이터플랫폼 구축사업은 **마이크로서비스, API 중심 아키텍처, 쿠버네티스 운영**이 핵심이며, 이에 따라 **DevOps 파이프라인의 자동화 정교도**가 시스템 운영 안정성에 직결됩니다.
* **CI/CD 자동화 → IaC 구성 → 보안 연계 → 운영 모니터링**이 유기적으로 연동되어야 안정적이고 반복 가능한 플랫폼 운영이 가능합니다.
* 특히 **GitOps 기반 운영 체계와 ArgoCD + Helm**을 통한 무중단 배포 체계를 필수적으로 구축해야 합니다.

### [DevOps 및 자동화 전략을 위한 테크니컬 서버 구성도 - Text 기반 도식]

|  |
| --- |
| [개발자 PC]  │  └──[Git Repository (GitLab, GitHub)]  │  └── Webhook Trigger  │  ▼  [CI Server (Jenkins, GitLab CI)]  ├─ Code Checkout  ├─ Build (Maven, Gradle)  ├─ Static Code Analysis (SonarQube)  └─ Container Image Build (Docker)  │  ▼  [Artifact Repository (Nexus, JFrog)]  │  └─ Build Artifact 저장  ▼  [CD Server (ArgoCD, Spinnaker)]  ├─ YAML 기반 배포 정의  ├─ GitOps 기반 자동 배포  ├─ Canary/Blue-Green 배포  └─ Rollback 관리  │  ▼  [Kubernetes Cluster (Tanzu / OpenShift)]  ├─ APP Runtime 실행  ├─ Monitoring Agent 내장 (Prometheus Node Exporter)  └─ Logging Agent 내장 (Fluentd / Sidecar)  │  ├────────▶ [Monitoring Server (Prometheus + Grafana)]  │ └─ 실시간 메트릭, Alerting, SLA 대시보드  │  └────────▶ [Logging Server (ELK / Fluentd + Elasticsearch)]  └─ 통합 로그 수집, 필터링, 보관 |
|  |

#### ✅ [기능별 DevOps 서버 구성 요소 정의]

|  |  |  |
| --- | --- | --- |
| 서버/컴포넌트 | 주요 역할 | 연계 기술 |
| **Git Repository (GitLab)** | 개발자 코드 저장 및 변경 감지 | Git, Webhook |
| **CI Server (Jenkins)** | 빌드, 테스트, 보안 스캔 등 수행 | Maven, Gradle, SonarQube, Docker |
| **Artifact Repository (Nexus)** | 빌드 산출물 및 Docker 이미지 저장 | Docker Registry, Maven Repo |
| **CD Server (ArgoCD)** | GitOps 방식으로 Kubernetes 배포 자동화 | Git, YAML, Helm, Kustomize |
| **Kubernetes Cluster** | 마이크로서비스 및 API 배포 플랫폼 | Docker, Helm, ServiceMesh |
| **Monitoring Server (Prometheus + Grafana)** | 시스템 메트릭 수집 및 시각화 | Node Exporter, Alertmanager |
| **Logging Server (Fluentd + Elasticsearch + Kibana)** | 모든 계층의 로그 통합 수집 및 분석 | Fluentd, Logstash, Kibana |

#### ✅ [전략적 의미 및 적용 포인트]

|  |  |
| --- | --- |
| 항목 | 전략적 효과 |
| **CI/CD 서버 분리** | 개발 → 테스트 → 운영환경 분리를 통해 안정적인 릴리즈 체계 확보 |
| **GitOps 기반 CD** | 배포 추적성과 복원력을 확보하며, YAML 기반 선언형 운영 체계 정립 |
| **Container 기반 아키텍처** | 이식성과 확장성 확보, 장애 격리에 강한 인프라 구조 완성 |
| **Logging/Monitoring 서버 통합** | 시스템 전반의 장애 감지, 사용자 행위 추적, SLA 관리 가능 |

### ✅ ****CI/CD 통합 파이프라인 흐름도 (텍스트 기반 도식)****

[개발자 코드 작성]

│

└─▶ [Git Repository (GitLab/GitHub)]

│

└─Webhook Trigger (Push/PR 이벤트)

│

▼

[CI Server (Jenkins / GitLab CI)]

├─ 코드 Pull

├─ 정적 코드 분석 (SonarQube)

├─ 단위 테스트 (JUnit, PyTest 등)

├─ 빌드 (Maven / Gradle)

└─ Docker 이미지 생성

│

▼

[Artifact Repository (Nexus / JFrog)]

├─ Docker Image 저장

└─ 패키지(.jar/.war) 저장

│

▼

[CD Server (Argo CD / Spinnaker)]

├─ GitOps 기반 배포 정의 (Helm / Kustomize)

├─ Canary / Blue-Green 전략 적용

├─ 대상 클러스터 상태 모니터링

└─ 롤백 관리

│

▼

[Kubernetes 클러스터 (Tanzu / OpenShift)]

├─ Microservices 배포

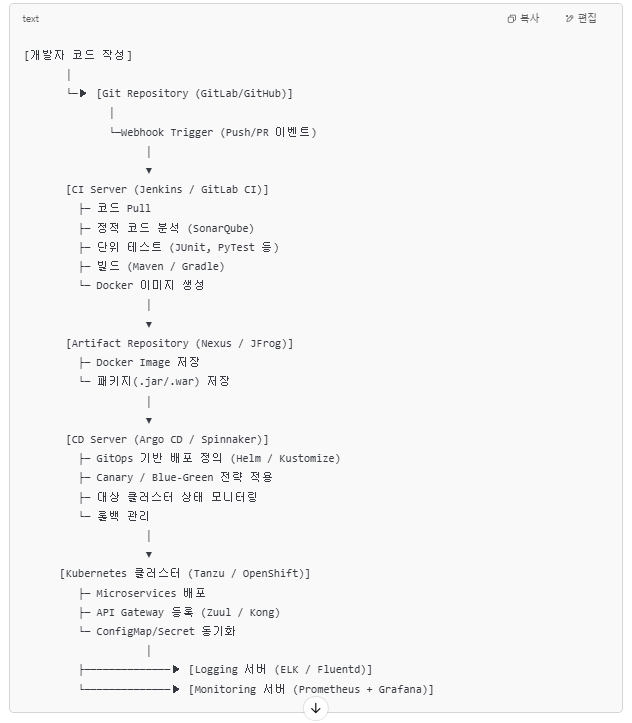
├─ API Gateway 등록 (Zuul / Kong)

└─ ConfigMap/Secret 동기화

│

├──────────────▶ [Logging 서버 (ELK / Fluentd)]

└──────────────▶ [Monitoring 서버 (Prometheus + Grafana)]



#### ✅ ****주요 연계 기술 스택****

|  |  |  |
| --- | --- | --- |
| 단계 | 주요 도구/기술 | 역할 |
| Git 저장소 | GitLab, GitHub | 코드 관리 및 CI 트리거 |
| CI 엔진 | Jenkins, GitLab CI | 빌드, 테스트, 보안 검사 |
| 정적 분석 | SonarQube | 코드 품질 분석 |
| 테스트 | JUnit, Mockito | 유닛/통합 테스트 수행 |
| 빌드 도구 | Maven, Gradle | 애플리케이션 빌드 |
| 컨테이너 | Docker | 이미지 생성 및 배포 단위 구성 |
| 아티팩트 저장소 | Nexus, JFrog Artifactory | 빌드 결과물 저장 |
| CD 엔진 | ArgoCD, Spinnaker | GitOps 기반 자동 배포 |
| 배포 환경 | Kubernetes, OpenShift | 마이크로서비스 실행 |
| 모니터링 | Prometheus, Grafana | 실시간 성능 모니터링 |
| 로깅 | Fluentd, ELK Stack | 통합 로그 수집 및 분석 |

#### ✅ ****전략적 의미****

* **배포 자동화**: Git → Build → Deploy의 완전 자동화로 릴리즈 시간 단축
* **품질 보장**: 자동화된 테스트 및 정적 분석으로 코드 안정성 확보
* **보안 내재화**: 코드 수준에서 취약점 탐지 → 운영 반영 전 검증
* **유연한 롤백**: GitOps 구조 기반으로 배포 이력 관리 및 손쉬운 되돌리기
* **멀티환경 대응**: Dev, QA, Stage, Prod 환경을 파이프라인 내 자동 분기 처리

### ✅ GitOps 기반 자동화 구성도 (텍스트 기반 도식)

[개발자]

└──▶ Git Commit/Push

│

▼

[Git Repository (GitLab / GitHub)]

└──▶ [Argo CD / Flux CD] ←───┐

│ │

│ (자동 주기적 동기화 or Webhook)

▼ │

[Deployment YAML, Helm Charts, Kustomize]

│ │

▼ │

[CI Server (Jenkins / GitLab CI)] ─────────────┐

├─ 빌드 및 테스트 │

├─ 컨테이너 이미지 생성 │

└─ 이미지 Push → [Docker Registry] │

▼

[Kubernetes Cluster (Tanzu / OpenShift)]

├─ Application Deployment

├─ ConfigMap, Secret 자동 주입

├─ TLS/Ingress 설정

├─ Auto Rollback & Self-healing

└─ 상태 피드백

┌───────────────────────────────┐

│ Observability │

│ ├─ Logging (ELK / Fluentd) │

│ └─ Monitoring (Prometheus) │

└───────────────────────────────┘



#### ✅ GitOps 구성요소 설명

|  |  |
| --- | --- |
| 구성요소 | 설명 |
| **Git Repository** | YAML, Helm, Kustomize 기반의 선언형 배포 정의 관리 |
| **Argo CD / Flux CD** | Git 상태와 클러스터 상태를 주기적으로 비교하여 자동으로 동기화 및 배포 수행 |
| **CI Server** | 코드 변경에 따라 테스트, 빌드, 컨테이너 이미지 생성 수행 (예: Jenkins, GitLab CI) |
| **Docker Registry** | 빌드된 컨테이너 이미지 저장소 (Nexus, JFrog 등) |
| **Kubernetes Cluster** | 배포 대상 인프라: Tanzu, OpenShift, GKE 등 |
| **Observability** | 로그/모니터링 수집기 (ELK, Fluentd, Prometheus 등) |

#### ✅ 전략적 의미

|  |  |
| --- | --- |
| 항목 | 전략적 의미 |
| **신뢰성 강화** | Git의 변경 이력을 기준으로 상태를 추적할 수 있어 감사 및 복구가 용이 |
| **무중단 배포** | Blue-Green, Canary 등 배포 전략을 GitOps로 선언 가능 |
| **운영 자동화** | 배포 정의와 실제 클러스터 상태 간 자동 동기화를 통해 운영 인력 부담 감소 |
| **보안성 향상** | Git에 의한 변경 감사를 통해 인가되지 않은 배포 방지 가능 |
| **DevSecOps 실현** | Git + CI + CD + 관찰성(Logging/Monitoring) 연계로 완성형 DevSecOps 체계 구현 |

### ✅ DevSecOps 통합 흐름도 (텍스트 기반 도식)

[개발자 (Dev)]

└──▶ Git Commit/Push

│

▼

[Source Repository (GitHub / GitLab)]

│

├─▶ 1차 보안 검사 (SAST - 정적 코드 분석)

│ ⤷ SonarQube, Checkmarx, CodeQL

▼

[CI 서버 (Jenkins / GitLab CI)]

├─ 빌드 & 유닛 테스트

├─ 의존성 취약점 분석 (SCA: Software Composition Analysis)

│ ⤷ OWASP Dependency-Check, Snyk, Nexus IQ

├─ 컨테이너 이미지 생성

├─ 이미지 취약점 검사 (Container Scanning)

│ ⤷ Trivy, Clair, Anchore

└─ 이미지 업로드 → [Secure Docker Registry]

│

▼

[CD 단계 - Argo CD / Spinnaker]

├─ 배포 전 검증:

│ - 배포 정책 검사 (OPA/Gatekeeper)

│ - Secret 및 ConfigMap 스캔

│

├─ Canary/Blue-Green 무중단 배포

└─ 클러스터 보안 검증 (Kube-bench, Kube-hunter)

▼

[Kubernetes Cluster (Tanzu, OpenShift)]

├─ Runtime 보안 모니터링 (Falco, Sysdig)

├─ 네트워크 보안 제어 (Cilium, Calico, Istio)

├─ API Gateway 보안 적용 (OAuth2.0, Rate Limit, JWT)

└─ 관측 기반 대응

▼

[Observability/Monitoring]

├─ Logging (ELK/Fluentd/Splunk)

├─ Monitoring (Prometheus, Grafana)

└─ Alerting (PagerDuty, Opsgenie)

▼

[Security Team (SecOps)]

├─ 자동화된 감사 리포트 수신

├─ 위협 탐지 및 정책 업데이트

└─ 컴플라이언스 대응 (ISMS-P, GDPR, 내부감사 등)



#### ✅ 핵심 설계 포인트

|  |  |  |
| --- | --- | --- |
| 구분 | 설계 요소 | 설명 |
| **코드단 보안** | SAST, SCA | 코드 취약점 및 오픈소스 라이브러리 보안 점검 |
| **CI 단계** | 이미지 취약점 검사 | 빌드 시점에 Docker 이미지의 보안 검증 |
| **CD 단계** | OPA 정책 검증, Canary 배포 | 배포 전 정책 준수 및 안정성 확보 |
| **운영단 보안** | Falco, Kube-bench | 런타임 보안 및 쿠버네티스 보안 상태 점검 |
| **통합 관측** | Logging + Monitoring + Alerting | 이상 탐지 및 대응 자동화 기반 운영 안정화 |
| **보안 감사** | 자동 리포트 및 규제 준수 | 금융권 수준의 보안 요구사항 충족 가능 |

#### ✅ 전략적 의미

* **보안의 자동화 및 내재화**로 보안에 대한 사후 대응이 아닌 **사전 예방 체계 구축**
* **CI/CD와 보안 도구의 통합**을 통해 개발 생산성과 보안성을 동시에 확보
* **금융권 규제 대응 (ISMS-P, 전자금융감독규정 등)** 기반 설계로 프로젝트 리스크 최소화
* **개발-배포-운영 전체 보안 흐름 투명화**로 감사 및 컴플라이언스 대응 강화

### ****카나리/블루그린 배포 전략 구조****

#### ✅ [1] 배포 전략 비교 개요

|  |  |  |
| --- | --- | --- |
| 구분 | 카나리 배포 (Canary Deployment) | 블루그린 배포 (Blue-Green Deployment) |
| 개념 | 신규 버전(V2)을 소수 트래픽에만 점진적으로 적용하여 모니터링 후 전체 전환 | 기존 버전(V1)과 신규 버전(V2)을 완전히 분리하고, 스위칭 방식으로 한 번에 전환 |
| 주요 목적 | 신규 기능의 **점진적 오픈** 및 이슈 사전 탐지 | **무중단 배포** 및 전체 스위칭의 안정성 확보 |
| 트래픽 전환 | 점진적 비율 조절 (예: 5% → 25% → 50% → 100%) | 스위치 방식 전환 (Green → Blue) |
| 롤백 대응 | 일부 트래픽에서만 실행되므로 **빠른 롤백** 가능 | 전체 스위칭 전이므로 **즉시 복구** 가능 |
| 적용 기술 | Argo Rollouts, Istio, Service Mesh, NGINX Ingress | Spinnaker, Kubernetes Ingress, Route 관리 |
| 적합 상황 | 신규 기능 실험, 점진적 오픈이 필요한 경우 | 단일 릴리즈, 단순한 배포 전환이 필요한 경우 |

#### ✅ [2] 카나리 배포 구조도 (텍스트 기반)

[사용자 트래픽]

│

▼

[Ingress / API Gateway]

│

├──▶ 90% ─────▶ [K8s 서비스: app-v1 (Blue)]

└──▶ 10% ─────▶ [K8s 서비스: app-v2 (Canary)]

│

└─▶ [Prometheus, Grafana → 에러율/응답속도 모니터링]

│

└─▶ 문제 없음 확인 시 → 비율 증가



**설명**:

* Canary 버전인 app-v2로 10% 트래픽만 유입
* 지속적인 성능 모니터링 후 점진적으로 트래픽 확장
* 문제 발생 시 빠르게 100%를 app-v1로 롤백

#### ✅ [3] 블루그린 배포 구조도 (텍스트 기반)

text

복사편집

[사용자 트래픽]

│

▼

[Ingress / Load Balancer]

│

├──▶ [K8s 서비스: app-v1 (Blue)] ← 현재 운영 중

└──▶ [K8s 서비스: app-v2 (Green)] ← 신규 배포 대기

<배포 전>

모든 트래픽: app-v1

<배포 후>

스위치 전환 → 모든 트래픽: app-v2



**설명**:

* app-v1과 app-v2는 별도 인스턴스로 **완전히 독립된 환경**
* 스위칭 시 단 한 줄의 설정으로도 전체 트래픽 전환
* 기존 버전은 **즉시 롤백 가능**

#### ✅ [4] 전략적 의미

|  |  |
| --- | --- |
| 항목 | 전략적 효과 |
| **무중단 서비스** | 실시간 사용 중단 없이 배포 가능 (특히 금융권 필수 요소) |
| **장애 최소화** | 트래픽 분산 및 관측 도구 기반 자동 롤백 |
| **보안성 향상** | 버전별 취약점 점검 후 정식 전환 구조 |
| **DevSecOps 정합성** | 배포 프로세스에 보안 검사, 트래픽 제어 포함 가능 |

#### ✅ [5] 적용 기술 스택 제안

|  |  |
| --- | --- |
| 레이어 | 기술 |
| 배포 관리 | Argo CD, Spinnaker, FluxCD |
| 트래픽 제어 | Istio, Linkerd, Envoy, NGINX Ingress |
| 모니터링 | Prometheus, Grafana, Datadog |
| 정책 제어 | Open Policy Agent (OPA), Gatekeeper |
| 인증 제어 | OAuth2.0, JWT, Keycloak |

### ✅ [1] Service Mesh 연계 구성도 (텍스트 기반 도식)

[사용자 또는 외부 채널 요청]

│

▼

[API Gateway / Ingress Gateway (Istio)]

│

▼

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

| Namespace: marketing |

| ┌───────────────────────┐ |

| │ microservice-a │◀┐

| │ (통합 데이터 조회) │ │

| │ + envoy sidecar │ │

| └───────────────────────┘ │

| │ │ Istio

| ▼ │ mTLS +

| ┌───────────────────────┐ │ 인증/정책

| │ microservice-b │ │

| │ (고객 행동 분석) │◀┘

| │ + envoy sidecar │

| └───────────────────────┘

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

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

| Namespace: etl |

| ┌───────────────────────┐ |

| │ microservice-c │◀┐

| │ (ETL 처리기) │ │

| │ + envoy sidecar │ │

| └───────────────────────┘ │

| │ │ Telemetry

| ▼ │ 정책설정

| ┌───────────────────────┐ │

| │ microservice-d │◀┘

| │ (데이터 적재기) │

| │ + envoy sidecar │

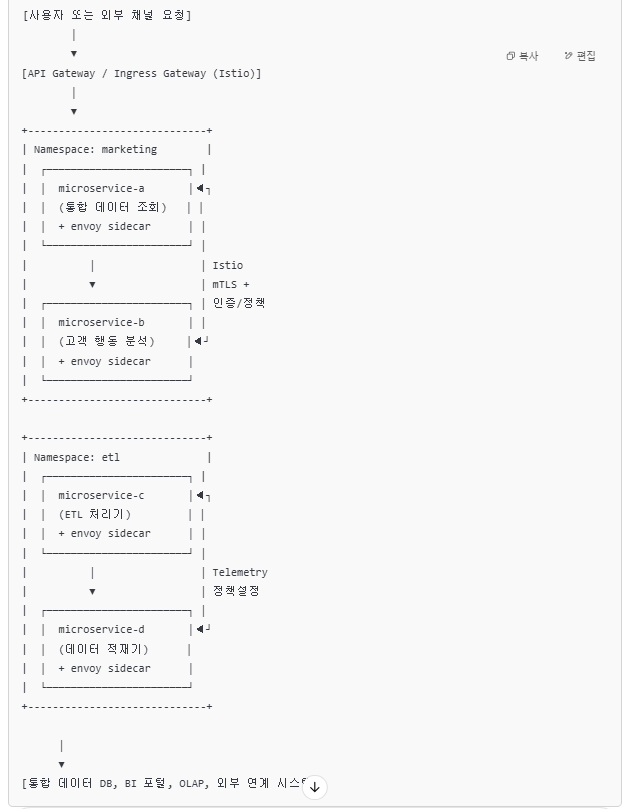
| └───────────────────────┘

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

│

▼

[통합 데이터 DB, BI 포털, OLAP, 외부 연계 시스템]



#### ✅ [2] 주요 설계 포인트

| **구성 요소** | **역할** |
| --- | --- |
| **Envoy Sidecar** | 각 마이크로서비스에 부착되어 트래픽 라우팅, 인증, 모니터링 수행 |
| **Istio Control Plane** | 트래픽 정책, 보안정책, 서비스 디스커버리, 인증, 모니터링 설정 |
| **Ingress Gateway** | 외부와 내부 메쉬를 연결하는 진입점 (API Gateway 역할 포함 가능) |
| **PeerAuthentication / AuthorizationPolicy** | 서비스 간 인증 및 권한 정책 정의 (mTLS 포함) |
| **VirtualService / DestinationRule** | 트래픽 분기, A/B 테스트, 카나리 배포 제어 |
| **Telemetry / Mixer / Prometheus / Grafana** | 로그, 메트릭 수집 및 실시간 모니터링 기반 연동 |

#### ✅ [3] 전략적 효과

| **전략 항목** | **의미** |
| --- | --- |
| **Zero Trust 보안** | 서비스 간에도 반드시 인증(mTLS) 수행하여 보안 강화 |
| **가시성 향상** | 서비스 간 호출 횟수, 응답시간, 실패율 등을 Prometheus+Grafana로 추적 |
| **정책 기반 제어** | 트래픽 우선순위, 레이트 리밋, 서킷 브레이커 적용 가능 |
| **장애 격리 및 회복성** | Fault Injection, Retry, Timeout 등으로 장애 격리 및 자가 회복 가능 |
| **서비스 디스커버리 자동화** | Kubernetes DNS 또는 Istio Registry 기반 자동 라우팅 처리 |

#### ✅ [4] 추천 기술 스택

| **영역** | **기술 스택** |
| --- | --- |
| Service Mesh Core | Istio, Linkerd, Kuma, Consul Connect |
| Sidecar Proxy | Envoy |
| 모니터링 | Prometheus, Grafana, Kiali, Jaeger |
| 보안정책 | mTLS, JWT, OPA(Gatekeeper), SPIRE(워크로드 인증) |
| 배포 자동화 | Argo Rollouts, Helm, GitOps + ArgoCD |

## ****마무리말****

DevOps 및 자동화 전략은 개발 속도와 안정성을 동시에 확보하는 필수적인 설계 요소입니다.  
CI/CD 파이프라인을 통해 코드 변경 사항이 실시간으로 빌드, 테스트, 배포되며, Infrastructure as Code(IaC)를 활용하여 인프라 설정의 일관성을 보장합니다.  
특히 Kubernetes와 Docker를 이용한 컨테이너 오케스트레이션은 MSA 환경에서의 확장성과 무중단 배포를 지원하며, ArgoCD와 Spinnaker는 무중단 배포 전략으로 안정적인 업데이트를 가능하게 합니다.  
이를 통해 금융권 시스템의 가용성과 보안을 보장하며, 운영 효율성을 극대화합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **CI/CD 파이프라인** | 지속적인 통합과 배포를 자동화하여 개발 주기를 단축하고, 인프라의 일관성을 유지한다. |
| **Infrastructure as Code (IaC)** | 클라우드 환경에서의 일관된 배포와 버전 관리가 가능하여, 확장성과 관리 용이성을 높인다. |
| **컨테이너 오케스트레이션** | Kubernetes를 통한 자동 확장 및 무중단 배포가 가능하며, 서비스 복원력이 극대화된다. |
| **GitOps 기반 배포** | ArgoCD를 통한 선언적 배포는 변경 이력을 추적 가능하게 하며, 롤백이 쉽다. |
| **멀티클라우드 지원** | Spinnaker와 Terraform을 활용하여 AWS, GCP, Azure 등 멀티클라우드에 유연하게 대응할 수 있다. |

# ****데이터 관리 전략****

## ****도입전안내말****

데이터 관리 전략은 EA(Enterprise Architecture) 아키텍처 설계에서 데이터의 저장, 처리, 접근, 보안을 체계적으로 관리하기 위한 중요한 요소입니다.  
특히 금융권과 같은 대량 트랜잭션을 처리하는 시스템에서는 데이터의 일관성, 무결성, 보안이 중요한 역할을 합니다.  
GUID 처리, Data Consistency, Data Partitioning 전략을 명확히 정의함으로써 대규모 데이터 처리와 보안성을 동시에 확보합니다.

## ****데이터 관리 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **9.1 GUID 처리 방안** | 9.1.1 글로벌 유니크 ID 생성 | 전 세계적으로 중복되지 않는 고유 식별자 생성 | UUID (Universally Unique Identifier), Snowflake | 글로벌 서비스 확장 시 중복 없이 ID 관리 가능하며, 데이터 추적이 용이함 |
|  | 9.1.2 트랜잭션 추적 | 각 트랜잭션의 고유 ID를 통해 이력 관리 및 감사 추적 가능 | Distributed Tracing (Spring Sleuth, Zipkin) | 금융 거래의 흐름을 추적하고, 장애 발생 시 원인 파악이 빠름 |
| **9.2 Data Consistency** | 9.2.1 Eventual Consistency | 분산 환경에서 데이터 일관성이 최종적으로 보장되도록 처리 (예: NoSQL DB, Kafka) | Cassandra, DynamoDB, Kafka | 글로벌 분산 시스템에서 성능 최적화를 극대화하며, 비동기 처리의 유연성을 높임 |
|  | 9.2.2 Strong Consistency | 모든 트랜잭션이 즉시 일관성을 보장하도록 처리 (예: RDBMS, ACID Transactions) | Oracle, MySQL, PostgreSQL | 금융 거래와 같은 실시간 데이터 처리에서 데이터 정합성을 보장함 |
|  | 9.2.3 Read-Write Split | 읽기 요청과 쓰기 요청을 분리하여 성능을 최적화 (Master-Slave 구조) | MySQL Replica, PostgreSQL Replication | 대량의 읽기 요청이 발생해도 성능 저하 없이 처리할 수 있음 |
| **9.3 Data Partitioning** | 9.3.1 Sharding | 데이터베이스를 여러 개의 샤드로 나누어 분산 처리 | MongoDB Sharding, Cassandra, MySQL | 대용량 데이터 처리 시 병목현상을 제거하고 수평 확장이 용이함 |
|  | 9.3.2 Replication | 데이터를 다수의 노드에 복제하여 가용성과 성능 향상 | MySQL Replication, PostgreSQL Streaming Replication | 장애 발생 시 빠른 복구가 가능하며, 읽기 성능이 극대화됨 |
|  | 9.3.3 Multi-Master Replication | 여러 마스터 노드에서 동시에 읽기와 쓰기 작업을 수행하여 성능 최적화 | Galera Cluster, CockroachDB | 다중 트랜잭션 처리 시 충돌 방지 및 고가용성 보장 |
| **9.4 데이터 보안** | 9.4.1 데이터 암호화 | 저장 시와 전송 시 데이터를 암호화하여 정보 유출 방지 | AES-256, RSA, TDE (Transparent Data Encryption) | 금융 정보 보호를 위한 필수적인 보안 조치, GDPR 및 PCI DSS 준수 가능 |
|  | 9.4.2 접근 제어 | 데이터베이스에 대한 접근 권한을 역할 기반으로 설정 | Role-Based Access Control (RBAC), ABAC | 민감 데이터에 대한 접근 제어를 강화하여 보안성을 높임 |
|  | 9.4.3 Audit Trail | 데이터 변경 이력을 추적하여 보안 사고 발생 시 분석 가능 | ELK Stack, Splunk, Oracle Audit Vault | 데이터 조작에 대한 추적이 가능하며, 법적 규제 준수를 지원 |
| **9.5 데이터 백업 및 복구** | 9.5.1 백업 정책 정의 | 주기적인 백업을 통해 장애 발생 시 데이터 손실 최소화 | Oracle RMAN, AWS S3 Backup, Azure Backup | 주기적인 백업과 다중 지역 백업으로 복구 가능성을 높임 |
|  | 9.5.2 복구 시나리오 정의 | 장애 발생 시 신속한 복구를 위한 복구 시나리오 설계 | Point-in-Time Recovery, Snapshot, Log-based Recovery | 금융권의 무중단 서비스를 위한 필수 요소이며, 신속한 복구가 가능함 |

## ****마무리말****

데이터 관리 전략은 금융권과 같은 대규모 트랜잭션 처리 시스템에서 안정성, 무결성, 확장성을 보장하기 위한 핵심적인 설계 요소입니다.  
GUID를 통해 글로벌 유니크한 트랜잭션 식별을 가능하게 하며, Data Consistency 전략으로 실시간 거래의 정합성을 보장합니다.  
또한 Sharding과 Replication을 통해 대용량 데이터를 분산 처리하여 성능 최적화를 이루고, 데이터 보안과 접근 제어를 통해 금융 규제에 대응합니다.  
마지막으로, 주기적인 백업과 복구 시나리오는 장애 발생 시 빠른 복구를 가능하게 하여, 시스템의 신뢰성을 극대화합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **GUID 처리 방안** | 글로벌 식별자를 통해 트랜잭션의 유일성을 보장하며, 장애 분석 시 빠른 추적이 가능하다. |
| **Data Consistency** | 금융 거래와 같은 실시간 데이터 처리에서 강력한 일관성을 보장하여 데이터 무결성을 유지한다. |
| **Data Partitioning** | 수평적 확장을 통해 대규모 데이터를 효율적으로 처리하며, 장애 시 빠른 복구가 가능하다. |
| **데이터 보안** | 강력한 암호화와 접근 제어를 통해 금융권의 보안 규제를 준수하고, 데이터 유출을 방지한다. |
| **백업 및 복구** | 장애 발생 시 빠른 복구를 통해 금융권의 무중단 서비스를 보장하며, 비즈니스 연속성을 유지한다. |

# 통합 설계 및 연계 전략

## ****도입전안내말****

통합 설계 및 연계 전략은 여러 시스템 간의 원활한 데이터 교환과 통신을 보장하기 위한 핵심 설계 요소입니다.  
EA(Enterprise Architecture) 관점에서 금융권과 같은 대규모 시스템에서는 다양한 애플리케이션, 데이터베이스, 외부 API와의 안정적이고 빠른 연계가 필수적입니다.  
이를 위해 API Gateway, Message Queue, Enterprise Service Bus(ESB), Batch Integration 등 다양한 통합 방안을 전략적으로 도입하여, 시스템 간 상호 운용성과 확장성을 극대화합니다.

## ****통합설계 및 연계 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **10.1 API Gateway** | 10.1.1 Zuul | Netflix OSS 기반의 API Gateway로, API 트래픽 관리와 로드 밸런싱을 지원 | Spring Cloud Zuul, Ribbon | 마이크로서비스 간의 통신을 효율적으로 관리하며, 부하 분산을 최적화함 |
|  | 10.1.2 Kong | 클라우드 네이티브 API Gateway로, 확장성과 보안에 강점이 있으며, 플러그인 구조로 다양한 기능 확장 가능 | Kong Gateway, Lua, OpenResty | 멀티 클라우드 환경에서도 유연한 API 라우팅이 가능하며, 보안 정책 관리가 용이함 |
|  | 10.1.3 Spring Cloud Gateway | Spring Boot와의 자연스러운 연동을 제공하며, 필터 체인을 통해 인증, 로깅, CORS 설정이 간편함 | Spring Cloud Gateway, Reactive Streams | MSA 구조에서의 API 관리를 간편하게 처리하며, Reactive 기반으로 높은 성능 제공 |
| **10.2 Message Queue** | 10.2.1 Kafka | 고속의 데이터 스트리밍과 분산 메시징을 지원하여, 실시간 데이터 처리에 강점 | Apache Kafka, ZooKeeper | 금융권의 거래 처리와 실시간 로그 분석에 최적화된 메시징 시스템 |
|  | 10.2.2 RabbitMQ | AMQP 프로토콜 기반의 메시지 브로커로, 비동기 메시징을 통해 시스템 간 안정적인 통신을 보장 | RabbitMQ, AMQP | 트랜잭션 처리와 비동기 이벤트 전송에 적합하며, 장애 시 메시지 재전송이 가능함 |
|  | 10.2.3 ActiveMQ | 오픈소스 메시지 브로커로 JMS(Java Message Service) 표준을 따르며, Queue 및 Topic 방식 모두 지원 | ActiveMQ, JMS | Java 기반의 통신에서 안정성과 확장성을 보장하며, 대규모 메시지 처리에 유리함 |
| **10.3 Enterprise Service Bus (ESB)** | 10.3.1 Mule ESB | API 통합 및 시스템 간 데이터 변환을 위한 오픈소스 ESB 플랫폼 | MuleSoft, DataWeave | 이기종 시스템 간의 데이터 호환성과 프로토콜 변환을 지원하여 통합 비용을 절감함 |
|  | 10.3.2 WSO2 ESB | 클라우드 네이티브 ESB 플랫폼으로, REST, SOAP, JMS 통신을 모두 지원 | WSO2 Integration Studio | RESTful API와 SOAP 통신의 중앙 집중 관리가 가능하며, 멀티 클라우드 환경에 최적화됨 |
|  | 10.3.3 TIBCO ESB | 엔터프라이즈급 통합 플랫폼으로, 실시간 메시징, 프로세스 관리, 데이터 통합을 지원 | TIBCO BusinessWorks, EMS | 금융권 대형 시스템 간 실시간 데이터 동기화와 복잡한 프로세스 처리에 최적화됨 |
| **10.4 Batch Integration** | 10.4.1 Spring Batch | 대량의 데이터를 일괄 처리하며, 트랜잭션 관리와 재시도 기능을 제공 | Spring Batch, Spring Boot | 금융권의 대규모 데이터 처리 시 안정성과 무결성을 보장함 |
|  | 10.4.2 Apache NiFi | 실시간 데이터 스트림 처리 및 변환에 최적화된 오픈소스 도구 | Apache NiFi, Kafka, HDFS | 복잡한 데이터 흐름을 시각적으로 관리하며, 실시간 처리에 강점이 있음 |
|  | 10.4.3 Talend Data Integration | 클라우드 및 온프레미스 환경에서의 데이터 통합 및 일괄 처리 지원 | Talend Studio, Talend Cloud | 다양한 데이터 소스 간의 통합을 시각적으로 설정 가능하며, ETL 처리에 최적화됨 |
| **10.5 데이터 연계 표준화** | 10.5.1 RESTful API | HTTP 표준을 기반으로 시스템 간 통신을 표준화 | Spring REST, Jersey, JAX-RS | 이기종 시스템 간의 통신을 단일 표준으로 통합하여 유지보수를 간소화함 |
|  | 10.5.2 gRPC | 고성능 원격 프로시저 호출(Remote Procedure Call)을 지원하는 Google의 오픈소스 프레임워크 | gRPC, Protocol Buffers | 고속의 바이너리 전송으로 대규모 마이크로서비스 통신에 적합하며, 네트워크 대역폭을 절약함 |
|  | 10.5.3 GraphQL | 클라이언트가 원하는 데이터만 요청할 수 있도록 설계된 쿼리 언어 | Apollo Server, Hasura | 복잡한 데이터 조회를 최적화하며, Over-fetching 및 Under-fetching 문제를 해결함 |

## ✅ [10.2 통합설계 및 연계전략 - 서버 구성 포함]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 핵심 기술 스택 | 연계 대상 서버 | 전략적 의미 |
| **10.1 API Gateway** | 10.1.1 Spring Cloud Gateway | 통합 데이터 AP → 포털/마케팅/OLAP으로 REST API 연계 및 인증/라우팅 처리 | Spring Cloud Gateway, JWT, OAuth2 | 통합 데이터 AP 서버 | Spring 기반 마이크로서비스에 최적화된 인증 흐름 적용 |
|  | 10.1.2 Kong Gateway | 외부 계열사 또는 BI 포털 Proxy → 내부 API 요청 시 보안 정책 적용 | Kong Gateway, OpenResty, OAuth2 | 대외 연계 Gateway, BI Proxy WEB | 외부 요청에 대해 스로틀링 및 인증 연동 처리 가능 |
|  | 10.1.3 Zuul (내부 특화) | 내부 서비스 간 경량 API Gateway | Zuul, Ribbon | 내부 WAS 연계 라우팅 서버 | 경량 구성을 통한 서비스 경로 제어 및 필터링 제공 |
| **10.2 Message Queue** | 10.2.1 Kafka | 빅스퀘어 → 마케팅시스템, OLAP 실시간 이벤트 전송 | Apache Kafka, Zookeeper | 마케팅 HUB, E-마케팅, OLAP 서버 | 대용량 로그 스트림 처리 및 비동기 이벤트 연계 |
|  | 10.2.2 RabbitMQ | 마케팅 → UMS/배너로의 실시간 응답 메시징 | RabbitMQ, AMQP | e-마케팅, UMS, 올원뱅크 | 이벤트 기반 메시지 전송으로 사용자 경험 향상 |
|  | 10.2.3 Redis Stream | 고속 임시 이벤트 처리 | Redis Stream | 마케팅 HUB, ETL | 성능 중심 경량 스트림 처리용 보완 메시징 |
| **10.3 Enterprise Service Bus (ESB)** | 10.3.1 Mule ESB | 데이터 정제 흐름 간 변환, 프로토콜 매핑 처리 | MuleSoft, DataWeave | 통합 데이터 ETL, FOS 연계 서버 | 메시지 포맷 변환, 계열사 연계 유연성 강화 |
|  | 10.3.2 WSO2 ESB | SOAP ↔ REST ↔ JMS 프로토콜 통합 지원 | WSO2 Integration Studio | 마케팅 HUB, 외부 연계서버 | 다양한 채널 기반 메시지 처리에 적합 |
| **10.4 Batch Integration** | 10.4.1 Spring Batch | 마케팅 데이터/성과/고객 타겟 데이터 일괄 처리 | Spring Batch, Control-M | 통합 데이터 ETL 서버 | 트랜잭션 단위 일괄처리와 이력성 확보 |
|  | 10.4.2 IBM DataStage | 계열사/내부 데이터 정제 및 마트 적재 | IBM DataStage, Oracle Client | 통합 데이터 ETL 서버 | 전통적 ETL 툴로 대량 데이터 처리 신뢰성 확보 |
|  | 10.4.3 Apache NiFi | 실시간/배치 병합 데이터 흐름 제어 | Apache NiFi, Kafka, HDFS | 통합 데이터 ETL / AP | 실시간 이벤트 흐름 기반 처리 자동화에 적합 |
| **10.5 데이터 연계 표준화** | 10.5.1 RESTful API | 통합 데이터 AP ↔ 포털/UI ↔ OLAP JSON 연동 | Spring REST, Swagger, OAuth2 | 통합 데이터 AP, 포털 UI, OLAP, 외부 API | 표준화된 REST 설계를 통한 확장 용이 |
|  | 10.5.2 gRPC | 서비스 간 고성능 RPC 통신 | gRPC, Protobuf | SpringBoot Microservices | 고속 바이너리 통신으로 내부 MSA 서비스 효율화 |
|  | 10.5.3 GraphQL | 포털 맞춤형 질의 API 구성 | Apollo Server, GraphQL Gateway | 통합 데이터 포털, BI 포털 | 사용자 정의형 데이터 요청 처리에 적합 |

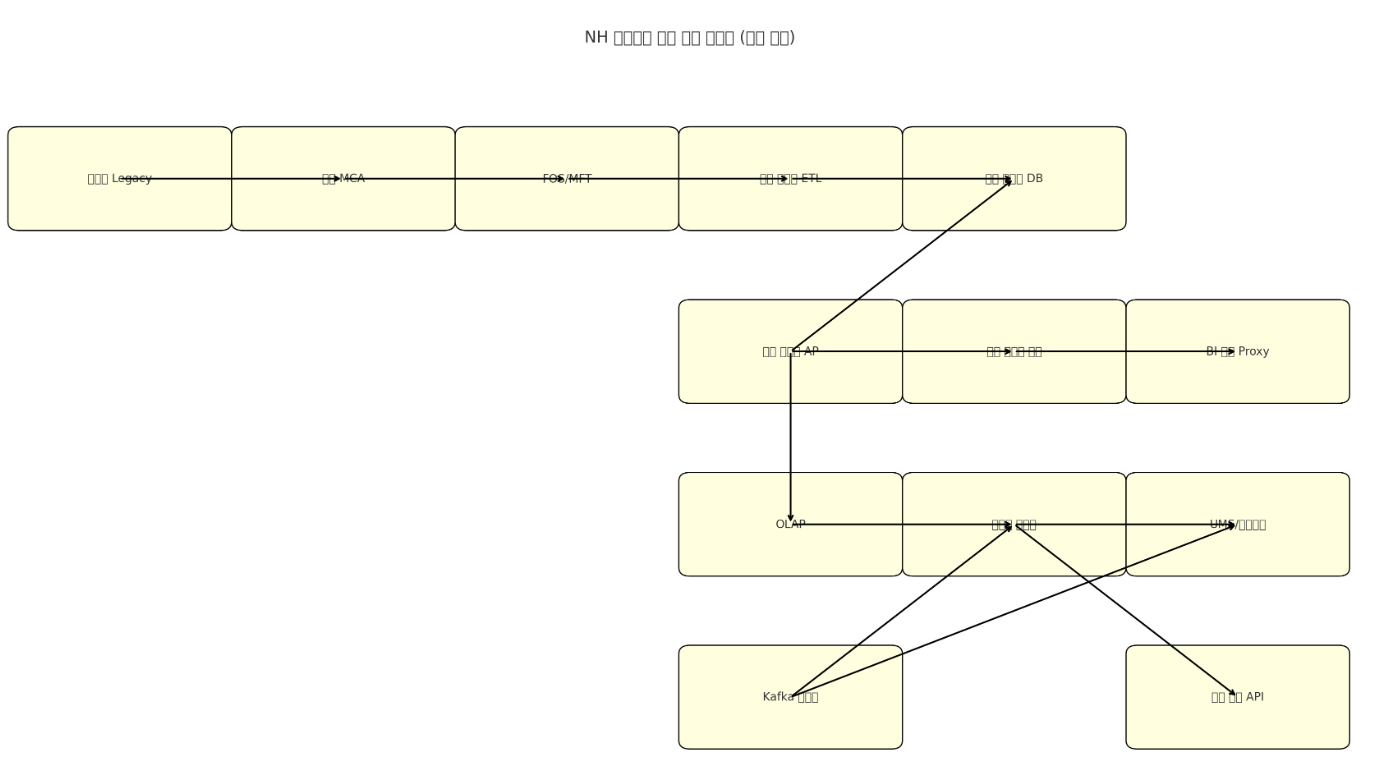
### 🧩 서버별 주요 연계 흐름 예시 (주요 대상 포함)

|  |  |  |
| --- | --- | --- |
| 서버명 | 주요 연계 방식 | 연계 대상 |
| **통합 데이터 AP** | REST API, OAuth2 | 포털 UI, OLAP, e-마케팅, 외부 API |
| **통합 데이터 ETL** | FOS, JDBC, MFT | 빅스퀘어, 계열사, 통합 DB, 마케팅 HUB |
| **통합 데이터 DB** | JDBC | AP, 포털, OLAP, 분석/보고 |
| **BI 포털 Proxy WEB** | HTTP API | 통합 데이터 포털 |
| **마케팅 시스템** | REST API, Kafka | UMS, OLAP, 외부채널, 올원뱅크 |
| **OLAP 시스템** | JDBC, Kafka | 통합 DB, ETL, 마케팅시스템 |
| **FOS 연계 서버** | FILE, MFT | 계열사 Legacy, ETL |
| **대외 MCA** | HTTP API | 통합 AP, 계열사, 외부 API |

### 🔍 전략적 시사점

* **API Gateway + Message Queue + ESB + Batch 통합 전략**은 통합 인터페이스의 핵심 기반
* **서버별 연계 역할과 특성에 맞춰 최적 기술 스택 적용**이 필요
* 연계 흐름은 **온라인(REST/gRPC) + 배치(FILE, FOS, JDBC) + 이벤트(Kafka)** 복합 구조
* 실시간 통합 데이터 요청을 위한 **API Gateway 기반 중앙 API 제어 계층 설계 필요**
* 데이터 적재/추출/변환 로직은 **ETL Batch + 실시간 연계 구성 병행**

## NH 농협지주 통합 연계 흐름도



### ✅ 전체 흐름 요약

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 주요 역할 | 연계 방식 |
| **계열사 Legacy 시스템** | 제3·4자 동의 고객정보, 거래·상품 정보 송신 | 배치 기반 FILE → FOS/MFT |
| **대외 MCA** | 실시간 API 처리 (동의 철회 등) | HTTP/API 연계 |
| **통합 데이터 ETL** | 배치 데이터 수집 및 적재 | FOS/MFT 및 JDBC |
| **통합 데이터 DB** | 통합 저장소 (Oracle HEX) | JDBC |
| **통합 데이터 AP** | API 기반 실시간 처리 및 분석 서비스 제공 | REST API, OAuth2 인증 |
| **통합 데이터 포털** | BI 분석 UI 제공, 사용자별 권한 접근 | JDBC + API 연계 |
| **OLAP/BI Proxy/마케팅 시스템** | 분석 결과 조회 및 실시간 캠페인 | Kafka, API, JDBC |

### 🔧 서버별 주요 기술 스택 (요약)

|  |  |
| --- | --- |
| 서버 | 기술 스택 |
| 통합 데이터 AP | Spring Boot, REST API, OAuth2/JWT, Kafka |
| 통합 데이터 ETL | IBM DataStage, Control-M, FOS/MFT |
| 통합 데이터 DB | Oracle, DB Safer, 보안 SW |
| 통합 포털 WEB | React/Thymeleaf, SpringBoot, xFrame5 |
| OLAP | JDBC, BI 솔루션 연계, 마케팅 분석 |
| Kafka 브로커 | Kafka, Zookeeper, Stream Processor |

### 온라인 배치 전체 연계흐름도

|  |
| --- |
| [계열사 Legacy 시스템]  ├──(배치: FILE, MFT)────▶ [통합 데이터 ETL]  │ └─▶ [통합 데이터 DB]  ├──(온라인: API, HTTP, 대외MCA)──▶ [통합 데이터 AP]  │ └─▶ [통합 데이터 DB]  └──(동의 철회 정보)  ├─▶ [은행 대외 MCA] ─▶ [지주 통합 데이터 AP] ─▶ [마케팅 시스템]  └─▶ [은행 대외 MCA] (MFT) ─▶ [지주 통합 데이터 AP] ─▶ [마케팅 시스템]  [빅스퀘어] ──▶ Kafka Stream, JDBC, MFT ─▶ [ETL] / [통합 데이터 DB] / [마케팅 허브]  [통합 데이터 포털] ──▶ JDBC / API ─▶ [통합 데이터 DB], [BI 포털 Proxy WEB], [마케팅 HUB] |

### ****서버별 연계 기술 구성도****

|  |  |  |  |
| --- | --- | --- | --- |
| 서버명 | 연계 방식 | 사용 기술 | 설명 |
| 통합 데이터 ETL | 배치 | MFT, FOS, JDBC, DataStage | 계열사 파일 수신, 전처리 및 적재 |
| 통합 데이터 AP | 온라인 | REST API, JWT, OAuth2, SpringBoot | 실시간 API 처리 및 인증 |
| 통합 데이터 DB | 직접접근 | JDBC, Oracle | 데이터 저장소 |
| 통합 데이터 포털 | 온라인 | React, SpringBoot, JDBC | 사용자 포털 UI |
| BI 포털 Proxy WEB | 중계 | REST API, HTTP | 포털 API 프록시 |
| 마케팅 시스템/허브 | 온라인 + 배치 | Kafka, REST API, MFT | 이벤트 소비 및 결과 수신 |

### ****연계별 인터페이스 명세 구조****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | 연계 방식 | 경로 | 포맷 | 인증방식 | 응답시간 SLA |
| API 연계 | HTTP | 대외MCA → 통합 데이터 AP | JSON/XML | JWT + OAuth2.0 | 3초 이내 |
| 배치 연계 | FILE/MFT | 계열사 → ETL | CSV / TXT | FOS 인증 | 매일 n시 |
| 이벤트 연계 | Kafka | 빅스퀘어 → 마케팅 허브 | JSON | 토픽기반 정책 | Near real-time |
| JDBC 연계 | JDBC | 포털/OLAP → 통합 데이터 DB | SQL | 내부 망 접근제어 | 실시간 |

### 클러스터 배치 구성도 및 흐름도 (텍스트 기반)

|  |
| --- |
| [PaaS 클러스터 (Tanzu 기반)]  ├── Web Layer  │ ├── 통합 데이터 포털 (React)  │ └── BI 포털 Proxy Web  ├── API Layer  │ ├── 통합 데이터 AP (SpringBoot)  │ └── 인증 서버 (Keycloak)  ├── ETL Layer  │ └── ETL 서버 (DataStage, MFT, Control-M)  ├── DB Layer  │ ├── 통합 데이터 DB (Oracle HEX)  │ └── 포털 DB (MySQL)  [통신 흐름]  사용자 → 포털(Web) → API Gateway → 통합 데이터 AP → DB  계열사 → MFT → ETL → DB  빅스퀘어 → Kafka → 마케팅 시스템 |

### 배치 관점 전체 마케팅 정보 흐름도

|  |
| --- |
| [1] 계열사 (투자증권 / 생명 / 손보 / 은행)  └─ 배치 전송 (FILE, CSV 등) ──▶ [대외 MCA]  └─ MFT 전송 (FOS) ──▶ [통합 데이터 ETL]  [2] 통합 데이터 ETL (DataStage)  ├─ 수신 파일 정제 및 변환  ├─ 고객동의 정보, 거래 정보, 상품 정보 등 전처리  └─ JDBC 적재 ──▶ [통합 데이터 DB (Oracle HEX)]  [3] 통합 데이터 DB  └─ 대상 조건 필터링 + 캠페인 규칙 매핑  (통합고객번호, 고객반응이력, 프로파일 기반)  [4] 통합 데이터 ETL  └─ 타깃 고객 리스트 생성 (Push 대상 추출)  ├─ 배치 전송 (MFT) ──▶ [마케팅 시스템 / 마케팅 허브 / e-마케팅]  └─ JSON 파일 / CSV 기반 전달  [5] 마케팅 시스템  ├─ 추출된 타깃 고객정보 기반 캠페인 실행  ├─ 실시간 또는 예약 푸시, 배너, 알림 설정  └─ REST API 호출 ──▶ [올원뱅크, 포털, UMS 등 채널 시스템]  [6] 채널 사용자 (모바일 뱅킹, 통합단말 등)  └─ 올원뱅크 / 인터넷뱅킹 앱 접속 시  └─ 실시간 마케팅 결과 수신 (API 또는 알림) |

#### 🧩 ****배치 기반 마케팅 처리 흐름 요약****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 단계 | 처리 주체 | 처리 방식 | 주요 기술/도구 | 설명 |
| ① | 계열사 | 배치파일 전송 | MFT, FOS | 고객 동의 기반 거래·상품·이력 파일 생성 후 전송 |
| ② | 통합 데이터 ETL | 파일 수신·정제 | DataStage, Control-M | 메타 매핑, 표준 필드 변환, 오류 처리 포함 |
| ③ | 통합 DB | 데이터 적재 | Oracle HEX | 통합고객번호 기준으로 마케팅 대상 후보 추출 가능 |
| ④ | ETL + DB | 마케팅 대상 생성 | SQL 필터링 + CSV | 캠페인 조건 기반 추출, 동의여부 및 채널 타깃 포함 |
| ⑤ | 마케팅 시스템 | 대상 정보 소비 | REST API, JSON | 마케팅 메시지 준비 및 채널 전달 연계 |
| ⑥ | 채널 시스템 | 사용자 노출 | 올원뱅크 앱, 웹 UI | 푸시 메시지, 배너, 팝업 등의 형태로 노출됨 |

#### 추가 고려사항 (실제 설계에 적용될 사항)

* 🔐 **보안 정책 적용**: MFT 구간 및 DB 저장 시 **AES-256**, **TLS 1.2** 기반 암호화 적용
* 📋 **이력 관리**: 고객동의 이력, 마케팅 응답 결과는 모두 **감사 로그 기록**
* 🔄 **실패 재처리**: ETL 구간에서 실패 건에 대한 **재시도 처리 로직** 포함
* 📡 **성과 수집**: 채널 노출 결과 (클릭/열람 등)는 다시 ETL 통해 **성과 반영 및 분석**

## ✅ 1. 인터페이스 상세 설계서 (표 형식)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | Source 시스템 | Target 시스템 | 연계 업무 및 데이터 | 연계 방식 | 연계 경로 |
| 대내 | 은행정보계 | 통합 데이터 ETL | 제3자 동의 고객정보, 거래정보, 상품정보 등 | FOS | 은행 정보계 → FOS → 통합 데이터 ETL |
| 대내 | 카드정보계 | 통합 데이터 ETL | 제4자 동의 고객정보, 거래정보, 상품정보 등 | FOS | 카드 정보계 → FOS → 통합 데이터 ETL |
| 대내 | 빅스퀘어 | 통합 데이터 ETL | 분석정보, 행태정보 등 | FOS | 빅스퀘어 → FOS → 통합 데이터 ETL |
| 대내 | 마케팅 허브 | 통합 데이터 ETL | 마케팅 성과/반응 정보 (Push) | FOS | 마케팅 허브 → FOS → 통합 데이터 ETL |
| 대내 | e-마케팅 | 통합 데이터 ETL | 마케팅 성과정보 (배너/팝업) | FOS | e-마케팅 → FOS → 통합 데이터 ETL |
| 대외 | NH투자증권 | 통합 데이터 ETL | 제3자 동의 고객/거래/상품정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 ETL |
| 대외 | NH투자증권 | 통합 데이터 AP | 제3자 동의 철회 고객정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 AP |
| 대외 | NH생명보험 | 통합 데이터 ETL | 제3자 동의 고객/거래/상품정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 ETL |
| 대외 | NH생명보험 | 통합 데이터 AP | 제3자 동의 철회 고객정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 AP |
| 대외 | NH손해보험 | 통합 데이터 ETL | 제3자 동의 고객/거래/상품정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 ETL |
| 대외 | NH손해보험 | 통합 데이터 AP | 제3자 동의 철회 고객정보 | 대외 MCA | 계열사 → 대외 MCA → 통합 데이터 AP |
| 대내 | 데이터포털 | 통합 데이터 DB | 사용자/메인/데이터 분석 및 요청 등 | JDBC | 데이터포털 → JDBC → 통합 데이터 DB |
| 대내 | 통합 데이터 ETL | 빅스퀘어 | 분석대상 데이터, 모형 개발 데이터 등 | FOS | 통합 데이터 ETL → FOS → 빅스퀘어 |
| 대내 | 통합 데이터 ETL | 마케팅 허브 | 타깃 고객 데이터 | FOS | 통합 데이터 ETL → FOS → 마케팅 허브 |
| 대내 | 통합 데이터 ETL | e-마케팅 | 타깃 고객 데이터 | FOS | 통합 데이터 ETL → FOS → e-마케팅 |

### ✅ 2. 설계 기준 요약

|  |  |
| --- | --- |
| 항목 | 기준 |
| **주요 연계 프로토콜** | FOS, 대외 MCA, JDBC |
| **데이터 포맷** | File 기반 (Batch), API 기반 (실시간), SQL 기반 (JDBC) |
| **연계 대상** | 내부 금융 계열사, 외부 계열사, 분석시스템, ETL 플랫폼 |
| **연계 대상 시스템** | ETL, AP, DB, 마케팅 허브, 빅스퀘어, 데이터포털 등 |
| **보안 적용** | 외부는 MCA 기반 IP 및 인증 제어, 내부는 FOS 보안 전송, JDBC 접근 제어 포함 |

## ✅ 온라인 연계 흐름도 (실시간 API 기반)

[1] 외부 요청자 (계열사 사용자/분석가/BI 포털 사용자)

│

▼

[2] BI 포털 Proxy WEB (HTTP API)

│

▼

[3] 통합 데이터 포털 (Spring Boot, UI + API)

│ └─ 인증: JWT / OAuth2.0

│ └─ API 요청: 사용자별 메타데이터, 데이터 요청

▼

[4] 통합 데이터 AP (Spring Boot, API Gateway 역할)

│ ├─ REST API (실시간 서비스 처리)

│ ├─ JWT 인증 검증 + OAuth2 인가 연계

│ └─ Kafka 이벤트 연동 (필요 시 비동기 처리)

▼

[5] 통합 데이터 DB (Oracle HEX 기반)

│

└─ 결과 응답: 가공된 JSON or 메타 데이터

▲

│

[6] 마케팅 시스템, e-마케팅 시스템, OLAP

└─ API Gateway 연계로 실시간 조회



### ✅ 주요 구성 요소 설명

|  |  |  |
| --- | --- | --- |
| 단계 | 구성 요소 | 주요 역할 및 처리 내용 |
| 1 | **외부 요청자** | 사용자, 계열사, BI 포털 접근 주체 |
| 2 | **Proxy WEB** | HTTP API 게이트웨이, SSL Termination |
| 3 | **통합 데이터 포털** | 사용자 요청 처리, REST 연동, 권한 검증 |
| 4 | **통합 데이터 AP** | API 라우팅 및 응답 처리, 인증/인가 적용, 데이터 가공 |
| 5 | **통합 데이터 DB** | Oracle 기반 실시간 데이터 조회 |
| 6 | **마케팅 / OLAP 시스템 등** | 분석/캠페인 요청 시 실시간 통합 데이터 AP 호출 |

### ✅ 연계 기술 요약

|  |  |  |
| --- | --- | --- |
| 구분 | 기술/프로토콜 | 비고 |
| API 인증 | JWT, OAuth2.0 | Keycloak 기반 권한 토큰 처리 |
| 요청 라우팅 | Spring Cloud GW, REST | 통합 데이터 AP 내부 API 설계 기반 |
| 응답 포맷 | JSON | API 응답을 JSON 포맷으로 통일 |
| DB 연결 | JDBC (Oracle) | 통합 데이터 DB 직접 조회 |
| 보안 | HTTPS, TLS 1.2+ | 외부 노출 API는 SSL 종료 지점 보호 |

## ✅ 배치 기술 흐름도 (데이터 수집 → ETL → 적재 → 연계)

markdown

복사편집

[1] Source 시스템 (은행정보계 / 카드정보계 / 계열사 등)

└─ 파일 생성 (CSV, TXT 등)

└─ 파일 전송 (MFT or FOS)

▼

[2] 수신 인터페이스 계층

├─ FOS or 대외 MCA 연계

└─ 파일 저장소 (NAS / FTP)

▼

[3] ETL 서버 (IBM DataStage)

├─ 수집된 원천 파일 읽기

├─ 데이터 정제/검증 (스키마 검증, 포맷 정규화, Null 처리 등)

├─ 이력 로깅 (적재 전/후 로그 저장)

└─ 변환 후 정제 마트 or 통합 데이터 DB로 적재

▼

[4] 통합 데이터 DB (Oracle HEX)

└─ 데이터 저장 (원천/정제/마스터 테이블)

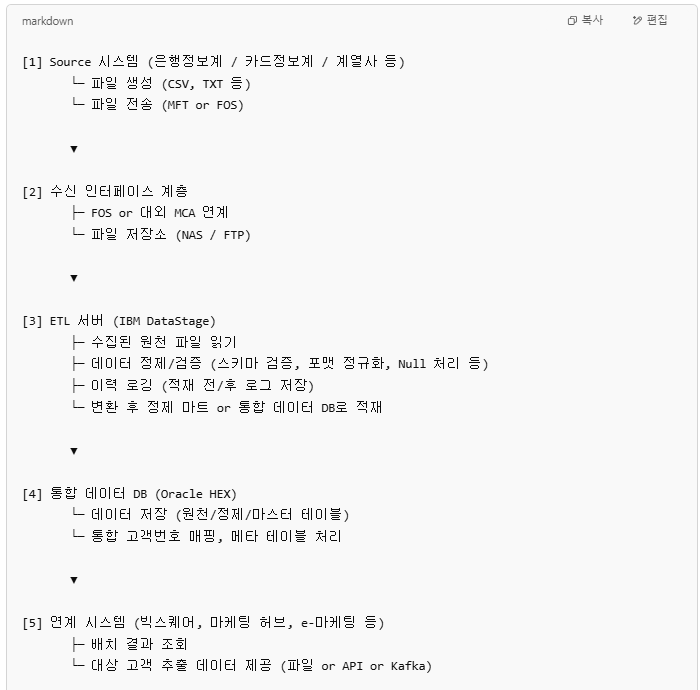
└─ 통합 고객번호 매핑, 메타 테이블 처리

▼

[5] 연계 시스템 (빅스퀘어, 마케팅 허브, e-마케팅 등)

├─ 배치 결과 조회

└─ 대상 고객 추출 데이터 제공 (파일 or API or Kafka)



### ✅ 구성 기술 및 스택 요약

|  |  |  |
| --- | --- | --- |
| 구성 단계 | 기술 스택 / 방식 | 설명 |
| 파일 수집 | MFT, FOS, FTP | 계열사 및 내부 파일 정기 수집 |
| 연계처리 | 대외 MCA, NAS | 분리된 연계 통신망 통한 수신 |
| ETL 처리 | IBM DataStage, Control-M | 수집-정제-로깅-적재 수행 |
| DB 적재 | Oracle HEX, JDBC | 통합 고객 마스터 및 데이터 저장소 |
| 로깅/오류처리 | Error Table, Logging DB, Slack Alert | 실패 시 로그 및 모니터링 시스템 연계 |
| 데이터 제공 | Kafka, File(MFT), REST API | 분석시스템/마케팅으로 배치 결과 전달 |

### ✅ 정기 배치 스케줄 예시

|  |  |  |  |
| --- | --- | --- | --- |
| 주기 | 대상 | 방식 | 비고 |
| 매일 자정 | 고객/거래/상품 데이터 | MFT + ETL | 은행/카드/보험 포함 |
| 매 1시간 | 마케팅 반응정보 | FOS + ETL | 마케팅 허브 연계 |
| 매주 월요일 | 고객 프로파일링 추출 | ETL | 빅스퀘어 전달 |

### 📌 주요 전략적 고려사항

* **이력 관리 체계 수립**: 수집 원본, 오류 로그, 적재 이력은 모두 추적 가능해야 함
* **배치 실패 시 재처리 전략 포함**: Retry, Alert, Manual Intervention 설계
* **Secure Transfer 설계**: MFT 전송 구간은 암호화 및 로그 감사 기능 필수
* **시간대별 적재 통합관리**: Control-M 기반 적재 순서 및 모니터링 체계 필요

## ✅ 1. 인터페이스 명세서 구조 (REST API / 배치 공통)

|  |  |
| --- | --- |
| 구분 | 내용 예시 |
| 인터페이스 ID | IF001\_REST\_CUSTOMER\_SEARCH |
| 인터페이스 명 | 고객정보 조회 |
| 연계 방식 | REST API / 배치 (FILE / MFT / JDBC) |
| 송신 시스템 | 통합데이터 AP / 통합데이터 ETL |
| 수신 시스템 | 마케팅 시스템 / 빅스퀘어 등 |
| 호출 방식 | HTTP POST / GET / PUT (REST) / FILE Polling (배치) |
| 인증 방식 | OAuth2.0 + JWT / FTP 인증 (배치) |
| 포맷 | JSON (REST) / CSV, TXT (배치) |
| 전문 샘플 | { "customerId": "1000001", "birth": "900101" } |
| 응답 샘플 | { "name": "홍길동", "grade": "Gold" } |
| 오류 코드 및 메시지 | 401, 500, Custom Code 등 |
| 전송 주기 | 실시간 / 매일 00:10 / 매시 10분 |
| 인터페이스 흐름도 | 하단 흐름도 참조 |

## ✅ 2. REST API 연계 흐름도 (OAuth2.0 + JWT)

scss

복사편집

[사용자/시스템] ──▶ [API Gateway (Zuul/Kong)]

└─ OAuth2.0 인증 요청

│

▼

[인증 서버 (Keycloak)]

└─ Access Token 반환 (JWT)

│

▼

[API Gateway] ── Authorization Header 설정 ─▶ [통합데이터 AP (SpringBoot)]

└─ DB 조회 (JDBC)

└─ 응답 JSON 반환

▲

응답 결과 반환 (HTTP 200, JSON)

🔐 **인증 방식**: OAuth2.0 기반 인증 + JWT 토큰  
🔁 **토큰 갱신**: Refresh Token 사용하여 재발급 가능  
🧩 **API 정책**: Rate Limiting, IP Whitelist, HMAC 보안 적용 가능



## ✅ 3. 배치 전송 흐름 상세도 (FOS/MFT 중심)

markdown

복사편집

[계열사 원천 시스템]

└─ 데이터 파일 생성 (CSV / TXT)

└─ 전송 (MFT 또는 FOS)

▼

[통합데이터 ETL 서버]

├─ 수신 디렉터리 Polling

├─ 데이터 파싱 및 정제 (DataStage)

├─ 오류/이력 로깅

└─ 적재 (Oracle DB)

▼

[데이터 소비 시스템]

├─ 타겟 마케팅 시스템으로 파일 전송

├─ 또는 Kafka 전송 (필요 시)

📌 **보안**: 전송 시 SFTP 또는 FOS 보안 채널 사용  
🕓 **타이밍**: Control-M 등으로 시간 예약 배치 스케줄 수행  
🔄 **재처리**: 오류 발생 시 에러 로그 기반 수동 or 자동 재처리



## ✅ 4. 계층별 연계 구성도 (AP ↔ ETL ↔ DB ↔ UI)

markdown

복사편집

[1] Channel Layer (UI)

└─ 통합데이터 포털 UI, BI 포털 Proxy WEB

└─ HTTP 요청 (REST)

▼

[2] Interface Layer

├─ API Gateway (Kong/Zuul)

└─ 인증 모듈 (Keycloak, JWT, OAuth2.0)

▼

[3] Application Layer (통합데이터 AP)

├─ Spring Boot API 서비스

└─ REST 요청 로직, Kafka 발행기 (비동기)

▼

[4] Data Processing Layer

├─ ETL: IBM DataStage

└─ 제어 스케줄러: Control-M

▼

[5] Data Storage Layer

├─ Oracle HEX DB (정제/원천/분석 데이터 저장)

└─ Kafka / Redis (캐시, 이벤트 처리)

▼

[6] External Consumers

└─ e-마케팅, 마케팅허브, 빅스퀘어, OLAP

🧩 **연계 표준화**: JSON + REST / Kafka + Avro / File + Metadata  
🔐 **보안 처리**: 계층별 인증 및 암호화 (JWT, TLS, DB 암호화)  
🗂️ **모듈화 설계**: 각 계층 독립적 재배포 및 확장 가능



## ****서버 구성 정보****

### ****서버 구성별 기술 스택 정보 표****

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **구분** | **서버명** | **주요 역할** | **기술 스택 (Application)** | **운영체제 / 미들웨어** | **보안 솔루션** | **비고** |
| 1 | 통합 데이터 AP | API 처리, 인증, 외부 연계 | Spring Boot, REST API, JWT, OAuth2, JPA | Linux (RHEL), Tomcat (Container), Tanzu | TOS, Deep Security | 실시간 분석 요청 처리 및 중계 허브 |
| 2 | 통합 데이터 DB | 통합 데이터 저장소 | Oracle DB, PL/SQL, Partitioning, Index Tuning | Linux (RHEL) | DB-Safer, Audit Log 솔루션 | 대용량 트랜잭션/분석 데이터 저장 |
| 3 | 통합 데이터 ETL | 데이터 정제 및 적재 | IBM DataStage, JDBC, Shell Script, Control-M Client | Linux (RHEL) | 접근제어, MFT 연계 보안 | 원천 시스템 → 통합 DB 연계 |
| 4 | 통합 데이터 포털 | 사용자 데이터 조회 및 관리 UI | Spring Boot, React, REST API, OAuth2 | Tomcat (xFrame5 기반), Tanzu | TOS, Deep Security | 사용자 권한/품질 리포팅 기능 포함 |
| 5 | BI 포털 Proxy Web | 사용자 요청 프록시 | Spring Boot, Web Front, API Gateway 역할 | Linux, Nginx | TLS/SSL 인증, API Key, Rate Limiting | 통합 데이터 포털 중계 |
| 6 | OLAP | 분석 데이터 제공 | MSTR, Tableau, OLAP Cube, JDBC | Windows/Linux 혼합 | 사용자 접근 제어, 인증 연계 | 메타 기반 정형 분석 지원 |
| 7 | 마케팅 시스템 | 캠페인 관리, 개인화 추천 | Kafka, Spring Boot, API 연계, ELT | Kubernetes, Docker | JWT 인증, 로그 추적 | Push, 배너, 타겟 연계 처리 |
| 8 | e-마케팅 | 반응/성과 기반 마케팅 | Kafka, API Gateway, Event Trigger, JSON | Docker, RESTFul Server | API 인증, 전송 암호화 | 고객 행태 기반 분석 대응 |
| 9 | 마케팅 HUB | 마케팅 이벤트 집계 | Spring API, Kafka Consumer, MFT | Linux, Apache | TLS, ACL 보안정책 | 통합 전송 채널 연계 |
| 10 | BigSquare | AI/ML 분석 시스템 | Python, Spark, Kafka, ML Pipeline | Linux, Hadoop Ecosystem | 보안 컨테이너, ACL | 예측 모델링 결과 제공 |
| 11 | 공통 연계 서버 | 외부 시스템 연계용 중계 | API Gateway (Kong/ZUUL), Kafka, MFT | Tanzu, Envoy Proxy | L4, WAF, TLS, OAuth | EAI, FEP, 외부기관 연계 |

### 🔍 ****특이사항 및 전략적 포인트****

* **컨테이너 기반 서비스**는 Tanzu를 중심으로 **Spring Boot + Tomcat** 또는 **Nginx** 환경에서 운영.
* **보안 요소**는 **TLS, OAuth2, JWT, API Key, ACL, WAF** 등 다계층 방어 설계.
* **배치/ETL 구성**은 **DataStage + Control-M + MFT** 연계로 자동화 및 오류 추적 강화.
* **분석 계층**은 **Kafka → OLAP/Portal → 시각화/결과 전송** 구조로 이벤트 기반 흐름 설계.



## Kafka 기반의 비동기 이벤트 흐름 구조

### 📌 ****이벤트 연계 상세 흐름도 (텍스트 기반)****

less

복사편집

[분석 시스템 (빅스퀘어)]

└─▶ Kafka Producer

├─ 분석 결과 이벤트 발행 (예: 고객 행동 분석 결과, 캠페인 추천 등)

├─ 이벤트 Topic: bigsquare.analysis.result

└─ 메타정보 포함 JSON 메시지 전송

↓

[Kafka Cluster (중앙 브로커)]

├─ Topic: bigsquare.analysis.result

├─ Topic: marketing.campaign.request

├─ Topic: marketing.feedback.response

└─ Event Log: Topic별 분산 로그 저장 및 병렬 처리

↓

[통합 데이터 AP (SpringBoot API)]

└─ Kafka Consumer 등록

├─ 분석 이벤트 수신 → DB 저장 (통합 데이터 DB)

├─ 캠페인 대상 고객 추출 → 마케팅 시스템 연계

├─ 응답 이벤트 적재 → e-마케팅 시스템 전달

└─ 이벤트 모니터링 로그 기록

↓

[마케팅 시스템 / e-마케팅 시스템]

└─ Kafka Consumer로 이벤트 수신

├─ 고객 개인화 메시지 생성

├─ 실시간 캠페인 생성 및 등록

├─ 푸시/배너 응답 이벤트 생성

└─ 이벤트 → feedback.response Topic 발행

↓

[통합 데이터 ETL]

└─ Kafka Topic 수신

├─ 마케팅 응답 정보 수신 (배치 적재용)

└─ 정제 후 통합 데이터 DB 적재

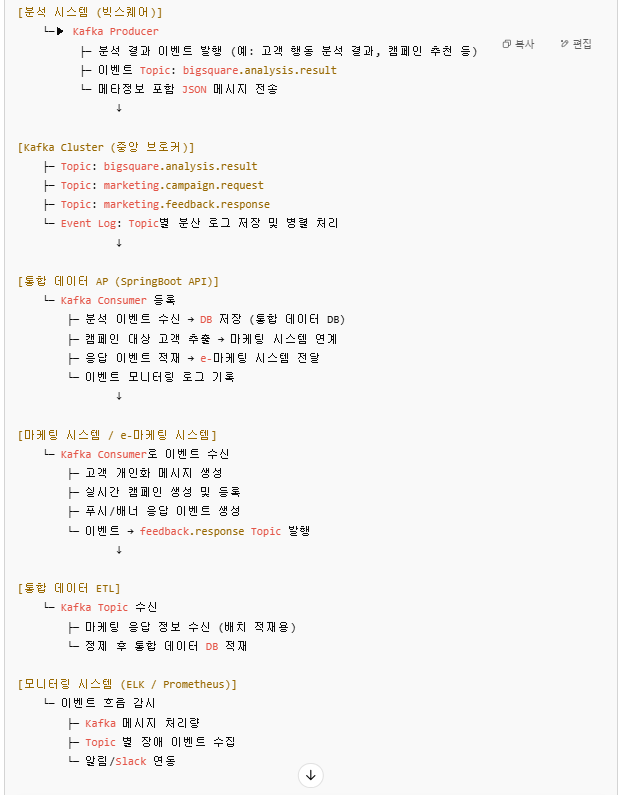
[모니터링 시스템 (ELK / Prometheus)]

└─ 이벤트 흐름 감시

├─ Kafka 메시지 처리량

├─ Topic 별 장애 이벤트 수집

└─ 알림/Slack 연동



### ✅ ****이벤트 연계 흐름 특징 요약****

|  |  |
| --- | --- |
| 구분 | 상세 설명 |
| **Event Producer** | BigSquare, 통합 데이터 AP, e-마케팅 시스템 |
| **Event Broker** | Kafka Cluster (분산형 메시지 처리) |
| **Consumer** | 통합 데이터 AP, 마케팅 시스템, ETL 등 |
| **이벤트 포맷** | JSON, 메타 포함 (이벤트 ID, 발생 시간, 출처, Payload 등) |
| **주요 연계 Topic** | bigsquare.analysis.result, marketing.campaign.request, marketing.feedback.response |
| **보안 및 인증** | Kafka TLS 적용, 이벤트 발신자 인증 (ACL), JWT 기반 이벤트 인증 |
| **모니터링** | Grafana + Prometheus로 이벤트 처리 상태 시각화 및 장애 알림 |

### 🧩 ****추가 설계 고려 요소****

* 이벤트 스키마 관리는 **Schema Registry** 기반으로 표준화
* **Retry Mechanism / Dead Letter Queue(DLQ)** 적용으로 장애 시 유실 방지
* **Event Logging & Audit Trail**로 전사 추적 가능성 확보
* 이벤트 ID 기반으로 **분산 트랜잭션 추적** 구조 구성

## ****대외 MCA – 통합 데이터 AP 호출흐름****

### 📌 ****1. 연계 흐름 개요****

css

복사편집

[계열사 시스템]

└─ HTTP API (REST, JSON)

└─▶ [대외 MCA (API Gateway)]

└─ 인증/라우팅 처리

└─▶ [지주 통합 데이터 AP (SpringBoot)]

├─ JWT 인증 검증

├─ 요청 파라미터 유효성 검증

├─ 통합 데이터 DB 조회 or 가공 처리

└─ JSON 응답 반환

### 📌 ****2. 기술 흐름 상세 설명****

|  |  |  |  |
| --- | --- | --- | --- |
| 단계 | 주체 | 연계 방식 | 주요 처리 내용 |
| 1단계 | 계열사 시스템 | HTTP/REST API | - 대외 MCA 엔드포인트로 실시간 요청 전송 - 요청 데이터: 고객 동의 철회 정보, 마케팅 수신 동의 정보 등 |
| 2단계 | 대외 MCA | API Gateway (인증/라우팅) | - JWT 또는 OAuth2 인증 처리 - 계열사 식별 및 요청 허용 정책 검증 - /api/consent/revoke 등 특정 경로로 통합 데이터 AP에 프록시 라우팅 |
| 3단계 | 통합 데이터 AP | Spring Boot REST API | - Keycloak 인증 토큰 해석 - 입력값 정합성/필수값 체크 - DB 조회 or Kafka 메시지 발행 등 비즈니스 로직 수행 - 처리 결과 JSON 형태 응답 |
| 4단계 | 대외 MCA | API Gateway | - 응답 결과 수신 후 계열사 시스템에 전송 - 예외 발생 시 표준 에러 코드 및 메시지 구성 포함 |
| 5단계 | 계열사 시스템 | HTTP/JSON 수신 | - 통합 데이터 AP 응답 수신 및 처리 - 결과에 따라 UI 반영, 추가 작업 수행 |

### 📌 ****3. 설계 요소별 보안 및 정책 고려사항****

|  |  |
| --- | --- |
| 항목 | 고려 사항 |
| 인증/인가 | OAuth2.0 + JWT 토큰 기반 인증 연계 (Keycloak 연동) |
| API 보안 | Rate Limiting, IP 화이트리스트, ACL 설정, 경로 기반 보안 정책 적용 |
| 통신 보안 | TLS 1.2 이상 암호화 적용 |
| 요청 로그 | API Gateway에서 Request/Response 전체 트랜잭션 로그 추적 (Fluentd 연계) |
| 장애 대응 | Timeout/Retry, Circuit Breaker 적용 가능, DLQ 처리 전략 수립 필요 |

### 📌 ****4. 예시: 호출 URI 및 응답 구조****

**[요청 URI 예시]**

makefile

복사편집

POST /api/consent/revoke HTTP/1.1

Host: api.nonghyup.com

Authorization: Bearer eyJhbGciOi...

Content-Type: application/json

{

"ci": "AEXX...ZZ12",

"revoke\_type": "MARKETING",

"timestamp": "2025-05-14T10:05:00Z"

}

**[응답 예시]**

json

복사편집

{

"status": "SUCCESS",

"revokeId": "RVK202505140001",

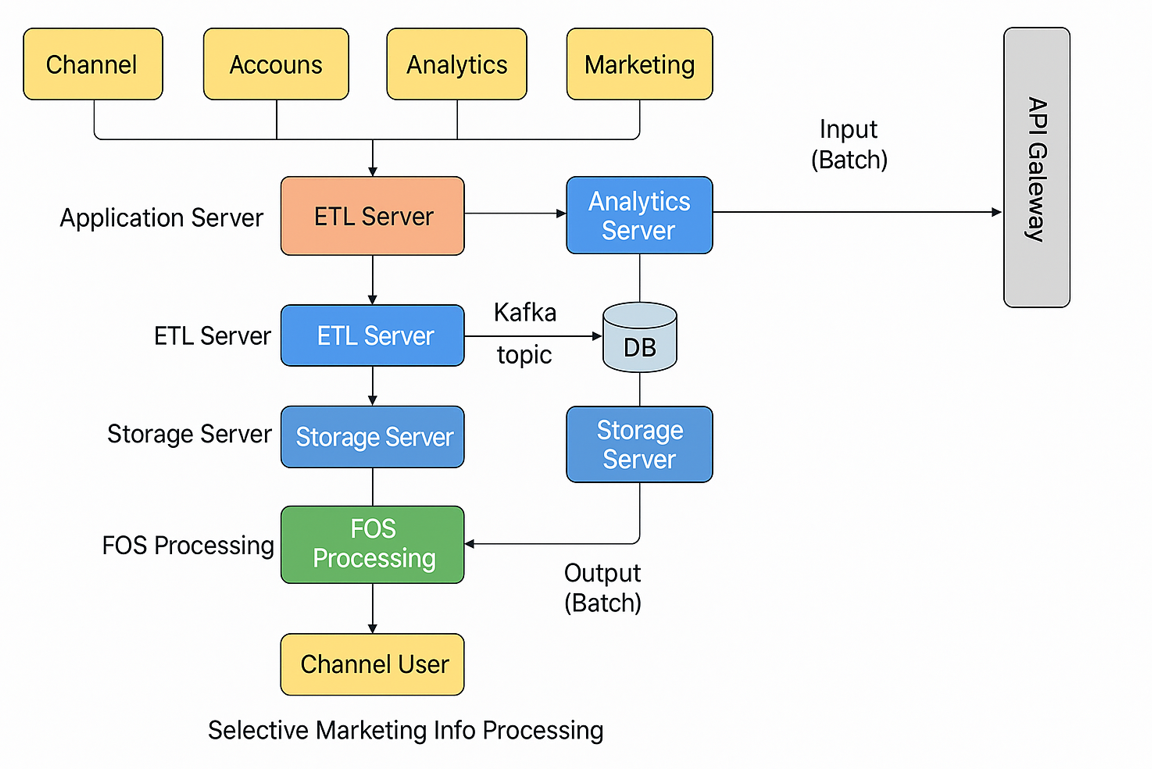
"message": "동의 철회 요청이 정상 처리되었습니다."

}

### ✅ ****전략적 의미****

* **계열사 호출 요청을 중앙 API 계층에서 통합 관리**함으로써 보안, 인증, 라우팅 정책의 일관성 확보
* **마이크로서비스 구조에서의 독립적 처리**가 가능하여 장애 격리 및 성능 분산 효과
* **거버넌스 및 감사 추적(Audit) 체계 수립이 용이**

## ****그림****



## ****마무리말****

통합 설계 및 연계 전략은 다양한 시스템 간의 데이터 교환을 표준화하고 최적화하는 중요한 역할을 합니다.  
API Gateway를 통해 외부와의 트래픽을 효율적으로 관리하고, Message Queue를 활용하여 비동기 데이터 처리를 최적화합니다.  
또한, ESB를 통해 서로 다른 시스템 간의 데이터 변환과 통합을 매끄럽게 진행하며, Batch Integration은 대용량 데이터 처리를 안정적으로 수행합니다.  
이러한 설계 전략을 통해 금융 거래의 신속성과 안전성이 보장되며, 시스템 간의 유연한 연계가 가능해집니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **API Gateway** | 마이크로서비스 간의 트래픽 관리와 부하 분산이 용이하며, 보안 정책 설정이 간편하다. |
| **Message Queue** | 비동기 메시징을 통해 트랜잭션 처리 속도를 최적화하며, 데이터 유실을 방지한다. |
| **Enterprise Service Bus (ESB)** | 이기종 시스템 간의 통합을 중앙에서 관리하여, 데이터 변환과 통신을 단일화할 수 있다. |
| **Batch Integration** | 대규모 데이터 처리 시 안정성을 확보하며, 시간 기반 트랜잭션 처리가 용이하다. |
| **데이터 연계 표준화** | RESTful API, gRPC, GraphQL을 통해 통신 표준화가 이루어지며, 확장성과 유지보수가 개선된다. |

# ****장애 대응 및 복구 전략****

## ****도입전안내말****

장애 대응 및 복구 전략은 EA(Enterprise Architecture) 체계에서 서비스의 연속성과 복원력을 보장하기 위한 핵심 요소입니다.  
특히 금융권 시스템은 무중단 운영, 거래 무결성, 실시간 복구가 필수 요건이므로, 장애 발생 시 빠르게 복구할 수 있는 구조와 프로세스를 사전에 설계해야 합니다.  
이 항목에서는 고가용성(HA), 장애 감지, 자동 복구, 재해복구(DR), 데이터 보호 및 백업 전략을 통합적으로 기술합니다.

## ****장애대응 및 복구 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **핵심 기술 스택** | **전략적 의미** |
| **11.1 고가용성(HA)** | 11.1.1 이중화(Active-Active/Passive) | 주요 컴포넌트 및 서버를 이중화하여 장애 발생 시 즉시 전환 가능 | Keepalived, HAProxy, AWS ALB | 장애 발생 시 무중단 서비스를 보장하며, 전환 지연 없이 운영 가능 |
|  | 11.1.2 Load Balancing | 트래픽을 다수 인스턴스로 분산시켜 특정 노드 과부하를 방지 | Nginx, F5, AWS ELB | 성능을 유지하면서 장애 위험을 분산시킴 |
|  | 11.1.3 Auto Scaling | 리소스 사용량에 따라 인스턴스를 자동으로 확장하거나 축소 | AWS ASG, Kubernetes HPA | 트래픽 증가 시 자동 확장으로 시스템 안정성을 유지 |
| **11.2 장애 감지 및 알림** | 11.2.1 실시간 헬스체크 | 서비스 상태를 실시간으로 감지하여 비정상 상태 발생 시 자동 조치 가능 | Prometheus, Grafana, Kubernetes Readiness/Liveness | 이상 징후를 조기에 감지하여 장애 확산을 방지함 |
|  | 11.2.2 Alert 시스템 | 장애 발생 시 개발자 및 운영자에게 실시간 알림 전송 | PagerDuty, Opsgenie, Slack Alerts | 빠른 의사결정과 대응을 통해 다운타임을 최소화함 |
| **11.3 자동 복구 및 Self-healing** | 11.3.1 컨테이너 자동 재시작 | 장애 발생 시 해당 컨테이너를 자동으로 재시작하여 서비스 복구 | Kubernetes, Docker Restart Policies | 인력 개입 없이 자동 복구되어 MTTR을 단축함 |
|  | 11.3.2 Failover Routing | 장애 발생 시 대체 경로로 자동 전환하여 요청을 우회함 | Route 53, Istio, Envoy Proxy | 네트워크 단절이나 서버 장애 시에도 무중단 운영 가능 |
| **11.4 재해 복구(DR)** | 11.4.1 DR Site 운영 | 주요 시스템을 별도 물리적 위치에 복제하여 장애 시 즉시 전환 가능 | AWS Multi-AZ, On-prem DR Site | 지진, 화재 등 물리적 재해에도 대응 가능한 안전 구조 확보 |
|  | 11.4.2 데이터 복제 및 스냅샷 | 주요 데이터는 실시간으로 복제되며, 주기적인 스냅샷 백업을 통해 복구 시간 최소화 | AWS RDS Replication, Snapshot, Oracle DG | 복구 시점 단축과 데이터 손실 최소화 실현 |
| **11.5 백업 및 복원 전략** | 11.5.1 백업 주기 및 정책 | 일별/주별/월별 백업 정책 설정으로 정기적인 데이터 보호 확보 | Oracle RMAN, AWS Backup | 정기적 백업을 통해 운영 중단 없이 복구 가능 |
|  | 11.5.2 복원 자동화 시나리오 | 재해 발생 시, 자동화된 복원 프로세스를 통해 즉시 복구 가능 | Terraform, Ansible, CloudFormation | 재해 상황에서도 복구 시나리오의 자동화를 통해 시간 단축과 운영 복원력을 확보 |

### ****(실제)**** 장애대응 및 복구 전략 – 서버 구성 포함

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 적용 대상 서버 | 설명 | 핵심 기술 스택 | 전략적 의미 |
| **11.1 고가용성(HA)** | 11.1.1 이중화 (Active-Active/Passive) | 통합 데이터 AP 서버, API Gateway, Kafka Broker, DBMS | 주요 업무 서버를 이중화하여 장애 발생 시 무중단 전환 | Oracle RAC, Kafka MirrorMaker2, Keepalived, HAProxy | 단일 장애점(SPOF)을 제거하고 가용성 극대화 |
|  | 11.1.2 Load Balancing | API Gateway, SpringBoot WAS, BI 포털 WEB/WAS | 클라이언트 요청을 균등하게 분산 | Nginx, F5 LTM, Envoy Proxy | 트래픽 집중 방지 및 확장 용이 |
|  | 11.1.3 Auto Scaling | 통합 데이터 AP, BI 포털, ETL 실행 서버 | Pod 및 Node 자동 확장 설정 | K8s HPA/VPA, Cluster Autoscaler | 급변하는 요청량 대응 및 비용 효율화 |
| **11.2 장애 감지 및 알림** | 11.2.1 실시간 헬스체크 | SpringBoot AP, Kafka, DB, Gateway | 서비스/노드 상태 지속 체크 | K8s Probe, Prometheus, Grafana | 장애 조기 감지로 대응력 향상 |
|  | 11.2.2 Alert 시스템 | 모든 모듈 공통 | 이상 징후 시 Slack/메일/SMS 자동 알림 | Alertmanager, Opsgenie, Slack | 장애 감지 후 즉각 대응 가능 |
| **11.3 자동 복구** | 11.3.1 컨테이너 자동 재시작 | 통합 데이터 AP, 통합 데이터 ETL, Kafka, Prometheus | Pod CrashLoopBack 시 자동 재기동 | RestartPolicy: Always, PodDisruptionBudget | 인력 개입 없는 자가 치유 가능 |
|  | 11.3.2 Failover Routing | 외부 요청 수신 계층 (API Gateway, LB), DNS | 특정 노드 또는 AZ 장애 시 우회 | Route53, Istio, Envoy, DNS Failover | 중단 없는 서비스 제공 |
| **11.4 재해 복구(DR)** | 11.4.1 DR Site 운영 | 통합 데이터 DB, ETL 서버, 로그 저장소 | DR 센터에 이중화 배치 | Oracle DataGuard, Kafka Mirror, Velero | 재해 발생 시 신속한 전환 가능 |
|  | 11.4.2 데이터 복제 및 스냅샷 | Oracle DB, Kafka 로그, MongoDB 등 | 실시간 복제 및 Snapshot 백업 적용 | Oracle DG, AWS RDS Multi-AZ, Kasten K10 | 복구 시간 최소화, RPO 0에 근접 |
| **11.5 백업/복원 전략** | 11.5.1 백업 주기 및 정책 | 통합 데이터 DB, 로그 서버, BI 포털 | 일간/주간/월간 스케줄링 백업 구성 | Oracle RMAN, AWS Backup, Velero | 안정적 복원성과 감사 대응 |
|  | 11.5.2 복원 자동화 시나리오 | 통합 데이터 AP, ETL, 분석 시스템 | 장애 시 자동화된 복구 절차 실행 | Ansible, Terraform, CloudFormation | 복구 프로세스 신속화 및 매뉴얼 오차 방지 |

### 🔧 관련 서버군 요약

|  |  |  |
| --- | --- | --- |
| 서버 유형 | 주요 역할 | 장애대응 주요 적용 방식 |
| **통합 데이터 AP 서버** | 실시간 API 처리 | Auto-scaling, 이중화, 인증 연계 |
| **통합 데이터 ETL 서버** | 배치 수집/적재 | Job Retry, FOS 연계, 복구 자동화 |
| **Kafka Broker/Stream** | 이벤트 큐잉 | Topic Replica, MirrorMaker2, 장애 전파 차단 |
| **통합 데이터 DB (Oracle HEX)** | 데이터 저장/조회 | Oracle RAC, DG 구성, WAL 기반 복제 |
| **BI 포털 WAS/WEB** | 사용자 분석 UI | WAS 이중화, LB 분산, 무중단 배포 |
| **로그 서버 (Elasticsearch, Fluentd)** | 통합 로그 수집 | Fluentd 이중화, Elastic Stack HA |
| **Prometheus + Grafana** | 메트릭 수집 및 알림 | Alertmanager 연계, NodeExporter 분산 |
| **API Gateway (Zuul, Kong)** | 외부 연계 API 집중 처리 | LB 분산, 인증 Proxy 구성 |

### 📌 시사점

* **모든 주요 컴포넌트는 Kubernetes 기반으로 구성되어 있어 Self-healing 및 자동확장이 가능하도록 설계되어야 하며, 장애 복구는 단일 서비스 단위가 아닌 연계된 서비스 흐름 내에서 고려되어야 합니다.**
* NH 농협지주의 안정성 요구 수준을 고려하여, **DB 계층은 RAC 또는 DataGuard**, **Kafka 계층은 Replica + Mirror**, **ETL 계층은 Job Fail Recovery 및 모니터링** 전략을 병행 적용해야 합니다.
* 특히, **DR 센터를 활용한 통합 테스트 및 연 1회 이상 장애 복구 시나리오 수행**은 실제 금융기관 준수사항으로 설계되어야 합니다.

## ****마무리말****

장애 대응 및 복구 전략은 단순한 백업 정책을 넘어서 전체 시스템의 복원력(Resilience)을 설계하는 요소입니다.  
이중화, 자동 복구, 재해 복구, 실시간 감시 및 알림 시스템은 운영 안정성을 강화하고 MTTR을 최소화합니다.  
특히 금융권 시스템처럼 24/7 가용성이 필수인 경우, DR 센터 구성 및 자동 전환 구조는 필수 요소이며, 데이터 무결성과 보안 감사도 함께 고려해야 합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **고가용성(HA) 구성** | 서비스 단절 없는 운영을 위해 이중화, 자동 스케일링, 로드 밸런싱 전략이 필수적이다. |
| **실시간 감지 및 알림** | 장애를 조기에 인지하여 대응 속도를 높이며, 운영 안정성과 고객 신뢰를 확보한다. |
| **자동 복구(Self-healing)** | 시스템 복원력을 확보하여 인적 개입 없이도 자동으로 안정화 가능한 구조를 갖춘다. |
| **재해 복구(DR) 전략** | 지리적 이중화를 통해 대규모 장애 시 빠른 복구 및 비즈니스 연속성을 유지한다. |
| **백업 및 복원 자동화** | 재해 발생 시 복원 시나리오 자동화를 통해 다운타임과 인적 리스크를 최소화한다. |

# 운영관리 및 정책 수립 전략

## ****도입전안내말****

운영관리 및 정책 수립 전략은 어플리케이션 아키텍처 설계 이후, 실제 운영 단계에서 시스템의 안정성과 지속 가능성을 보장하기 위한 핵심 요소입니다.  
EA(Enterprise Architecture) 관점에서 운영 전략은 모니터링, 변경관리, 릴리즈 및 배포, 정책 표준화, 조직 역할 분담 등 다양한 영역에 걸쳐 정의되어야 하며, 이를 통해 체계적인 운영과 SLA(Service Level Agreement)를 실현할 수 있습니다.  
특히 금융 시스템은 고가용성, 보안, 규제 대응이 필수인 만큼 운영 정책의 정합성과 자동화 수준이 성공적인 운영관리의 관건이 됩니다.

## ****운영관리 및 정책 수립 전략 작성****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **운영 도구/기술** | **전략적 의미** |
| **12.1 운영 조직 구성** | 12.1.1 역할 기반 조직 모델 | 개발, 운영, 보안, 감사 등 각 기능별로 책임을 명확히 구분하고 역할을 배분 | ITSM, R&R 매뉴얼, DevSecOps 체계 | 명확한 역할과 책임 분담을 통해 운영 중 충돌을 방지하고 협업을 촉진함 |
|  | 12.1.2 DevOps 연계 | 개발과 운영이 긴밀하게 연계되는 협업 체계 구축 | Jira, Confluence, GitLab CI/CD | 개발과 운영의 통합을 통해 배포 효율성을 높이고 장애 대응 속도를 향상시킴 |
| **12.2 변경관리 전략** | 12.2.1 Change Control Process | 운영 중 코드, 구성, 인프라 변경 시 승인 및 테스트 절차 정의 | ITIL 변경관리 체계, ServiceNow, CMDB | 무계획 변경으로 인한 장애를 방지하고 안정적인 시스템 운영을 유도함 |
|  | 12.2.2 형상관리 및 릴리즈 전략 | 소스 코드 및 설정의 형상 이력을 추적하고 릴리즈 기준을 문서화 | Git, Nexus, Release Notes 관리 | 형상 추적 및 롤백 대응이 가능해지며, 감사 및 컴플라이언스 대응에 유리함 |
| **12.3 배포 및 운영정책** | 12.3.1 블루-그린/카나리 배포 전략 | 배포 시 무중단 전환이 가능하도록 트래픽을 점진적으로 신규 시스템에 전환 | Spinnaker, ArgoCD, Istio | 사용자에게 영향을 주지 않고 안전하게 신규 기능을 적용할 수 있음 |
|  | 12.3.2 운영 시간 정의 및 SLA 설정 | 운영 시간대, 가용성 기준, 응답 시간, MTTR/MTBF 등을 포함한 서비스 기준 설정 | SLA 문서, 운영 정책 수립서 | 고객 신뢰를 확보하고 운영팀의 성과 기준을 명확히 설정할 수 있음 |
|  | 12.3.3 로그 및 이력 데이터 보존 정책 | 감사, 보안, 분석 목적으로 로그 및 트랜잭션 이력을 일정 기간 이상 보관 | ELK, Splunk, Oracle Audit Vault | 규제 대응 및 보안 사고 분석 시 활용되며, 데이터 보존의 법적 요건을 충족함 |
| **12.4 정책 표준화 및 자동화** | 12.4.1 표준 운영 절차 정의(SOP) | 장애 대응, 변경 적용, 배포 등 반복되는 절차를 표준 문서로 정의 | Runbook, SOP 문서화 시스템 | 운영 표준화를 통해 신규 인력 투입 시 교육 시간을 단축하고, 오류 발생을 최소화함 |
|  | 12.4.2 자동화 기반 운영 정책 수립 | 정기 점검, 리소스 확장, 장애 대응, 스케줄 배치 등을 자동화하여 운영의 효율성을 높임 | Ansible, Terraform, Cron, CloudWatch | 인적 리소스 투입 최소화로 비용 절감과 운영 품질 향상이 가능함 |
|  | 12.4.3 감사 및 보안 운영정책 설정 | 로그 감사, 접속 통제, 권한 이력 관리 등 보안 중심의 운영 정책 적용 | IAM, AD, SIEM | 보안 사고 발생 시 원인 추적이 가능하며, 외부 감사와 법적 대응이 수월함 |

## ****마무리말****

운영관리 및 정책 수립은 단순한 운영 안정성 확보를 넘어서, 기업의 서비스 지속성과 신뢰도, 규제 대응력까지 포함하는 아키텍처 설계의 마지막 퍼즐입니다.  
역할 기반 운영 조직과 DevOps 연계를 통해 유기적인 협업 체계를 구축하고, 변경관리와 자동화를 통해 운영의 실수를 줄이며 효율성을 극대화할 수 있습니다.  
정책 표준화와 SLA 기반 운영 전략은 고객과 조직 모두에게 신뢰와 일관성을 제공하며, 금융권처럼 보안과 감사가 중요한 환경에서는 필수적인 전략이 됩니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **운영 조직 구성** | 기능 기반으로 명확한 역할을 분담하여 장애 대응과 의사결정의 속도를 높이고 조직의 책임 경계를 명확히 한다. |
| **변경관리 및 릴리즈 전략** | 무계획 변경에 따른 리스크를 최소화하고, 형상 추적을 통한 복원력과 규제 대응을 강화한다. |
| **배포 정책 및 SLA 수립** | 블루-그린 배포 등을 통해 무중단 서비스를 실현하며, SLA 기반 운영은 신뢰도 있는 서비스를 유지하는 기준이 된다. |
| **정책 표준화 및 자동화** | 표준화된 절차와 자동화된 운영 체계를 통해 운영 효율성과 품질을 동시에 확보할 수 있다. |
| **보안 및 감사 대응 정책** | 외부 감사, 법적 요구사항, 고객 보호에 효과적으로 대응 가능하며 조직의 위험을 최소화할 수 있다. |

# ****성능·확장성·보안 최적화 요건 정리****

## ****도입전안내말****

성능, 확장성, 보안은 EA(Enterprise Architecture) 아키텍처 설계의 핵심 평가 항목이며, 시스템의 성공적인 운영과 지속적인 성장 가능성을 판단하는 기준입니다.  
금융, 공공, 제조 등 고신뢰성 환경에서는 TPS, 응답시간, 부하분산, 사용자 수용량, 보안 내성 등을 기준으로 구조적/기술적 최적화를 명확히 설계해야 합니다.  
본 항목에서는 설계 시 반드시 고려되어야 할 성능, 확장성, 보안 요건들을 세부 기술 요소와 함께 표준화합니다.

## ****성능·확장성·보안 최적화 요건 작성****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 요건** | **설명** | **적용 기술/전략** | **EA 전략적 의미** |
| **성능 요건** | 13.1.1 최대 TPS(Target TPS) | 시스템이 초당 처리할 수 있는 트랜잭션 수 정의 (예: 10,000 TPS) | 성능 테스트 도구 (JMeter, Gatling), Caching | 아키텍처 설계 초기에 기준값을 설정함으로써 과부하에 대한 대비 전략을 수립 가능 |
|  | 13.1.2 응답 시간(Response Time) | 트랜잭션 처리 후 사용자 응답까지의 시간 (예: 평균 3초 이내) | DB 튜닝, Query 최적화, Async 처리가능 구조 | 사용자 경험 향상과 SLA 만족도를 위한 기본적인 성능 보증 요건 |
|  | 13.1.3 부하 분산(Load Balancing) | 시스템 부하를 분산시켜 특정 노드에 집중되지 않도록 하는 구조 | Nginx, HAProxy, Cloud Load Balancer | 고부하 환경에서도 안정적 운영을 가능하게 하며, 단일 장애점 제거 효과 |
| **확장성 요건** | 13.2.1 수평 확장성(Horizontal Scaling) | 인스턴스 또는 노드를 추가하여 처리 용량 확장 가능 | Kubernetes, Auto Scaling Group (ASG) | 시스템 성능을 비즈니스 성장에 맞춰 유연하게 확장할 수 있도록 설계함 |
|  | 13.2.2 수직 확장성(Vertical Scaling) | CPU, 메모리 등 자원 증가를 통해 단일 시스템 성능 향상 | VM 업그레이드, 클라우드 리소스 조정 | 초기 시스템 구성 간소화에 유리하며, 중단 없는 자원 증가 가능 |
|  | 13.2.3 Multi-region 구성 | 지역별 인프라 분산 배치를 통해 글로벌 서비스 대응 가능 | AWS Global Accelerator, GCP Multi-region | 장애 격리 및 서비스 지연 최소화를 통해 해외 사용자 경험을 개선 |
| **보안 요건** | 13.3.1 인증/인가(Identity Management) | 사용자 인증 및 역할 기반 접근 제어 (OAuth2, RBAC 등) | Keycloak, Spring Security, IAM | 데이터 보호 및 보안 위협 최소화를 위한 필수 설계 요건 |
|  | 13.3.2 API 보안 | API 호출에 대한 인증, 요청 제한, 데이터 암호화 등 통신 보안 적용 | JWT, HMAC, TLS/SSL, API Gateway Rate Limiting | 내부 시스템과 외부 연계 간 신뢰 보장을 위한 최소한의 보안 장치 |
|  | 13.3.3 데이터 보안 및 무결성 | 저장 및 전송 중의 민감 정보 암호화, 변경 이력 추적, 무결성 보장 | AES-256, SHA-256, Oracle TDE, Splunk | 보안 규제(GDPR, 금융보안원 가이드라인) 대응을 위한 정책 기반 설계 |
|  | 13.3.4 취약점 및 공격 대응 | OWASP Top 10, DDoS, XSS, SQL Injection 등 주요 보안 위협에 대한 방어책 수립 | WAF, IDS/IPS, 소스코드 보안 점검 (SonarQube) | 애플리케이션 레벨 보안을 통한 위협 탐지 및 대응 체계 확보 |

### 성능·확장성·보안 최적화 요건 (NH 농협지주 데이터 플랫폼 구축사업 기준)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | 세부 요건 | 설명 | 적용 기술/전략 | 관련 서버 | EA 전략적 의미 |
| 성능 요건 | 13.1.1 최대 TPS (Target TPS) | 시스템이 초당 처리할 수 있는 트랜잭션 수 설정 (예: 10,000 TPS 이상) | JMeter, Gatling, Kafka, Redis | 통합 데이터 AP 서버, 마케팅 시스템 | 성능 기준에 따라 트래픽 수용 설계 가능 |
|  | 13.1.2 응답 시간 (Response Time) | 사용자 요청에 대한 평균 응답 시간을 3초 이내로 보장 | Async API, DB 튜닝, Caching | 통합 데이터 AP 서버, 통합 데이터 DB, OLAP | 사용자 UX 향상 및 SLA 만족 |
|  | 13.1.3 부하 분산 (Load Balancing) | 트래픽을 여러 서버로 분산하여 안정성 확보 | Nginx, HAProxy, F5 L4 | API Gateway, Web Proxy, 통합 데이터 AP | 단일 장애점 제거 및 고가용성 확보 |
|  | 13.1.4 비동기 처리 최적화 | Kafka 기반의 이벤트 처리 및 메시지 큐 활용 | Kafka Stream, RabbitMQ | 마케팅 시스템, 빅스퀘어, 통합 데이터 AP | 실시간 처리 성능 극대화 |
| 확장성 요건 | 13.2.1 수평 확장성 | 서비스 단위로 인스턴스를 추가 확장 | Kubernetes, Node AutoScale | 통합 데이터 AP, ETL 서버, 포털 WAS | 서비스 무중단 확장 구조 실현 |
|  | 13.2.2 수직 확장성 | CPU, 메모리 증가로 서버 단위 성능 강화 | VM 업그레이드, 클라우드 리소스 조정 | 통합 데이터 DB, OLAP, BI 포털 | 초기 안정성과 유연성 확보 |
|  | 13.2.3 멀티 리전 구성 | 지리적 이중화를 통한 장애 격리 | AWS Multi-AZ, GSLB | DR센터 서버군, 통합 데이터 DB 백업 | 재해 복구 및 글로벌 대응 가능 |
|  | 13.2.4 클러스터 분리 확장 | 업무 목적별 클러스터 분리 | K8s Namespace, Multi-Cluster | 분석 클러스터, 운영 클러스터 | 자원 간섭 최소화 및 부하 분산 |
| 보안 요건 | 13.3.1 인증/인가 관리 | 사용자 인증, Role 기반 권한 부여 | OAuth2.0, RBAC, Keycloak | API Gateway, 통합 데이터 포털, 통합 데이터 AP | 계층별 인증 체계 구축으로 보안 강화 |
|  | 13.3.2 API 보안 | API 호출 시 토큰 인증, 요청 제한, 암호화 적용 | JWT, TLS/SSL, Rate Limiting | 통합 데이터 AP, API Gateway | 서비스 간 신뢰성 강화 및 API 보안 확보 |
|  | 13.3.3 데이터 보안 및 무결성 | 저장/전송 데이터 암호화 및 감사 | Oracle TDE, AES256, TLS | 통합 데이터 DB, 마케팅 DB, ETL 서버 | 민감 정보 보호 및 감사 대응 용이 |
|  | 13.3.4 취약점 및 공격 대응 | 주요 공격 유형 대응 체계 구현 | WAF, IDS/IPS, SonarQube | Web Proxy, API Gateway, 외부망 DMZ | 보안 침해 대응 및 개발단계 취약점 제거 |
|  | 13.3.5 로그 및 이력 추적 | 시스템 로그 이력 및 보안감사 대응 | Fluentd, ELK, Splunk | 통합 로깅 서버, 마케팅 시스템, 통합 데이터 포털 | 보안 사고 추적, 로그 이력 통합 관리 |
|  | 13.3.6 DevSecOps 보안 내재화 | 개발과 보안의 통합 프로세스 구축 | SonarQube, GitHub Action | CI/CD 서버, GitLab, ArgoCD | 배포 전 자동 점검 기반 보안 내재화 |

#### 🔧 실 적용 예시 – NH 농협지주 데이터 플랫폼 기준

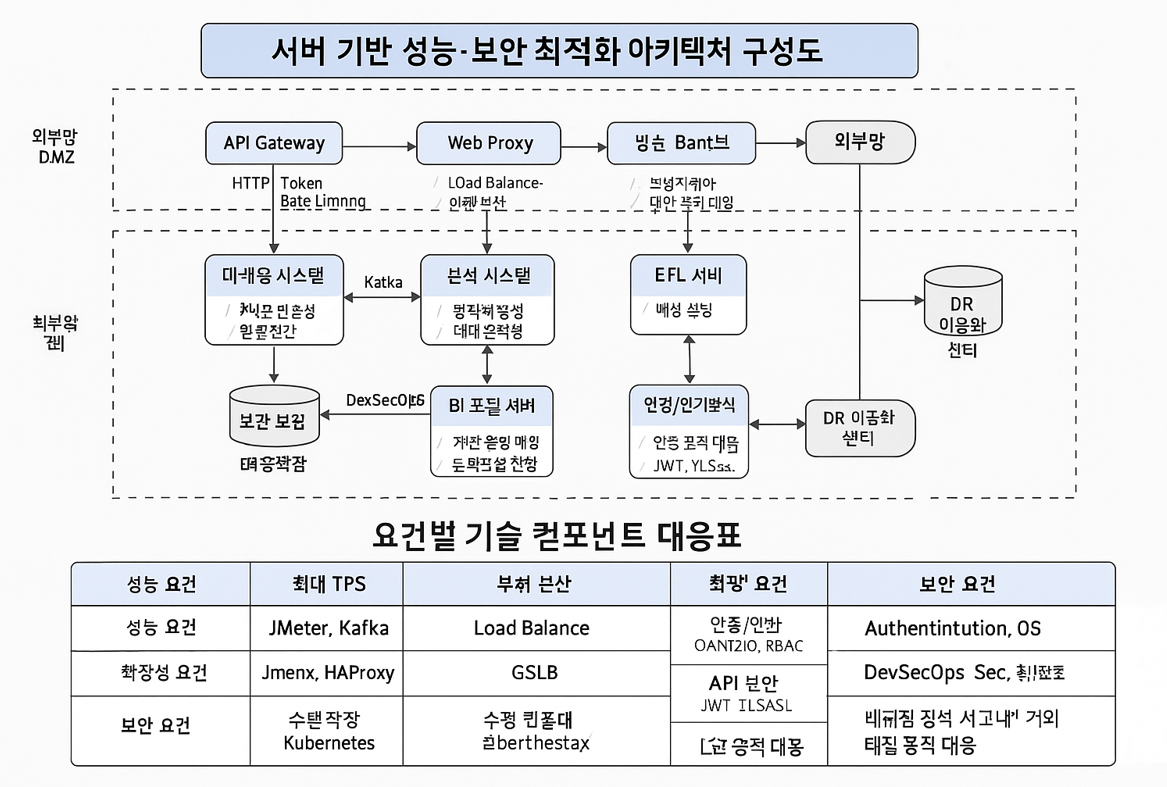
|  |  |  |
| --- | --- | --- |
| 계층 | 적용 요건 | 대표 기술/전략 |
| API Gateway | 인증, Rate Limiting, 부하 분산 | Kong, Spring Cloud Gateway, OAuth2, JWT |
| Application Layer | 수평 확장, 응답속도, 비동기 처리 | Spring Boot, Kafka, Redis, WebFlux |
| Data Layer | 데이터 암호화, 무결성, 확장성 | Oracle TDE, Redis Cluster, DB Replication |
| 분석/BI | 고속 조회 성능, 데이터 분리 접근 | OLAP, Pre-Aggregation, ElasticSearch |
| 배치/ETL | 처리량 보장, 재처리, 트랜잭션 보장 | Spring Batch, Control-M, DataStage |
| 모니터링 | 시스템 응답 추적 및 이상 감지 | Prometheus, Grafana, Fluentd, AlertManager |
| 보안망 | 방화벽, 접근 통제, 통신 암호화 | WAF, L4/L7, VPN, TLS/SSL |

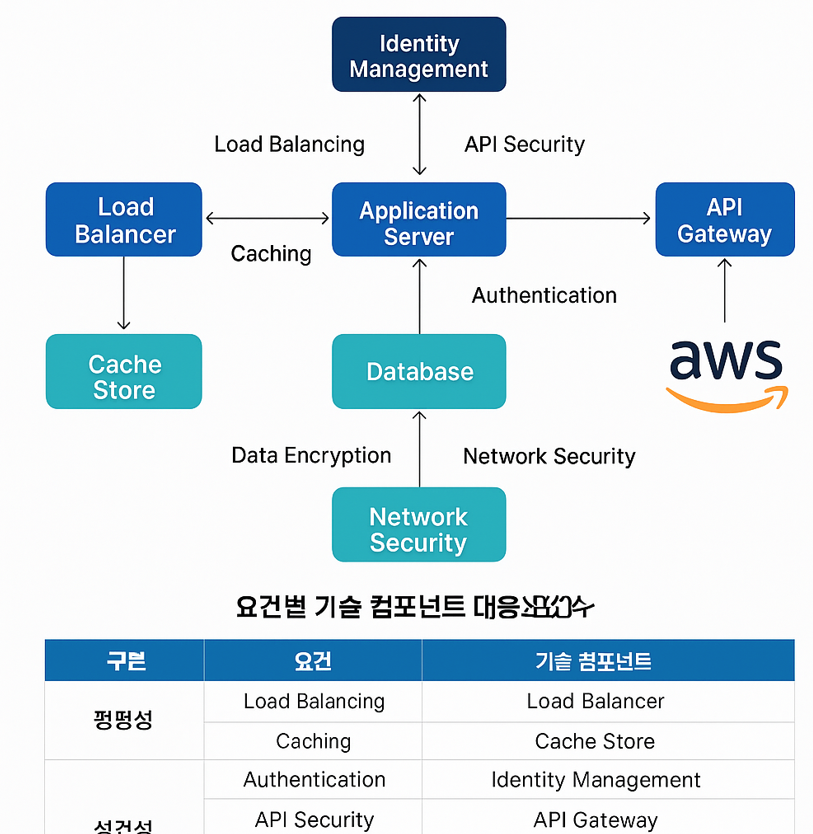
|  |  |
| --- | --- |
| 서버명 | 역할 요약 |
| 통합 데이터 AP 서버 | API 처리, 외부 연계, 인증/인가, 실시간 요청 처리 |
| 통합 데이터 DB 서버 | Oracle 기반 원천 데이터 저장, 분석 데이터 관리 |
| ETL 서버 (DataStage) | 배치 데이터 수집, 정제, 적재 처리 |
| OLAP 서버 | 분석 결과 리포팅, 통합 조회 |
| BI 포털 서버 | 사용자별 대시보드, 데이터 검색 및 관리 |
| Web Proxy / API Gateway | 외부 API 수신, 보안 연계, 인증 토큰 중계 |
| 마케팅 시스템 | 개인화 마케팅 처리, 캠페인 정보 전달 |
| 분석 시스템 (빅스퀘어 등) | ML/DL 기반 분석, Kafka 실시간 데이터 제공 |
| DR 이중화 센터 | 장애 대비 재해 복구를 위한 백업 구성 |

#### 📌 시사점

* NH 농협지주의 플랫폼은 금융업 특성상 **고신뢰/고보안/고성능이 동시에 요구되는 하이브리드 구조**입니다.
* 특히, **TPS 보장 및 SLA 응답시간 기준 설정은 초기 설계단계에서 반드시 수치로 반영되어야 하며**, 이를 기반으로 캐시, 비동기 처리, 메시지 브로커 등의 구조적 선택이 이루어져야 합니다.
* 보안 측면에서는 **API 인증·인가뿐만 아니라 로그 이력 관리, 데이터 암호화, 통신 보호를 전 계층에서 구현**하는 것이 필수입니다.

### ****구성도****





|  |
| --- |
| [사용자 브라우저/앱]  │  ▼  [L4 Load Balancer (F5, Nginx)]  │  ▼  [WAF (웹방화벽: Cloud WAF or Appliance)]  │  ▼  [API Gateway (Zuul, Kong, Spring Cloud Gateway)]  │  ├─────────▶ [Auth Server (Keycloak, OAuth2.0)]  │ └─ Token 생성 및 검증 (JWT, OIDC, MFA)  │  ▼  [Application Server Layer (Spring Boot on PaaS)]  ├─ Sessionless 처리 (Stateless 구조 / Token 기반)  ├─ 서비스 Mesh 연동 (Istio, Envoy Proxy)  └─ Async 처리 및 Event 기반 처리 (Kafka, RabbitMQ)  │  ▼  [ETL 서버 (DataStage, Control-M)]  ├─ MFT/FOS 연계 (계열사 → ETL)  └─ 정제 / 이력 로깅 / 배치 처리  │  ▼  [통합 DB (Oracle HEX + TDE 암호화)]  ├─ AES-256, TDE 적용  ├─ Oracle DataGuard 통한 DR 구성  └─ 백업 및 복원 (RMAN, Snapshot)  │  ▼  [분석 계층 (OLAP, BI, 빅스퀘어)]  ├─ OLAP: 정제 데이터 조회 (JDBC)  ├─ BI 포털 Proxy Web (PaaS)  └─ Kafka 기반 실시간 분석 연계  │  ▼  [모니터링 및 보안 계층]  ├─ 로그 수집: Fluentd → Elasticsearch (ELK Stack)  ├─ 모니터링: Prometheus + Grafana  ├─ Audit 로그: Splunk, Graylog  └─ 취약점 점검: SonarQube, Wazuh |

### ✅ ****[요건별 기술 컴포넌트 대응표]****

|  |  |  |  |
| --- | --- | --- | --- |
| 분류 | 요건 항목 | 적용 기술 | 적용 위치 (서버) |
| **성능** | 최대 TPS 보장 | Nginx, Redis Cache, Kafka | API Gateway, App Server |
|  | 응답 시간 최소화 | Async 처리, DB Query 튜닝 | App Server, Oracle DB |
|  | 부하 분산 | F5, HAProxy, Spring Cloud LB | LB, API Gateway |
| **확장성** | 수평 확장성 | Kubernetes (HPA), Auto Scaling | 모든 PaaS App Layer |
|  | 수직 확장성 | VM, Pod 리소스 조정 | App / DB / ETL 서버 |
|  | DR/멀티 리전 | Oracle DG, Cloud Multi-Zone | DB / DR 서버 |
| **보안** | 인증/인가 | OAuth2.0, JWT, RBAC | API Gateway, App Server |
|  | 전송 보안 | TLS/SSL, Mutual TLS | L4, WAF, Gateway |
|  | API 보안 | Rate Limit, HMAC, API Key | API Gateway |
|  | 데이터 무결성 | AES-256, Oracle TDE, Splunk | DB, ETL, 로그 수집기 |
|  | 취약점 대응 | WAF, IDS, SonarQube | WAF, Code Repo, Gateway |
| **가용성/복구** | Auto-Healing | Kubernetes Liveness/Readiness | 모든 PaaS App |
|  | 이중화/Failover | Active-Active, Route53 | App Server, DB |
|  | 자동 백업/복구 | AWS Backup, Ansible, RMAN | DB 서버, ETL |

### ✅ **[모듈별 흐름도 – 텍스트 구성도]**

|  |
| --- |
| 1. [사용자 요청]  └──▶ [Channel UI (Web/Mobile)]  └──▶ API 요청 (REST/HTTPS)  │  ▼  2. [API Gateway (Kong / Spring Cloud Gateway)]  ├── 인증 요청 → [인증서버 Keycloak, JWT 발급]  ├── 인증 토큰 검증 (OAuth2.0 / OpenID)  ├── API Routing 및 Rate Limiting  └──▶ 요청 라우팅 → [Business Logic 모듈]  3. [Application Server (Spring Boot)]  ├── Business Logic 처리 (Facade - Rule - Thing 구조)  ├── 비동기 이벤트 발생 → [Kafka Producer]  ├── 필요 시 DB 접근 요청  └──▶ DB 요청 → [DAO Layer → Data Layer (JPA/MyBatis)]  4. [Data Layer - 통합 DB (Oracle HEX)]  ├── 정제된 데이터 저장 (계열사 데이터 수신 후 정제)  ├── 인덱스 기반 조회 처리 최적화  └── 보안 (TDE, 암호화), 백업 (RMAN, Snapshot)  5. [Batch Processing (ETL - DataStage)]  ├── 계열사 MFT/FOS 연계 파일 수신  ├── 파일 → 파싱 및 정제 → 적재  ├── 데이터 이력 관리, 삭제, 통계화  └── 정제 후 DB 반영 + 분석 시스템 전송  6. [Event & Messaging (Kafka / RabbitMQ)]  ├── SpringBoot 서비스 → Kafka Topic 발행  ├── 소비자: e마케팅 / 마케팅 HUB / 빅스퀘어 분석  ├── 이벤트 기반 흐름 제어 (Async)  └── 실시간 마케팅 타겟팅 / 푸시 전달 등  7. [BI & 분석 서비스 (OLAP, BI 포털)]  ├── OLAP: 정제 DB 연계 분석 (JDBC)  ├── BI 포털: 사용자 보고서 UI 구성  ├── 마케팅 타겟 추출 결과 제공  └── 계열사 분석가 요청 데이터 연계  8. [로깅 및 모니터링 (ELK, Prometheus)]  ├── Fluentd → Logstash → Elasticsearch (로그 수집)  ├── Prometheus → Grafana 대시보드  ├── 실시간 알림 (Slack, PagerDuty)  └── 시스템 이상 감지 → 자동 복구 연결  9. [보안 제어 모듈]  ├── 인증: OAuth2.0, JWT, MFA (Keycloak)  ├── 인가: RBAC + ABAC  ├── 통신: SSL/TLS, Mutual TLS  ├── Web 보안: WAF + API Gateway HMAC  └── 데이터 보안: AES-256, TDE, Masking |

#### 📌 주요 흐름 예시 요약

* **온라인 처리**: Channel → API Gateway → App → DB
* **배치 흐름**: 계열사 파일 전송 → ETL → DB 적재 → 분석 전송
* **비동기 이벤트**: Kafka → e마케팅/타겟팅/알림 연동
* **분석 보고**: BI 포털/OLAP → 정제 DB 조회
* **모니터링/로깅**: Fluentd/Prometheus → 장애 감지 + 시각화

#### ✅ 1. ****모듈별 컴포넌트별 서버 매핑도****

|  |  |  |  |
| --- | --- | --- | --- |
| 모듈 | 주요 컴포넌트 | 연관 서버 구성 | 비고 |
| Channel Layer | Web UI, Mobile App | BI 포털 WEB 서버, 통합포털 UI 서버 | React, Vue, REST API |
| Interface Layer | API Gateway, 외부 API 연계 | API Gateway 서버 (Kong, Spring Gateway) | L4 + WAF 통과 |
| Business Logic Layer | Facade, Rule, Thing, DAO | 통합 데이터 AP (SpringBoot) | JWT 기반 인증처리 포함 |
| Data Layer | 통합 DB, 분석 DB | 통합 데이터 DB (Oracle HEX), BI 포털 DB | TDE, 백업/복구 구성됨 |
| ETL Layer | 수집/정제/적재, 배치 처리 | 통합 데이터 ETL 서버 (IBM DataStage) | MFT/FOS 통해 계열사 연계 |
| Analytics Layer | OLAP, BI 포털 | OLAP 서버, BI 포털 WAS, BI 포털 DB | JDBC 기반 연계 |
| Event & Messaging Layer | Kafka Producer/Consumer, Topic Processor | Kafka 브로커 서버 | 마케팅 시스템/분석 연동 |
| Logging & Monitoring | Fluentd, ELK, Prometheus, Grafana | 통합 로그 수집기 서버, 모니터링 서버 | 통합 대시보드 연계 |
| Security/Auth Layer | 인증/인가 모듈, API 인증 | Keycloak, SSO 서버, WAF, API Gateway | OAuth2, RBAC, JWT 토큰 처리 |

#### 기술스택흐름도

|  |
| --- |
| [사용자 채널] (Web, Mobile)  └─▶ [React, Vue]  └─▶ HTTP/HTTPS  ▼  [API Gateway]  └─▶ Spring Cloud Gateway / Kong  └─▶ OAuth2.0 + JWT 인증  ▼  [Spring Boot App (통합 데이터 AP)]  ├─ FACADE → RULE → THING → DAO  ├─ JPA / MyBatis → JDBC  └─▶ Oracle DB (HEX)  ▲  [ETL (DataStage)]  └─ FILE (MFT/FOS) → 정제 후 INSERT  ▼  [BI / OLAP]  └─ JDBC → 통합 DB  ▼  [Kafka] ← 이벤트 발행  └─ Consumer: 마케팅 시스템 / 분석  ▼  [Logging & Monitoring]  └─ Fluentd, Logstash → Elasticsearch  └─ Prometheus → Grafana |

#### ✅ 3. ****실시간 vs 비동기 구간 분리도****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 연계 구간 / 컴포넌트 | 연계 방식 | 기술 스택 | 목적 및 특징 |
| **실시간** | 사용자 → API Gateway | REST (HTTPS) | Spring Cloud Gateway, JWT | 인증 포함, 사용자 요청 실시간 처리 |
|  | API Gateway → 통합 데이터 AP | REST API | Spring Boot, OAuth2 | 마이크로서비스 구조 |
|  | 통합 데이터 AP → DB | JDBC | JPA, Oracle | 단건 처리 중심, 즉시 응답 |
|  | Kafka Publish / Subscribe | Kafka Stream | Kafka, Zookeeper | 실시간 분석/이벤트 기반 메시징 |
|  | BI 포털 → OLAP/DB 조회 | JDBC | BI 툴, Oracle | 실시간 분석 결과 조회 |
|  | 인증 서버 ↔ Gateway, App | Token 인증 | Keycloak, JWT | 인증/인가 실시간 검증 |
|  | API Gateway → 외부 연계 | REST API | TLS, JWT | 외부 계열사 Open API 처리 |
| **비동기** | 계열사 → MFT/FOS → ETL | 배치 전송 (FILE) | FOS, DataStage | 야간 대용량 처리 중심 |
|  | ETL → 통합 데이터 DB | JDBC | Oracle, DB Client | 정제/집계 후 저장 |
|  | ETL → 마케팅 HUB / e마케팅 (MFT) | 배치 전송 (파일) | MFT, FOS | 정해진 주기에 대용량 데이터 전송 |
|  | BI 포털 → 보고서 발행 | 스케줄 기반 | BI Tool Scheduler | 사용자 지정 시간대 리포트 출력 |
|  | 시스템 로그 → Logstash → Elasticsearch | 로그 스트리밍 | Fluentd, ELK | 비동기 로그 처리, 보안 감사 대응 |

#### ✅ 1. Kafka 메시지 흐름 구조도

[통합 데이터 AP] 또는 [ETL 모듈]

└─▶ Kafka Producer

└─▶ Kafka Topic: marketing.target.user

└─▶ Kafka Broker Cluster

├─▶ Consumer #1 (마케팅 시스템)

├─▶ Consumer #2 (e-마케팅)

└─▶ Consumer #3 (분석 플랫폼 - BigSquare)

**주요 전략**

* Producer는 비동기 이벤트 발생 시 Topic으로 발행
* Consumer는 Topic을 구독하고 이벤트에 따라 처리
* 메시지 유실 방지를 위한 ACK + Retry 설계 포함
* 메시지 key 기반 파티셔닝으로 부하 분산 적용

#### ✅ 2. JWT 기반 인증 처리 흐름

scss

복사편집

[사용자 채널 (Web, App)]

└─▶ 로그인 요청

└─▶ [Keycloak 인증 서버]

└─▶ 사용자 인증 후 JWT 토큰 발급

└─▶ 사용자 → API 요청 시 JWT 포함

[API Gateway (Spring Gateway / Kong)]

└─▶ JWT 검증

├─▶ 유효할 경우 내부 서비스 요청 전달

└─▶ 실패 시 인증 오류 반환 (HTTP 401)

[통합 데이터 AP / 마케팅 시스템]

└─▶ JWT Payload 내 사용자 권한 파악 (RBAC 처리)

└─▶ Role에 따른 데이터 접근 제어

**기술 스택**: OAuth2, Keycloak, Spring Security, JWT, Kong, OpenID Connect

#### ✅ 3. MFT 기반 배치 연계 흐름도

[계열사 시스템 (투자증권, 생명, 손해보험 등)]

└─▶ 내부 파일 생성 (고객정보, 상품정보 등)

└─▶ [대외 MCA 연계]

└─▶ [MFT 서버] (FOS 구성)

└─▶ [ETL 서버 (DataStage)]

└─▶ 통합 DB 적재 (Oracle HEX)

**주요 처리 과정**

* 배치 기반 파일 전송 (보통 새벽~이른 아침)
* FOS 수신 디렉터리 → ETL 스케줄러 감지
* 수집 → 정제 → 적재 → 로깅 및 백업 처리

#### ✅ 4. CI/CD 배포 프로세스 흐름도

[개발자]

└─▶ Git Commit (GitLab)

└─▶ CI Trigger: GitLab CI / Jenkins

├─▶ Build (Maven/Gradle)

├─▶ 테스트 자동화 수행

├─▶ Docker Image 생성

└─▶ Image → Artifact Registry 저장

[Argo CD or Spinnaker]

└─▶ GitOps 방식으로 변경 감지

└─▶ YAML 기반 배포 정의 → Kubernetes 반영

└─▶ 컨테이너 자동 배포 (무중단 Rolling Update)

[모니터링]

└─▶ Prometheus + Grafana / Slack 알림으로 배포 성공 여부 확인

**기술 스택**: GitLab, Jenkins, Argo CD, Helm, Kubernetes, Docker, Slack

#### ✅ 5. 모듈 간 클러스터 구성 및 라우팅 구조

[사용자 채널 영역]

├─▶ Web UI 클러스터 (BI 포털)

└─▶ Mobile App (React Native)

▼

[인터페이스 레이어]

└─▶ API Gateway Cluster (Spring Gateway / Kong)

└─▶ OAuth2 인증 서버(Keycloak) 연동

▼

[비즈니스 서비스 클러스터]

└─▶ 통합 데이터 AP 클러스터 (SpringBoot)

├─ FACADE

├─ RULE

└─ THING, DAO

▼

[데이터 저장소 클러스터]

├─▶ 통합 데이터 DB (Oracle HEX)

├─▶ BI 포털 DB (MySQL)

└─▶ OLAP DB 연계 (JDBC)

▼

[배치/ETL 클러스터]

└─▶ DataStage ETL 서버 (정제, 적재, 삭제 등)

▼

[분석/이벤트 클러스터]

├─▶ Kafka Broker + Topic 클러스터

└─▶ 소비자: 마케팅 시스템, e-마케팅, 분석 시스템

▼

[로깅/보안/모니터링]

├─▶ Fluentd/Logstash → ELK

├─▶ Prometheus + Grafana

└─▶ WAF, L4, ACL → API 인증 / 트래픽 제어

## ****마무리말****

성능, 확장성, 보안은 단순한 기술적 고려사항이 아닌, 비즈니스 지속 가능성과 직접 연결되는 핵심 전략 요건입니다.  
TPS 및 응답시간 기준 수립을 통해 서비스 품질을 보장하고, 유연한 확장 전략을 설계함으로써 트래픽 증가에도 무중단 서비스를 유지할 수 있습니다.  
또한, 다계층 보안 및 위협 대응 전략은 보안 사고를 미연에 방지하고, 신뢰할 수 있는 서비스 인프라를 구성합니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **요소** | **EA 관점의 전략적 의미** |
| **성능 기준 수립** | 사전 정의된 성능 기준은 설계 품질과 성능 리스크 대응 수준을 높이며, SLA 기반 품질 보장을 유도한다. |
| **확장 전략** | 수평/수직 확장과 멀티리전 구성은 클라우드 네이티브 아키텍처의 핵심이며, 미래 대응력을 강화한다. |
| **보안 설계** | API·데이터 보안, 인증/인가 체계, 공격 대응 수단은 신뢰성과 법적 규제 준수의 기준선 역할을 한다. |

# ****마이그레이션 전략****

## ****도입전안내말****

마이그레이션 전략은 기존 시스템에서 신규 EA 기반 아키텍처로의 전환 시, 다운타임 최소화, 데이터 정합성 유지, 사용자 영향 최소화를 실현하기 위한 핵심 절차입니다.  
EA(Enterprise Architecture) 관점에서 마이그레이션은 단순한 기술 이전이 아닌, 시스템 전환의 리스크를 제어하고 비즈니스 연속성을 보장하는 전략적 활동입니다.  
특히 금융, 공공 시스템과 같이 고신뢰성 환경에서는 안정성과 투명성을 기반으로 한 단계적 마이그레이션이 필수입니다.

## ****마이그레이션 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **설명** | **적용 방법/도구** | **EA 전략적 의미** |
| **14.1 대상 식별 및 분류** | 14.1.1 시스템 구성요소 분석 | 마이그레이션 대상 시스템의 범위, 기능, 연계 인터페이스 식별 | EA 도면, CMDB, System Inventory Sheet | 마이그레이션 범위를 정확히 정의함으로써 리스크를 사전 제거 가능 |
|  | 14.1.2 데이터 분류 및 정합성 검토 | 이전 대상 데이터 구조, 무결성, 종속 관계 분석 | ERD, 데이터 품질 도구(DQMS) | 데이터 무결성을 확보하며, 복원 가능성 및 감사 대응력을 확보 |
| **14.2 마이그레이션 방식** | 14.2.1 일괄 전환(Big Bang) 방식 | 정해진 시점에 전체 시스템을 새로운 구조로 일괄 전환 | 강제 커트오버, 사전 병행 테스트 | 프로젝트 기간 단축이 가능하나 리스크가 크므로 주요업무 적용은 제한 |
|  | 14.2.2 단계적 전환(Phased) 방식 | 모듈/업무별로 순차적으로 전환하여 운영 리스크를 최소화 | 점진적 배포, 업무 단위 분리 | 안정성 중심의 마이그레이션, 테스트 및 모니터링 병행이 가능 |
|  | 14.2.3 병행 운영(Parallel Run) 방식 | 일정 기간 동안 구시스템과 신시스템을 병행 운영하며 결과 비교 | 데이터 동기화 스크립트, 중간 게이트웨이 | 신뢰성을 검증하며, 고객 영향을 최소화할 수 있는 실전 대응 전략 |
| **14.3 전환 기술 요소** | 14.3.1 데이터 마이그레이션 도구 활용 | 대량 데이터를 무중단으로 추출, 정제, 적재 수행 | Talend, AWS DMS, Oracle Data Pump | 대량 데이터 이관 시 자동화 도구를 활용하여 이관 시간 단축과 오류 감소 실현 |
|  | 14.3.2 API 연계 구조 리디자인 | 기존 연계 방식을 REST, gRPC 등 표준 API 방식으로 변환 | API Gateway, Spring Cloud Gateway | 중장기적으로 유지보수와 확장이 용이한 인터페이스 구조 확보 |
|  | 14.3.3 데이터 정합성 검증 프로세스 | 마이그레이션 이후 데이터 일치 여부를 정량적으로 검증 | Hash Check, Row Count, 샘플 비교 도구 | 신뢰성 확보 및 회계/감사 대응 근거자료로 활용 가능 |
| **14.4 전환 운영 계획** | 14.4.1 롤백 및 복구 전략 | 전환 실패 시 복원 가능한 백업 정책 수립 | Snapshot, PITR, DB Export/Import | 마이그레이션 실패 대비책 확보로 리스크 제어 가능 |
|  | 14.4.2 마이그레이션 테스트 전략 | 병목/성능/보안/연계 검증을 위한 사전체크리스트 기반의 테스트 수행 | 성능테스트 도구, API 테스트, DB Query Trace | 실전 운영 시 예상 문제를 사전에 제거하고, 서비스 안정성을 높이는 핵심 단계 |
|  | 14.4.3 커뮤니케이션 및 전환 일정 관리 | 사용자, 운영자, 외부 연계기관과 사전 공지 및 전환 타임라인 공유 | Change Schedule, Release Calendar | 혼선 방지를 통해 고객 신뢰 유지 및 대외 서비스의 비즈니스 연속성 보장 |

## ****마무리말****

마이그레이션 전략은 단순 이전이 아닌, 서비스 연속성과 사용자 신뢰를 보장하는 프로젝트 전환의 핵심입니다.  
단계별/병행 방식의 전환 전략을 통해 리스크를 최소화하고, 데이터 정합성과 API 호환성을 확보함으로써 운영 안정성과 통합 효율을 극대화합니다.  
또한, 전환 전후 테스트와 복구계획을 사전에 수립함으로써 장애 발생 시에도 신속한 회복이 가능하며, 조직 전반의 마이그레이션 대응 능력을 향상시킵니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **항목** | **EA 관점의 전략적 의미** |
| **대상 시스템 분석** | 정확한 범위와 구조 파악을 통해 전환 계획의 신뢰성을 높이고, 예산 및 일정 오류를 방지한다. |
| **단계적·병행 전환 방식** | 위험 분산과 실전 검증을 통해 서비스 중단 없는 마이그레이션이 가능하며, 고객 피해를 최소화할 수 있다. |
| **데이터 무결성 확보** | 회계/법적 감사 대응 가능성과 운영 신뢰성을 동시에 확보할 수 있는 기반이 된다. |
| **자동화된 마이그레이션 도구** | 대규모 이관 작업 시 인적 오류를 최소화하고 일관된 품질을 유지할 수 있다. |
| **복구 및 커뮤니케이션 계획** | 전환 실패 시 빠른 복구를 위한 복원력 확보와 사전 공유를 통해 혼선을 줄이고 조직 간 협업을 강화한다. |

# ****종합 요약 및 리스크 대응 전략****

## ****도입전안내말****

본 항목은 지금까지 정의된 어플리케이션 아키텍처 설계 요소를 총괄 정리하고, 프로젝트 수행 및 운영 중 발생할 수 있는 리스크에 대한 사전 대응 전략을 제시합니다.  
EA 관점에서 이는 기술, 운영, 조직, 외부환경 요인에 대해 구조적 대응 체계를 사전에 수립함으로써 시스템 안정성과 사업 성공률을 극대화하는 활동입니다.  
특히 복잡도가 높은 금융/공공/대형 민간 프로젝트에서는, 리스크 대응 체계의 유무가 프로젝트 성패를 결정짓는 요인 중 하나입니다.

## ****종합 요약 및 리스크 대응 전략 작성****

|  |  |  |
| --- | --- | --- |
| **구분** | **핵심 내용 요약** | **EA 관점의 전략적 의미** |
| **15.1 아키텍처 총괄 요약** | - 계층형 구조(Channel–Interface–Business–Data)로 설계 - MSA+Monolithic 하이브리드 구조 - 성능, 보안, 확장성 기준 명확화 | 체계적 계층 분리를 통해 유지보수성과 확장성을 확보하고, 유연한 비즈니스 구조를 실현함 |
| **15.2 기술 전략 요약** | - Spring Boot, Kubernetes 기반 클라우드 네이티브 구성 - CI/CD 자동화 및 IaC 전략 도입 - 실시간 모니터링 체계 | 현대화된 기술 스택을 바탕으로 민첩한 배포와 자동화된 운영을 가능하게 하며, 장애 대응 속도를 향상시킴 |
| **15.3 데이터 전략 요약** | - GUID 기반 트랜잭션 관리 - Sharding+Replication 혼합 전략 - 암호화 및 감사대응 구조 적용 | 고성능 데이터 처리와 보안 이슈 대응을 모두 만족하는 전략적 설계 기반을 제공함 |
| **15.4 보안 전략 요약** | - OAuth2.0, RBAC 기반 인증/인가 설계 - TLS 및 API 보안 적용 - 실시간 감시 및 감사로그 정책 수립 | 금융권/공공 환경의 보안 요구사항을 충족하며, 외부 감사 및 규제에도 유연하게 대응 가능함 |
| **15.5 운영 전략 요약** | - SOP 기반 운영 절차 정립 - DevOps 연계 구조 - 자동화 기반 점검 및 복구 체계 확보 | 운영 효율성과 인력 리스크 감소를 동시에 실현하며, 운영 품질을 정량적으로 유지할 수 있음 |
| **15.6 마이그레이션 요약** | - 단계적/병행 방식 적용 - 전환 실패 대비 복구 정책 및 테스트 전략 확보 | 시스템 전환 시점의 리스크를 최소화하고 사용자 혼란과 운영 장애를 방지함 |

## ****주요 리스크 및 대응 전략****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **리스크 항목** | **발생 가능성** | **영향도** | **주요 발생 원인** | **사전 대응 전략** |
| **01. 성능 저하** | 높음 | 높음 | TPS 예측 미흡, DB 튜닝 부족 | 성능 테스트 사전 수행(JMeter), Index 설계, 캐싱 전략 적용 |
| **02. 보안 사고 발생** | 중간 | 매우 높음 | 인증/인가 미흡, 로그 미보관 | OAuth2.0, TLS, 감사 로그 정책 강화, 정기 보안 진단 도입 |
| **03. 연계 실패/중단** | 중간 | 높음 | 연계 API 버전 충돌, 동기 구조 병목 | 연계 표준화, 비동기 전환, Mock API 기반 사전 테스트 수행 |
| **04. 마이그레이션 실패** | 낮음 | 매우 높음 | 전환 계획 부재, 데이터 정합성 오류 | 병행 운영+Rollback 시나리오 수립, 전환 테스트, 데이터 검증체계 마련 |
| **05. 운영 오류 및 미숙** | 중간 | 중간 | 운영 인력 부족, 문서화 미비 | SOP 표준화, 교육 계획 수립, 자동화 운영체계(Ansible, Terraform 등) 도입 |
| **06. 장애 장기화** | 낮음 | 높음 | DR체계 미비, Auto Healing 설정 누락 | DR Site 구성, 장애감지-자동전환-복구 시나리오 완성 |
| **07. 기술 종속성 및 버전 문제** | 중간 | 중간 | OSS/상용 툴의 장기 지원 종료 또는 변경 이슈 | 기술 로드맵 수립, 대체 솔루션 준비, 기술 스택 표준화 및 모니터링 |

## ****마무리말****

EA 기반 어플리케이션 아키텍처 설계는 단순한 구조 설계가 아니라, 비즈니스 전략, 기술 실행력, 보안 대응력, 운영 안정성을 통합적으로 고려한 실행 프레임워크입니다.  
본 문서에서는 전환과 운영에 필요한 모든 기술적/관리적 요소를 정의하고, 각 단계에서의 리스크에 선제 대응할 수 있는 전략을 제시하였습니다.  
이로써 시스템의 품질, 성능, 확장성, 보안을 동시에 확보하며, 실전 프로젝트에서의 안정적 수행 기반을 확보할 수 있습니다.

## ****시사점 (표 형식)****

|  |  |
| --- | --- |
| **요소** | **EA 기반 시사점** |
| **전략적 정합성 확보** | 각 영역(계층, 보안, 연계, 운영 등)이 통합된 일관된 전략으로 정리되어 프로젝트의 통제력을 높일 수 있다. |
| **리스크 사전 대응 체계** | 기술적·운영적 리스크를 구조적으로 예측하고 대응 전략을 구체화함으로써 프로젝트 안정성과 성과를 동시에 확보할 수 있다. |
| **EA 실전 적용 가능성** | 문서 내 정의된 모든 항목은 실제 프로젝트 수행 시 실무적으로 바로 활용 가능하도록 구체화되어 있다. |

# 부록

## API Gateway

### 도입전안내말

NH 농협금융지주 통합 데이터 플랫폼에서 API Gateway는 계열사 간 데이터 통합과 서비스 접근을 위한 핵심 진입점 역할을 합니다. 이는 단순한 라우팅을 넘어 보안, 모니터링, 트래픽 관리 등 통합적인 API 관리 체계를 제공하며, 금융권의 높은 보안 요구사항과 성능 기준을 충족시키는 게이트웨이 아키텍처를 구현합니다.

### 표 기반 본문

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 설계 구현 전략 | 기술 스택 |
| Spring Cloud Gateway 도입 | 라우팅 및 필터링 | 다양한 백엔드 서비스로의 동적 라우팅 및 요청/응답 필터링 | Predicate 기반 라우팅 규칙 정의 및 필터 체인 설계 | Spring Cloud Gateway(v3.1), Spring WebFlux |
|  | 로드밸런싱 | 서비스 인스턴스 간 트래픽 분산 | 서비스 디스커버리 연동 및 로드밸런싱 알고리즘 최적화 | Spring Cloud LoadBalancer, Ribbon |
|  | 서킷 브레이커 | 장애 발생 시 빠른 폴백 및 시스템 보호 | 회로 차단 임계값 설정 및 폴백 메커니즘 구현 | Resilience4j, Spring Cloud Circuit Breaker |
| API 관리 체계 | API 버전 관리 | API 변경 시 하위 호환성 보장 | URI/Header 기반 버전 관리 전략 및 변경 관리 프로세스 | API Versioning, Semantic Versioning |
|  | 속도 제한 | 서비스별/클라이언트별 요청 제한 | 계층적 Rate Limiting 정책 및 할당량 관리 체계 | Redis Rate Limiter, Bucket4j |
|  | API 카탈로그 | API 메타데이터 및 문서화 | 자동화된 API 문서 생성 및 개발자 포털 구축 | Springdoc OpenAPI, Swagger UI |
| 보안 강화 | 인증 및 권한 관리 | 중앙화된 인증 및 세분화된 권한 제어 | OAuth2.0/OIDC 기반 인증 및 RBAC/ABAC 권한 모델 적용 | Spring Security, Keycloak(v16.1), JWT |
|  | API 키 관리 | 클라이언트별 API 키 발급 및 관리 | 키 생성, 갱신, 폐기 자동화 및 키 관리 정책 수립 | API Key Management, HashiCorp Vault |
|  | 요청 검증 | API 요청 콘텐츠 검증 및 위협 방어 | 입력 값 검증, SQL 인젝션 방지, 컨텐츠 필터링 | JSON Schema Validation, OWASP ModSecurity |
| 모니터링 및 분석 | 실시간 모니터링 | API 성능 및 가용성 모니터링 | 핵심 메트릭 수집 및 대시보드 구성 | Prometheus, Grafana, Micrometer |
|  | 로깅 및 추적 | 요청/응답 로깅 및 분산 추적 | 구조화된 로깅 및 트랜잭션 추적 체계 구축 | ELK Stack, Jaeger, Spring Cloud Sleuth |
|  | 분석 대시보드 | API 사용 패턴 및 성과 분석 | 사용량, 성능, 오류율 등 종합 분석 환경 구축 | Kibana, Grafana Dashboard, Custom Analytics |
| 확장 및 연계 | 멀티 게이트웨이 관리 | 환경별/도메인별 게이트웨이 관리 | 중앙 집중식 게이트웨이 설정 관리 및 배포 자동화 | Spring Cloud Config, GitOps |
|  | 서비스 메시 연동 | 서비스 메시 플랫폼과의 통합 | API Gateway와 서비스 메시 역할 분담 전략 | Istio Integration, Envoy Proxy |
|  | 레거시 시스템 연계 | 기존 시스템의 API Facade 구현 | 레거시 시스템 래핑 및 프로토콜 변환 | Custom Adapters, Protocol Conversion |

### 마무리말

NH 농협금융지주 통합 데이터 플랫폼의 API Gateway 구축은 계열사 간 데이터와 서비스의 안전하고 효율적인 통합을 위한 핵심 기반을 제공합니다. Spring Cloud Gateway를 중심으로 한 현대적 아키텍처는 높은 성능과 확장성을 보장하면서, 금융권에 필수적인 보안 기능을 통합적으로 제공합니다. 특히 중앙화된 인증 및 권한 관리, 실시간 모니터링, 그리고 유연한 확장성은 금융 서비스의 안정성과 민첩성을 동시에 충족시키는 요소입니다. API Gateway를 통해 내외부 서비스와 데이터에 대한 일관된 접근 체계를 구축함으로써, 농협금융지주는 계열사 간 시너지를 극대화하고 고객 중심의 통합 금융 서비스를 효과적으로 제공할 수 있을 것입니다.

## 메시지 큐 및 이벤트 스트리밍

### 도입전안내말

금융 서비스의 실시간성과 안정성을 보장하기 위해 NH 농협금융지주 통합 데이터 플랫폼은 강력한 메시지 큐 및 이벤트 스트리밍 아키텍처를 필요로 합니다. 본 섹션에서는 계열사 간 데이터 공유와 비즈니스 이벤트 처리를 위한 이벤트 기반 아키텍처의 핵심 요소와 구현 전략을 제시합니다.

### 표 기반 본문

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 설계 구현 전략 | 기술 스택 |
| Kafka 기반 이벤트 플랫폼 | 클러스터 아키텍처 | 고가용성 및 확장성 확보를 위한 클러스터 설계 | 다중 브로커 구성, 랙 인식 배치, 리전 간 복제 | Apache Kafka(v3.0), ZooKeeper, Kraft |
|  | 토픽 설계 | 효율적인 데이터 흐름을 위한 토픽 구조화 | 도메인 기반 토픽 분류, 파티션 전략, 압축 정책 | Topic Naming Convention, Partitioning Strategy |
|  | 고성능 처리 | 대용량 트랜잭션의 실시간 처리 | 프로듀서/컨슈머 최적화, 배치 처리, 병렬 처리 | Producer/Consumer Tuning, Parallel Processing |
| 이벤트 처리 패턴 | 이벤트 소싱 | 상태 변경을 이벤트 시퀀스로 저장 및 재구성 | 이벤트 저장소 설계, 이벤트 리플레이 메커니즘 | Event Store, CQRS, Snapshot |
|  | CQRS | 명령과 쿼리 책임 분리 | 쓰기/읽기 모델 분리, 비동기 이벤트 전파 | Command Services, Query Services, Event Handlers |
|  | 사가 패턴 | 분산 트랜잭션 처리 | 코레오그래피/오케스트레이션 사가 구현 | Choreography Saga, Orchestration Saga |
| 이벤트 스키마 관리 | 스키마 레지스트리 | 이벤트 스키마 중앙 관리 및 버전 제어 | 스키마 진화 전략, 호환성 검사 자동화 | Confluent Schema Registry, Avro/Protobuf |
|  | 메시지 계약 | 서비스 간 데이터 교환 계약 정의 | 계약 기반 테스트, 스키마 검증 파이프라인 | Schema Validation, Contract Testing |
|  | 데이터 형식 표준화 | 이벤트 페이로드 표준화 | JSON 스키마 정의, 필드 명명 규칙 수립 | Avro, JSON Schema, Protocol Buffers |
| 변경 데이터 캡처(CDC) | 데이터베이스 연계 | DB 변경사항의 실시간 이벤트화 | 로그 기반 CDC, 트리거 기반 CDC | Debezium(v1.8), Oracle GoldenGate, Maxwell |
|  | 초기 로드 전략 | 기존 데이터의 일괄 로드 | 스냅샷 기반 초기 로드, 점진적 마이그레이션 | Snapshot Mode, Bootstrap Mode |
|  | 변경 추적 및 복구 | 이벤트 누락 방지 및 복구 | 체크포인팅, 딜리버리 보장 전략 | Exactly-once Delivery, Dead Letter Queue |
| 이벤트 처리 파이프라인 | 스트림 처리 | 실시간 이벤트 가공 및 분석 | 윈도우 처리, 상태 기반 처리, 조인 연산 | Kafka Streams, KSQL, Flink |
|  | 이벤트 라우팅 | 조건 기반 이벤트 전달 | 필터링, 포워딩, 브랜칭 로직 | Kafka Connect, Camel K, Spring Cloud Stream |
|  | 이벤트 저장 및 분석 | 이벤트의 저장 및 분석을 위한 싱크 | 데이터 레이크 통합, 분석 DB 연계 | Kafka Connect Sinks, HDFS Sink, Elasticsearch |
| 운영 및 모니터링 | 클러스터 모니터링 | Kafka 클러스터 상태 및 성능 모니터링 | 핵심 메트릭 수집, 알림 설정, 대시보드 구성 | Confluent Control Center, Kafka Exporter, Prometheus |
|  | 메시지 추적 | 메시지 흐름 추적 및 디버깅 | 메시지 ID 기반 추적, 로깅 강화 | Distributed Tracing, Correlation ID, ELK Stack |
|  | 장애 대응 | 장애 상황 감지 및 자동 복구 | 리밸런싱 전략, 자동 복구 메커니즘 | MirrorMaker, Rebalance Strategies, Self-healing |

### 마무리말

NH 농협금융지주 통합 데이터 플랫폼의 메시지 큐 및 이벤트 스트리밍 아키텍처는 실시간 금융 데이터 처리와 계열사 간 원활한 정보 교환의 핵심 기반을 제공합니다. Apache Kafka를 중심으로 구축된 고성능 이벤트 플랫폼은 금융 트랜잭션의 신뢰성 있는 처리와 실시간 데이터 분석을 가능하게 하며, 이벤트 소싱과 CQRS 패턴의 적용은 복잡한 금융 비즈니스 프로세스의 안정적인 처리를 지원합니다. 특히 Debezium을 활용한 변경 데이터 캡처(CDC) 기술은 레거시 시스템과의 실시간 통합을 용이하게 하여, 기존 시스템의 변경을 최소화하면서도 데이터의 실시간 활용이 가능한 환경을 제공합니다. 이러한 이벤트 기반 아키텍처는 농협금융지주가 고객 행동에 실시간으로 대응하고, 데이터 기반의 선제적 금융 서비스를 제공하는 기술적 토대가 될 것입니다.

## 서비스 통합 및 오케스트레이션

### 도입전안내말

NH 농협금융지주 통합 데이터 플랫폼에서 서비스 통합 및 오케스트레이션은 다양한 서비스와 프로세스를 유기적으로 연결하여 복잡한 금융 업무 흐름을 자동화하는 핵심 요소입니다. 본 섹션에서는 마이크로서비스 간 효율적인 통신, 비즈니스 프로세스 자동화, 분산 시스템의 회복성 확보를 위한 종합적인 전략을 제시합니다.

### 표 기반 본문

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 설계 구현 전략 | 기술 스택 |
| 서비스 메시 아키텍처 | 서비스 간 통신 | 서비스 간 안정적 통신 및 트래픽 제어 | 데이터 플레인/컨트롤 플레인 분리, 사이드카 패턴 | Istio(v1.12), Envoy Proxy, Linkerd |
|  | 트래픽 관리 | 고급 라우팅 및 트래픽 제어 | 가중치 기반 라우팅, 카나리 배포, A/B 테스트 지원 | VirtualService, DestinationRule, ServiceEntry |
|  | 보안 강화 | 서비스 간 통신 보안 | mTLS 적용, 서비스 간 인증/인가 강화 | PeerAuthentication, AuthorizationPolicy, Cert-Manager |
| 비즈니스 프로세스 관리 | 워크플로우 정의 | 금융 비즈니스 프로세스 모델링 | BPMN 2.0 기반 프로세스 정의, 도메인 특화 워크플로우 | Camunda BPM(v7.16), BPMN 2.0, DMN |
|  | 프로세스 자동화 | 장기 실행 프로세스 자동화 | 상태 기반 워크플로우, 사용자 작업 관리 | Task Management, Process Instance, User Task |
|  | 정책 및 의사결정 | 비즈니스 규칙 및 의사결정 자동화 | 규칙 엔진 통합, 의사결정 테이블 관리 | DMN, Drools, Rule Engine Integration |
| 분산 작업 오케스트레이션 | 장기 실행 워크플로우 | 오래 지속되는 작업 관리 | 지속성 있는 실행, 장애 허용 워크플로우 | Temporal, Cadence, Workflow as Code |
|  | 데이터 파이프라인 | 복잡한 데이터 처리 파이프라인 조율 | DAG 기반 워크플로우, 의존성 관리 | Apache Airflow(v2.2), Argo Workflows, DAG |
|  | 작업 스케줄링 | 주기적/일회성 작업 관리 | 크론 기반 스케줄링, 이벤트 기반 트리거 | Quartz, Scheduled Tasks, Event Triggers |
| 서비스 디스커버리 | 동적 서비스 검색 | 마이크로서비스 위치 자동 발견 | 서비스 등록/검색 메커니즘, 상태 체크 | Eureka, Consul, Kubernetes Service Discovery |
|  | 서비스 레지스트리 | 서비스 메타데이터 중앙 관리 | 서비스 카탈로그 구축, 버전 관리 | Service Registry, API Catalog |
|  | 부하 분산 | 서비스 인스턴스 간 부하 분산 | 다양한 로드밸런싱 알고리즘, 상태 기반 라우팅 | Client-side LB, Server-side LB, Health Checks |
| 회복성 패턴 | 서킷 브레이커 | 장애 격리 및 빠른 실패 | 회로 차단 임계값 설정, 폴백 메커니즘 구현 | Resilience4j, Spring Cloud Circuit Breaker |
|  | 재시도 및 백오프 | 일시적 장애 복구 | 지수 백오프, 지터 추가, 최대 재시도 제한 | Retry Pattern, Exponential Backoff, Jitter |
|  | 벌크헤드 | 리소스 격리 및 과부하 방지 | 세마포어/스레드풀 격리, 타임아웃 설정 | Bulkhead Pattern, Isolation, Resource Limits |
| 분산 트랜잭션 관리 | 사가 오케스트레이션 | 분산 트랜잭션 조율 | 중앙 오케스트레이터 기반 사가 구현 | Orchestration Saga, Coordinator Service |
|  | 사가 코레오그래피 | 이벤트 기반 분산 트랜잭션 | 이벤트 발행/구독 기반 사가 구현 | Choreography Saga, Event-Driven Architecture |
|  | 보상 트랜잭션 | 실패 시 일관성 보장 | 보상 작업 정의, 롤백 메커니즘 구현 | Compensating Transactions, Rollback Actions |

### 마무리말

NH 농협금융지주 통합 데이터 플랫폼의 서비스 통합 및 오케스트레이션 아키텍처는 복잡한 금융 서비스 프로세스를 효율적으로 관리하고 자동화하는 핵심 기반입니다. Istio 기반의 서비스 메시는 마이크로서비스 간 안정적인 통신과 고급 트래픽 관리를 제공하며, Camunda BPM을 활용한 비즈니스 프로세스 관리는 복잡한 금융 업무 흐름의 자동화와 시각화를 가능하게 합니다. 또한 Resilience4j를 적용한 회복성 패턴은 분산 시스템에서의 장애 격리와 신속한 복구를 지원하여 금융 서비스의 안정성을 크게 향상시킵니다. 사가 패턴 기반의 분산 트랜잭션 관리는 계열사 간 데이터 일관성을 유지하면서도 확장성 있는 시스템 구축을 가능하게 합니다. 이러한 종합적인 서비스 통합 및 오케스트레이션 전략은 농협금융지주가 고객 중심의 통합 금융 서비스를 유연하고 안정적으로 제공하는 기술적 토대가 될 것입니다.

## FOS

### 📌 ****FOS (File Object Storage) 솔루션 정리표****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **구분** | **세부 항목** | **주요 내용** | **설계 구현 전략** | **기술 스택** | **관련 서버** |
| 기능적 역할 | 파일 전송/수신 | 외부/계열사로부터의 배치 파일 수신, 내부 ETL 서버로 전달 | - 전용 수신 디렉토리 설정 - 사용자/계열사별 권한 분리 - 자동 모니터링 및 알림 구성 | SFTP, FTPS, SCP, rsync | MFT 서버 (FOS 클러스터) |
| 보안 정책 | 암호화/접근 제어 | 파일 전송 시 TLS 암호화, 접근 제어 및 계정 인증 강화 | - TLS 1.2 이상 적용 - IP Whitelist 기반 접속 허용 - 사용자별 접근 디렉토리 분리 | TLS, AES256, OpenSSH, PAM | FOS 전용 보안 게이트웨이 |
| 연계 방식 | 수신 후 연계 | 파일 수신 후 DataStage ETL로 전달 | - Watcher 기반 수신 감지 - ETL Trigger Script 호출 - 오류 시 알람 발송 | Bash, Control-M, Webhook | ETL 서버 (DataStage) |
| 운영 정책 | 파일 보관 정책 | 파일 보관 기간, 용량 제한 정책 등 운영 기준 수립 | - 일별/월별 자동 보관 정책 적용 - 보관 후 이중 백업 - 접근 로그 저장 | cron, logrotate, rsync | NAS, 백업용 스토리지 |
| 오류 대응 | 재전송/로그 추적 | 수신 실패 시 자동 재시도 및 장애 로그 저장 | - 이중 전송 정책 구성 - Failover 디렉토리 구분 - Slack/Email 기반 알림 | Retry Script, Slack API | MFT 서버 + 로깅 서버 |
| 자동화 구성 | 인터페이스 자동화 | 파일 인터페이스 자동 감지 및 처리 프로세스 자동화 | - 폴링 기반 Trigger 구성 - 파일 Signature 검증 적용 - Meta정보 별도 저장 | Bash, Python, Control-M | ETL Trigger용 Scheduler 서버 |
| 감사/로그 | 전송 이력 추적 | 계열사별 전송 이력 및 접근 내역 저장 | - 이력 로그 ELK 저장 - 전송 성공/실패 현황 시각화 - SLA 기준 로그 유지 | Fluentd, Logstash, Kibana | 로깅 서버, ELK 클러스터 |
| 연계 형식 | 포맷 및 규칙 | 배치 데이터 포맷 사전 협의, Naming 규칙 정형화 | - CSV/Fixed Length/JSON 등 표준 포맷 명세 적용 - Naming Rule 템플릿 정립 - 파일 Header/Trailer 적용 | Unix tools, JSON Validator | 모든 송신 계열사, ETL 서버 |
| 파일 검증 | 무결성 검사 | 파일 수신 후 Hash 값 비교 및 CheckSum 검증 | - MD5 또는 SHA256 Hash 저장 - 검증 실패 시 자동 차단 - 수신 확인 프로토콜 제공 | sha256sum, md5sum, Python | FOS 수신 서버 |
| 통합 관리 | 계열사 관리 기능 | 계열사 별 수신 현황, 파일 처리 상태 통합 관리 | - 관리자 대시보드 구성 - 수신 실패 로그 확인 - 전송 이력 보고서 생성 | Grafana, ELK, Slack Alert | 운영관리 서버, 모니터링 서버 |

## IBM DataStage 솔루션 정리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 설계 구현 전략 | 기술 스택 | 관련 서버 |
| 기능 역할 | ETL 처리 | 대량 데이터를 추출, 정제, 적재하는 ETL 플랫폼 | - 병렬 처리 기반 파이프라인 구성- 계열사별 데이터 로직 분리- GUI 기반 시각적 설계 | IBM DataStage (PX), DataStage Designer | ETL 서버 (Linux, 256GB RAM) |
| 구성 방식 | 병렬 아키텍처 | 병렬 처리 엔진을 기반으로 고속 대량처리 지원 | - SMP/MPP 모드로 Job 분산 실행- Partitioning 기능 활용- 대량 데이터 처리 병목 최소화 | DataStage PX Engine, APT Framework | ETL 전용 서버 클러스터 |
| 파일 연계 | 배치 연계 | FOS/MFT 서버로부터 수신한 파일 기반 데이터 적재 | - Pre-validation Job 구성- File metadata 자동 인식- Trigger 방식 연계 구성 | Unix Shell, Trigger Script, MFT/FOS | MFT 서버 ↔ ETL 서버 |
| 데이터 로딩 | Target 연계 | Oracle DB 및 통합 데이터 DB로 직접 로딩 | - JDBC 또는 Native Connector 활용- 데이터 무결성 체크- Merge, Upsert, Bulk Load 전략 적용 | Oracle Connector, ODBC, SQL Loader | 통합 데이터 DB 서버 |
| 스케줄링 | 배치 자동화 | Control-M 또는 자체 Scheduler 연계로 배치 자동 실행 | - 실행 순서 및 의존성 관리- 장애 발생 시 재시도 정책 구성- SLA 기반 배치 타임라인 구성 | Control-M, Cron, Python | ETL 서버 + 배치 스케줄러 서버 |
| 품질 검증 | 데이터 정합성 | 수신 데이터에 대한 유효성/정합성 검증 수행 | - Lookup 및 Join 기반 기준 데이터 검증- Stage간 Row Count 비교- 오류 데이터 별도 분리 저장 | QualityStage, Lookup Stage | ETL 서버 (DataStage Designer) |
| 모듈 구성 | Job 분류 관리 | 업무 단위/계열사 단위로 Job 분리 및 라이브러리화 | - Job Template 사전 정의- 계열사 마스터 테이블 기반 자동화- 재사용 가능한 공통 Stage화 | Job Sequence, Parameter Set | ETL 개발 및 운영서버 |
| 오류 처리 | 실패 처리 전략 | 오류 발생 시 별도 로그 수집 및 알림 구성 | - Reject Link 처리- Log 파일 → ELK 연동- 알림 전송(Slack, Email 등) | DataStage Director, ELK Stack | 로깅 서버, 알림 시스템 |
| 모니터링 | 실시간 모니터링 | Job 실행 상태 및 성능 모니터링 체계 구축 | - Stage별 처리 시간 분석- 실패 이력 추적- Grafana 대시보드 연계 | DataStage Operations Console, Grafana | 통합 모니터링 서버 |
| 보안 및 이력관리 | 감사 및 접근 제어 | Job 변경 이력, 사용자 접근, 데이터 흐름에 대한 로그화 | - Job 버전 관리- 사용자 권한(RBAC) 설정- 데이터 흐름 시각화 및 로그 보존 | LDAP, Git, DataStage Ops, Audit Log | 개발 서버, 보안 로그 저장소 |

**🧩 비고 및 활용 전략**

* NH 농협지주 데이터플랫폼 사업에서는 **계열사 배치 데이터 수집/정제/적재**를 중심으로 **DataStage를 표준 ETL 플랫폼**으로 활용합니다.
* 계열사/업무 도메인별로 Job을 모듈화하고, 통합 배치 스케줄링 체계를 통해 운영 자동화를 구현합니다.
* **FOS로 수신한 파일 → 전처리 → 통합DB 적재 → 마케팅/OLAP 연계**까지 전 과정을 실시간 모니터링과 이력 기반으로 관리합니다.

## 서버 정리

### ****통합 데이터 DB 서버****

통합 데이터 DB 서버는 **NH 농협지주 데이터 플랫폼 구축사업의 핵심 기반 인프라**로서, 전 계열사 및 데이터 소비 시스템들의 **데이터 통합, 저장, 조회, 분석의 중심 허브** 역할을 수행합니다. 단순한 저장소가 아니라, **데이터 품질 보장, 실시간 처리 대응, 분석 연계 기반 통합 운영의 최종 저장소**이며, 다음과 같은 이유로 반드시 필요합니다.

#### ✅ ****통합 데이터 DB 서버의 필요성 정리****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 설명 | 전략적 의미 |
| **1. 그룹 데이터 통합 저장소 역할** | 모든 계열사에서 수집한 데이터의 중앙 집중화 | 마케팅, 분석, 캠페인 등 전사 활용 기반 데이터 통합 |  |
| **2. 다양한 도메인 데이터 저장소** | 고객정보, 거래이력, 상품정보, 반응데이터, 분석결과 등 이기종 데이터 저장 | OLAP/BI/AI/마케팅/포털 등 전 영역의 데이터 활용을 위한 필수 조건 |  |
| **3. 다계층 시스템의 공통 접근 DB** | ETL, AP 서버, 포털, OLAP, 마케팅, UMS 등 모든 시스템이 접근 | API, JDBC, Kafka 등 연계 수단 통일 가능 |  |
| **4. 분석용 마트/정형 저장소로 활용** | 캠페인 타깃 추출, 실시간 개인화 추천 등을 위한 가공된 마트 저장 | 실시간 분석 및 이벤트 기반 마케팅 대응 가능 |  |
| **5. 메타 데이터 및 품질관리 기반 데이터 관리** | 메타정보, 품질이력, 정책기반 통제 테이블 구성 | 데이터 거버넌스 기반 표준화 가능 |  |
| **6. OLAP 시스템 연동 및 대용량 조회 최적화** | 정형분석/리포팅 기반 대용량 데이터 조회 최적화 구조 구축 | BI 포털, OLAP 분석 리포트 고속 조회 가능 |  |
| **7. API 기반 실시간 서비스 대응 저장소** | 통합 데이터 AP 서버를 통해 API 제공 시 조회되는 실시간 DB | 포털, 외부 API 서비스, 타겟 추출 요청 등에 즉시 응답 가능 |  |
| **8. 이력 관리 및 감사 대응 기능** | 데이터 변경, 적재, 삭제, 조회 이력 관리 및 감사 기능 필요 | ISMS-P, 금융보안원 감사 대비 |  |
| **9. 대용량 처리 및 고성능 구조 설계 필요** | 캠페인 타겟 추출, 통합 조회, 로그 분석 등 고TPS 응답 요구 | 성능 튜닝된 인덱싱, 파티셔닝, 병렬쿼리 구조 설계 필수 |  |
| **10. 백업 및 DR 대상 핵심 자산** | 복구, 장애대응, DR Site 운영 시 대상이 되는 가장 중요한 자산 | BCP/DRP/재해복구 체계 구축의 중심 저장소 |  |
| 구분 | **세부 항목** | **주요 내용** | **전략적 의미** |
| 1 | 그룹 데이터 통합 저장소 | 계열사·은행·채널 등 모든 주체로부터 수집된 데이터를 통합 저장 | **데이터 중심 거버넌스 실현**, 전사 분석·공유·활용 기반 확보 |
| 2 | 이기종 데이터 통합 | 고객, 거래, 상품, 이벤트, 로그 등 이기종 데이터 구조 수용 | **마이크로서비스/이벤트 기반 처리 통합 대응** |
| 3 | 공통 연계 대상 DB | AP 서버, ETL, OLAP, 포털, 마케팅 시스템 등 모든 시스템과 연계 | **서비스 간 데이터 일관성 유지**, **시스템 통합성 향상** |
| 4 | 분석 마트 저장소 | 정형 분석, 비정형 분석, 타깃 추출용 마트 구성 가능 | **BI·AI 시스템의 성능 최적화**와 **정책 기반 추천 정밀화** |
| 5 | 메타 및 품질 데이터 통합 | 메타데이터, 품질 상태, 이력 데이터 등을 DB에 통합 관리 | **데이터 신뢰도 향상**, **정책 기반 데이터 운영 가능** |
| 6 | 실시간 API 서비스 대응 | API 서버와 연계하여 실시간 조회 요청에 빠르게 응답 | **마케팅 자동화, 사용자 인터페이스 UX 향상에 기여** |
| 7 | OLAP 연계 구조 | OLAP 분석 시스템에서 직접 연계해 집계·분석 수행 | **대용량 OLAP 처리 최적화**, **중복 저장소 방지** |
| 8 | 이력 관리 및 감사 대응 | 데이터 변경 이력, 조회 이력, 삭제 이력 등 통합 관리 | **보안·규제 대응 (ISMS-P, 금융보안원 감사)** |
| 9 | 고성능 설계 기반 | 병렬 쿼리, 파티셔닝, 인덱싱 등 성능 기반 구조 설계 | **실시간 분석/캠페인 타깃 추출 시 안정적 성능 보장** |
| 10 | DR 및 백업 구조 | 복제, 스냅샷, DR Site 구성 가능 | **재해 복구 및 무중단 서비스 보장**으로 운영 연속성 확보 |

#### 통합 데이터 DB 서버 기반 아키텍처 도식도

[1] 계열사 Legacy 시스템

├── 배치 (FILE, FOS, MFT)

│ └──▶ [통합 데이터 ETL 서버]

│ ├── Data 정제 및 가공

│ └── JDBC 연계

│ └──▶ [통합 데이터 DB 서버]

│

└── 온라인 (HTTP, API, 대외 MCA)

└──▶ [통합 데이터 AP 서버]

└── JDBC 연계

└──▶ [통합 데이터 DB 서버]

[2] 통합 데이터 DB 서버

├── 데이터 마트 (고객, 상품, 거래, 마케팅 등)

├── 통합 고객번호 기반 통합 저장소

├── 분석용 정형/비정형 데이터 적재

├── 운영/분석 데이터 분리 설계

└── 데이터 보안 및 암호화 (TDE, 접근제어, 감사 로그)

[3] 데이터 소비 시스템

├──▶ [통합 데이터 포털 시스템]

│ └── JDBC, REST API 연계

├──▶ [BI 포털 시스템]

│ └── 메타 데이터 조회 및 리포팅 요청

├──▶ [OLAP 시스템]

│ └── 복수 마트 병합 Query, 다차원 분석

└──▶ [마케팅 시스템 / e-마케팅 / 캠페인]

└── 캠페인 대상 고객 추출, 실시간 응답 제공

[4] 연계 기술 구성

├── JDBC, REST API, OAuth2 인증 연계

├── Oracle HEX, RAC 구성 (HA 및 확장성)

└── 성능 튜닝 및 분산 인덱스 구조 운영

##### ✅ 전략적 의미 요약

|  |  |
| --- | --- |
| 구분 | 전략적 의미 |
| **데이터 중심 허브** | 통합된 고객·거래·마케팅 정보를 단일 DB에 집적하여 마스터 데이터 및 분석 기반을 확보 |
| **신뢰성 및 정합성** | ETL/CDC 기반의 정제 프로세스를 통해 무결한 데이터만 적재, 분석 신뢰도 향상 |
| **확장성과 고가용성** | RAC 또는 Oracle HEX 기반 고가용 구조, 향후 데이터 증가에 유연한 대응 가능 |
| **보안 및 감사 체계** | 데이터 접근 통제, 암호화, 감사 로그 관리로 금융보안 기준 만족 |
| **API·분석 시스템 연계** | 통합 데이터 AP를 통한 다양한 시스템과의 연계 가능성 확보 |

#### 🔧 ****적용 기술 스택 (예시)****

|  |  |
| --- | --- |
| 항목 | 기술 스택 |
| DBMS | **Oracle HEX**, PostgreSQL, MySQL (BI 포털), Redis (Cache) |
| 접속 기술 | JDBC, Kafka Sink, API Backend ORM (JPA, Hibernate) |
| 보안 관리 | Oracle TDE, DB 암호화, DB 접근제어, 감사로그 |
| 성능 최적화 | 파티셔닝, 병렬처리, 인덱싱, 메모리 캐시 연계 |
| 가용성 구조 | RAC, Active-Standby, DR Sync, RMAN 백업 |

#### 🧭 ****설계 시 고려 포인트****

* **모듈별 DB 분리 여부** (정형/비정형/OLAP/로그용)
* **BI용 DB 분산 vs 단일화 여부**
* **실시간 API 조회 대비 인덱싱 전략**
* **캠페인 시스템 연계 트래픽 집중에 대한 Read Replica 구성 여부**
* **DR 및 백업 리스크 관리 계획 수립**

#### 💡 ****결론****

**통합 데이터 DB 서버는 농협지주 전사 데이터를 통합 관리하고 서비스 연계의 중심 허브로 기능하는 ‘데이터 플랫폼의 심장’입니다.**  
데이터 수집부터 분석, 활용, 공유까지의 전 과정에서 단일화된 저장소로 기능하며, 데이터 품질, 보안, 성능, 확장성 모두를 고려한 설계가 필수입니다.

### ****통합 데이터 ETL 서버****

#### 통합 데이터 ETL 서버 필요성

|  |  |  |
| --- | --- | --- |
| 구분 | 세부 설명 | 전략적 의미 |
| **1. 이기종 계열사 데이터 수집 통합** | NH금융그룹의 다양한 계열사(은행, 카드, 보험, 증권 등)의 시스템으로부터 수집되는 구조, 포맷, 스키마가 서로 다른 데이터를 표준화 처리 | 다양한 원천 시스템의 데이터 연계를 통합 처리함으로써 연계 복잡도를 단일화하고 유지보수 효율 극대화 |
| **2. 데이터 전처리 및 정제** | 수집한 데이터에 대해 Null 제거, 이상치 제거, 포맷 표준화, 정합성 체크, 라벨링 등 다양한 품질 전처리 수행 | 정제되지 않은 데이터로 인해 발생하는 마케팅, 분석, 캠페인 품질 저하 방지 |
| **3. 데이터 적재 및 마트화 자동화** | 통합 데이터 DB 및 분석용 마트(OLAP, 캠페인 마트 등)로의 적재 로직 자동 실행 | 수작업 개입 없이 안정적인 일관 데이터 흐름 유지 |
| **4. 배치/실시간 하이브리드 연계 지원** | 계열사와의 연계는 배치(MFT/File), 실시간(Kafka, API)으로 혼재되어 있어 이를 모두 수용 가능한 유연한 ETL 구조 필요 | 복잡한 연계 구조를 단일 플랫폼에서 관리하고, 향후 API 기반 연계 전환도 유연히 대응 가능 |
| **5. 이력관리 및 데이터 흐름 추적** | 수집/정제/적재 과정에서의 로그, 상태, 오류, 파기 처리까지 관리 필요 | 금융권의 감사·보안 요건 충족 (데이터 추적성, 이력성 보장) |
| **6. 재사용 가능한 파이프라인 설계** | 공통 정제, 공통 파싱, 공통 적재 모듈 구성 필요 | 신규 데이터소스 추가 시 재사용성 증가로 비용 및 시간 절감 |
| **7. 확장성과 성능 보장** | 대용량 데이터의 빠른 병렬 처리와 분산처리 구조를 지원해야 함 (멀티노드, 멀티스레드 기반 설계) | 대량 배치 처리 시간 단축, 시스템 병목 방지 |
| **8. 데이터 폐기 및 보안처리 자동화** | 금융 규제 준수(예: 일정 기간 후 자동 파기) 및 민감정보 암호화·비식별화 지원 | ISMS-P, 개인정보보호법, 내부통제 준수 기반 설계 가능 |
| **9. 운영 자동화 및 모니터링 지원** | 작업 스케줄링, 실패 시 재시도, Slack/메일 알람 등 자동화 연계 필요 | 운영 효율화 및 장애 발생 시 빠른 복구 가능 |
| **10. 외부 연계 후속 처리 트리거 제공** | ETL 완료 후 통합데이터AP, 캠페인시스템, 마케팅시스템에 RESTful API 연계 트리거 발생 | 실시간 타겟 고객 추출, 이벤트 마케팅 연동 등 선제적 대응 가능 |

#### 통합 데이터 ETL 서버 기반 아키텍처 도식도

다음은 **통합 데이터 ETL 서버**를 중심으로 한 아키텍처 도식도를 **텍스트 기반**으로 작성한 예시입니다. 본 구성은 NH 농협지주 데이터 플랫폼 내에서 **데이터 수집 → 정제 → 적재 → 활용**의 흐름에서 ETL 서버가 수행하는 중추적 역할에 초점을 둡니다.

|  |
| --- |
| [1] 데이터 원천 계열사 시스템  ├── 은행 정보계  │ └── FILE 전송 (제3자 동의 고객정보, 거래정보 등)  ├── 카드 정보계  │ └── FILE 전송 (상품정보, 제4자 동의 정보 등)  ├── NH 생명/손해보험  │ └── 대외 MCA → FILE 전송 (FOS / MFT)  └── 빅스퀘어, 마케팅허브, e-마케팅  └── FOS / MFT 연계 → 대상 고객, 성과 반응 데이터 수신  ⇩  [2] 통합 데이터 ETL 서버 (IBM DataStage 기반)  ├── 수신 모듈  │ ├── FOS/MFT 파일 수신  │ └── 파일 스케줄링 및 이력 관리 (Control-M)  ├── 정제 처리 모듈  │ ├── 데이터 유효성 검사 (포맷/Null/오류값)  │ ├── 고객/상품 기준정보 매핑  │ ├── 중복/무결성 검증  │ └── 데이터 마스킹 및 개인정보 보호 로직 적용  ├── 적재 처리 모듈  │ ├── 데이터 마트 대상 추출 및 집계  │ └── Oracle DB 연동 (JDBC)  ├── 이력 로깅 및 감사  │ ├── 적재 로그 저장  │ └── 배치 처리 성공/실패 이력 기록  └── 오류 처리 및 알림  ├── 오류 데이터 분리 및 재처리 로직  └── 운영자 알림 (Slack, SMS, Email)  ⇩  [3] 데이터 적재 대상  ├──▶ [통합 데이터 DB (Oracle HEX)]  │ ├── 고객/상품/거래/마케팅 통합 마트  │ └── 분석 및 시각화 대상 테이블 구성  └──▶ [빅스퀘어] / [마케팅 허브]  └── PUSH 방식 전달 (FOS, MFT, JDBC 연계)  ⇩  [4] 데이터 활용 시스템  ├── 통합 데이터 포털 (JDBC 연계)  ├── OLAP (다차원 분석)  ├── BI 포털 (정형 리포트)  └── 마케팅/캠페인 시스템 (대상 추출, 반응 추적) |

##### ✅ 전략적 의미 요약

|  |  |
| --- | --- |
| 구분 | 전략적 의미 |
| **데이터 수집 자동화** | 다계열사에서 오는 비정형/정형 데이터를 FOS, MFT를 통해 자동 수신 |
| **정제·품질 향상** | ETL 정제 모듈에서 유효성, 무결성, 개인정보 보호 정책을 통합 처리 |
| **분산처리 기반 적재** | 대량의 데이터를 멀티 잡으로 병렬 처리하여 ETL 성능 확보 |
| **보안 및 감사 대응** | 배치 로그, 적재 로그, 에러 추적 이력으로 규제 대응 체계 완비 |
| **데이터 유통 허브화** | 정제된 데이터가 DB뿐만 아니라 외부 분석계, 마케팅계로 분산 제공됨 |

#### 🔧 적용 기술 스택 예시

|  |  |
| --- | --- |
| 항목 | 사용 기술 |
| ETL 솔루션 | **IBM DataStage**, Apache NiFi, Talend 등 |
| 스케줄링 | Control-M, Airflow, Cron |
| 로그/모니터링 | ELK Stack, Prometheus, Grafana |
| 보안/이력관리 | DB 암호화, 로그 이력 저장, 파기 모듈 |
| DB 연계 | JDBC, Oracle, PostgreSQL, Kafka Connect 등 |

#### 💡 결론

**통합 데이터 ETL 서버는 단순한 데이터 전달 장치가 아니라, “데이터의 품질, 흐름, 규제준수, 성능”을 통합 보장하는 중심 허브입니다.**

특히 **배치 기반 계열사 연계**, **정제 후 OLAP/분석/마케팅 시스템 연계**, **이력관리 및 자동화** 등은 이 ETL 서버 없이는 구현이 불가능하거나 비효율적입니다.

### ****통합 데이터 AP 서버****

#### ****통합 데이터 AP 필요성****

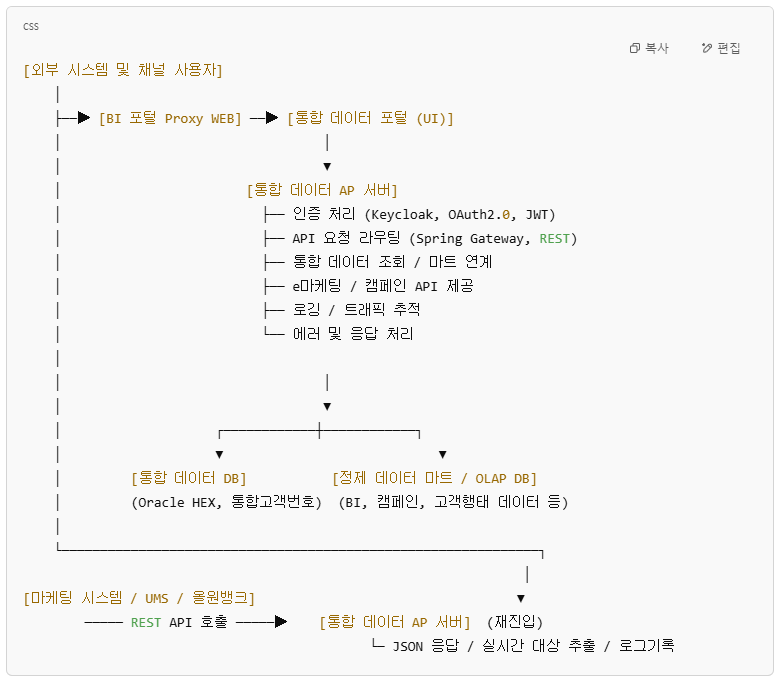
|  |  |  |
| --- | --- | --- |
| 구분 | 필요성 설명 | 전략적 의미 |
| **1. 분석 서비스의 비즈니스 로직 처리 허브** | OLAP, BI, 포털 등에서 요청되는 **데이터 분석/조회/가공 로직**을 수행 | DB 직접접속 부담을 줄이고, **중간 계층에서 로직을 분산** 처리 |
| **2. 외부 인터페이스 연계 API 게이트웨이 역할** | e마케팅, UMS, 포털, 캠페인 시스템 등에서 **REST API 호출을 통해 요청**을 수신 | 외부 연계를 일관된 구조로 제공하고, **다중 시스템 간 통합 API 관리** 가능 |
| **3. 데이터 통합 서비스 포인트** | 다양한 마트에서 정제된 데이터를 통합 조회하거나, 교차 비즈니스 로직을 수행 | **분산 마트** 기반에서 통합 응답을 구성할 수 있는 **비즈니스 추상화 계층** |
| **4. 인증/인가 및 보안 처리 연계** | OAuth2 + JWT 인증, Keycloak을 기반으로 **API 호출 인증과 권한 제어 수행** | 데이터 접근을 **보안 정책 기반으로 제어**, 금융보안 규제 대응 가능 |
| **5. API 형태의 통합 포털 서비스 제공** | 통합 데이터 포털, BI 포털, OLAP 등 **UI 시스템의 백엔드 요청을 수신**하여 처리 | API 중심 서비스 구조로 전환하여, **유연한 확장과 분리된 UI/로직 구조 구현** |
| **6. 외부 시스템 요청에 대한 단일 응답 엔드포인트** | 계열사, 은행 등에서 통합고객번호 기반 데이터 요청 시, **집계/변환/응답처리 수행** | 요청-응답 구조를 **표준화**, API 스펙을 통합 제공 가능 |
| **7. 실시간 응답형 캠페인 서비스 기반 확보** | 마케팅 시스템이 타겟 추출, 캠페인 실행 요청 시 AP 서버가 **기준 데이터를 실시간 응답** | 실시간 서비스 모델을 구성할 수 있는 핵심 구성요소로 작동 |
| **8. DB에 대한 접근 계층 분리 및 보안 확보** | BI 포털, OLAP, 포털 등에서 DB 직접 접근이 아닌 **AP 서버 경유 처리**로 통제 가능 | DB 보안을 강화하고, **DB접속의 관리 지점을 중앙화** |

통합 데이터 AP 서버는 단순 API 중계 기능을 넘어서,

🔐 **보안 관문**,  
🔄 **비즈니스 로직 수행 계층**,  
🔗 **외부 인터페이스 허브**,  
📈 **분석 기반 응답 처리 노드**  
역할을 동시에 수행합니다.

특히 농협지주처럼 **다양한 계열사 + 외부기관 + 내부 분석 시스템**이 동시에 연계되는 구조에서는  
**AP 서버 없이 데이터 플랫폼 전체를 연결하는 안정성과 유연성을 확보할 수 없습니다.**

#### 통합 데이터 AP 서버 기반 아키텍처 도식



#### 주요 기술 컴포넌트

|  |  |  |
| --- | --- | --- |
| 계층 | 컴포넌트 | 기술 스택 |
| 프론트엔드 | BI 포털 UI, 데이터 포털 UI | Vue.js, React, HTML5 |
| API 게이트웨이 | 통합 데이터 AP 서버 | Spring Boot, Spring Cloud Gateway |
| 인증 계층 | 인증/인가 처리 | OAuth2.0, JWT, Keycloak |
| DB 연계 | 통합 DB, 마트 DB | Oracle HEX, PostgreSQL, JDBC |
| 비즈니스 로직 | 데이터 필터링, 집계, 정책기반 응답 | Java, JPA, RESTful API |
| 로깅/모니터링 | 요청 로그, 에러 로그, 응답 추적 | ELK Stack, Prometheus, Grafana |
| 보안 관리 | API 권한, 요청 제한 | Rate Limiting, API Key, CORS, HMAC |

## 서버 기술스택 매핑 정리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **서버 구분** | **주요 역할** | **운영 체계** | **배치 기술 스택** | **연계 기술** | **활용 프레임워크** |
| **WEB 서버** | 사용자 UI 제공, 포털 진입 지점 | Linux (PaaS 기반) | xFrame5, Tomcat, Tanzu, Deep Security | HTTP(S), JWT | Spring Boot, React, REST API |
| **PT WAS 서버** | 통합 데이터 포털 업무처리 (PT 영역) | Linux (컨테이너 기반) | xDataSet5, SpringBoot, Tanzu, Deep Security | REST API, OAuth2.0 | Spring MVC, API Gateway |
| **BT WAS 서버** | 통합 데이터 포털 백엔드 로직 처리 | Linux (컨테이너 기반) | SpringBoot, Kafka Client, Redis | Kafka, JDBC | Microservice + Event Driven |
| **통합 데이터 AP** | API 처리, 외부 연계, 인증 중계 | Linux (컨테이너 기반) | SpringBoot, Keycloak, Kafka Client, JWT | REST API, OAuth2, JWT | API Gateway, Kafka Consumer |
| **ETL 서버** | 배치 데이터 수집, 정제, 적재 | Linux (On-premise) | IBM DataStage, Control-M Client, Oracle Client | FOS/MFT, JDBC | ETL Flow, Shell + DataStage PX |
| **통합 데이터 DB** | 원천 데이터 저장 및 분석 마트 | Linux (물리/가상 혼합) | Oracle HEX, Oracle TDE, DB Safer | JDBC, SQL Loader | PL/SQL, DDL/DML Script 기반 |
| **BI 포털 서버** | 데이터 시각화 및 분석 서비스 제공 | Linux (클라우드 기반) | SpringBoot, MySQL, Tableau, MSTR | REST API, JDBC | BI Portal Framework |
| **통합 데이터 포털** | 사용자 데이터 탐색, 요청, 보고서 생성 | Linux (컨테이너 기반) | React, SpringBoot, JWT, MySQL | REST API, JDBC | SPA + Spring 기반 UI |
| **Kafka 브로커 서버** | 실시간 데이터 전송 및 이벤트 처리 | Linux | Apache Kafka, Zookeeper | Kafka Topic, Consumer Group | Event Streaming + EDA |
| **MFT/FOS 서버** | 계열사 파일 수신 및 전달 | Linux | FOS Agent, SFTP, 파일 암호화 | SFTP, Trigger, Shell Script | 배치 연계 처리 엔진 |
| **OLAP 서버** | 분석 연계 및 마트 통합 조회 | Linux | Tableau, MSTR, BI Query | JDBC, API | 다차원 분석 연계 |
| **모니터링 서버** | 로그 및 시스템 메트릭 통합 분석 | Linux | Prometheus, Grafana, Fluentd, ELK | HTTP, Pushgateway | Metrics + Alert 기반 |
| **CI/CD 서버** | 코드 빌드, 배포 자동화 | Linux (클라우드 또는 온프레미스) | Jenkins, GitLab CI, Argo CD, Helm | GitOps, Docker Registry | Git 기반 DevOps 파이프라인 |
| **보안 인증 서버** | 사용자 인증 및 권한 관리 | Linux | Keycloak, OAuth2, JWT Provider | Token, OpenID Connect | IAM + 인증 게이트웨이 |

🧩 **전략적 의미 요약**

* **표준화된 기술 구성**으로 각 서버 유형에 적합한 컴포넌트를 도입
* **PaaS 기반 탄력적 배포 구조** → 컨테이너 기반 관리 (Tanzu, Kubernetes)
* **실시간-배치 연계 이중 구조** → Kafka + MFT 혼용으로 다양한 업무 대응
* **보안 연계 강화** → Deep Security, OAuth2, DB TDE 등 적용
* **DevSecOps 자동화** → GitOps & CI/CD 통한 민첩한 릴리즈 가능

## ✅ ****🔹 서버별 보안 설정 정책 정리표****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **서버 유형** | **보안 항목** | **설정 내용** | **적용 솔루션/기술** | **전략적 의미** |
| **WEB / UI 서버** | Web 접근제어 | xFrame, CORS, CSP 적용 | Nginx, Spring Security | 브라우저 기반 보안 위협 차단 |
|  | 인증 연계 | OAuth2 + JWT 기반 인증 | Keycloak, Spring Security | API 호출에 대한 보안 및 사용자 인증 강화 |
| **WAS 서버** | 실행 권한 제어 | 최소 실행 권한 Principle 적용 | Linux ACL, AppArmor | WAS 침해 가능성 차단 |
|  | 파일 무결성 점검 | Hash 검증 및 Tamper Alert | Tripwire, AIDE | 변조 탐지 및 운영 신뢰도 확보 |
| **ETL 서버** | 배치 스크립트 보안 | Shell Script 암호화/접근제어 | SecureCRT, SSH Key | 불법 접근 및 실행 차단 |
|  | 통신 암호화 | MFT 전송 구간 암호화 적용 | FOS, TLS | 민감 데이터 보호 |
| **DB 서버** | 저장 데이터 보호 | TDE(Transparent Data Encryption) 적용 | Oracle TDE, DB Safer | 저장 시 데이터 유출 방지 |
|  | 접근 감사 로깅 | DB 접속 이력 로깅 및 분기 감지 | Audit Vault, DB Firewall | 관리자 행위 감시 및 내부 위협 대응 |
| **Kafka 서버** | 메시지 무결성 검증 | TLS + SASL 인증 적용 | Apache Kafka, Zookeeper | 이벤트 메시지 조작 방지 |
| **CI/CD 서버** | 권한 제어 및 비인가 방지 | Git 권한 계층, Secret Management | Vault, GitLab IAM | DevOps 내부 침해 경로 제거 |

## ✅ ****🔹 컨테이너 기반 배포 컴포넌트 구성도 (PaaS 기반)****

[ 사용자 ]

↓

[ Load Balancer (L4/WAF) ]

↓

[ Ingress Controller ]

↓

[ API Gateway (Zuul / Kong) ]

↓

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

| PaaS (Tanzu / K8s 기반) |

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

| - Namespace: web-frontend |

| └── POD: React, Vue UI |

| - Namespace: portal-api |

| └── POD: Spring Boot |

| - Namespace: auth |

| └── POD: Keycloak |

| - Namespace: batch |

| └── POD: DataStage |

| - Namespace: logging |

| └── POD: Fluentd, Loki |

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

↓

[ Monitoring Cluster ]

└─ Prometheus, Grafana, ELK

## ****🔹 ETL – DB – 포털 연계 구조 상세도****

[계열사 Legacy / 외부 시스템]

└─ File 수집 (FOS/MFT) or API (HTTP)

↓

[통합 데이터 ETL (DataStage)]

- 전처리, 표준화, 정제

- 타깃 마트 분기 로딩

- 이력 저장

↓ (JDBC)

[통합 데이터 DB (Oracle HEX)]

- DW / DM 설계

- 통합고객번호 기반 인덱싱

- BI 분석용 데이터 준비

↓ (JDBC, REST)

[통합 데이터 포털 (SpringBoot + React)]

- 사용자 요청 처리

- 필터 적용 후 분석 데이터 조회

- 메타정보 관리

↓

[BI 포털 / OLAP / 마케팅 시스템]

- BI 리포팅

- 캠페인 타겟 추출

- 사용자 요청 처리 API

## NH 농협지주 통합 데이터 플랫폼 – Big Picture 구성도

|  |
| --- |
| [계열사 시스템 (Legacy)]  ┌────────────┐  ┌────────────┐ │NH 투자증권│  │NH 생명보험 │────┤NH 손해보험│  └────────────┘ └────────────┘  │ │  ▼ ▼  (MFT / FOS / 대외 MCA) ← (배치 + API 연계)  │  ▼  ┌────────────────────────────┐  │ [통합 데이터 ETL] │ ← DataStage 기반 전처리  │ - 정제, 로딩, 이력관리 │  └────────────────────────────┘  │  (JDBC / FOS)  ▼  ┌────────────────────────────┐  │ [통합 데이터 DB] │ ← Oracle HEX  │ - DW, DM, 통합 고객번호 │  └────────────────────────────┘  │  (API / JDBC / Kafka Stream)  ▼  ┌────────────────────────────────────────────┐  │ [통합 데이터 AP] (SpringBoot 기반) │  │ - API 게이트웨이 역할 (JWT, OAuth2 인증) │  │ - 외부 요청 처리 (BI, 캠페인, OLAP) │  └────────────────────────────────────────────┘  │  ┌──────┼──────────────────────┐  ▼ ▼  [통합 데이터 포털] [마케팅 시스템 / E-마케팅 / HUB]  - 메타 조회 / 분석 UI - 캠페인 타겟 발송  - 사용자 권한 기반 분석 - 사용자별 실시간 이벤트 처리  - JDBC + REST 연계 - Kafka 기반 이벤트 연계  │ │  └────────────┬────────────────┘  ▼  [BI 포털 / OLAP 시스템]  - 시각화 리포팅  - 통계/추론 분석  │  ▼  [채널 사용자 (웹/모바일/올원뱅크)] |
|  |

### 🔍 설명 및 전략 포인트

|  |  |  |
| --- | --- | --- |
| 구성 영역 | 핵심 요소 | 전략적 의미 |
| **수집 계층** | FOS, MFT, API | 계열사/내부 정보계로부터의 배치 + 실시간 연계 이중화 구성 |
| **ETL 계층** | DataStage | 대용량 데이터 정제 + 마스킹 + 표준화 처리 |
| **저장 계층** | Oracle HEX | DW+DM 통합 / 통합 고객번호 기반 관리 |
| **서비스 계층** | 통합 데이터 AP | 모든 시스템 연계의 중심, SpringBoot 기반 API 서비스 허브 |
| **포털 / 분석** | 통합 데이터 포털 / BI / OLAP | UI/UX 기반 분석 환경 제공, 실시간 데이터 추론 |
| **마케팅 시스템** | Kafka 기반 비동기 연계 | 이벤트 기반 고객 타겟팅, 캠페인 발송 통제 |
| **최종 사용자 채널** | 채널 사용자 (PC/Mobile) | 분석 기반 데이터 소비/조회 |

## OLAP-BI 포털간 데이터 통합 분석 흐름도

|  |
| --- |
|  |
| [1] 데이터 수집 및 적재  ─────────────────────────────────────────────  [계열사 Legacy / NH 계열사]  └─▶ [FOS / MFT / API / 대외 MCA]  └─▶ [통합 데이터 ETL]  └─▶ 정제/적재/변환  (DataStage 기반)  └─▶ 이력 로깅 및 메타 저장  [2] 통합 저장소 구성  ─────────────────────────────────────────────  [통합 데이터 DB (Oracle HEX)]  ├─▶ DW (Data Warehouse): 원천/마트/메타 통합  ├─▶ DM (Data Mart): 주제영역별 마트 구성  └─▶ 통합 고객번호/행태정보/이력정보 저장  [3] 분석 처리 및 가공  ─────────────────────────────────────────────  [OLAP 시스템]  ├─ JDBC 방식으로 통합 DB 직접 접근  ├─ 다차원 분석 모델 구성 (Cube 설계)  ├─ KPI 및 Target 고객 집계 쿼리 수행  └─ 분석 결과 생성 (SQL/MDX 기반 결과 셋)  [4] 시각화 및 리포팅  ─────────────────────────────────────────────  [BI 포털 시스템 (SpringBoot + MicroStrategy)]  ├─ OLAP 결과 수신 (JDBC or REST API)  ├─ 메타데이터 기반 대시보드 구성  ├─ 권한 기반 리포트 공유/다운로드 제공  ├─ Tableau / MicroStrategy 연동  └─ 마케팅 포털, 관리자 시스템과 연계  [5] 사용자 활용 및 액션  ─────────────────────────────────────────────  [내부 사용자 / 마케터 / 관리자]  └─▶ BI 포털 접속 (SSO 연계 + 권한 검증)  └─▶ 리포트 조회, 타겟 고객 추출  └─▶ 통계 기반 캠페인 설계 연계 |

### ­📌 주요 구성요소 요약

|  |  |  |  |
| --- | --- | --- | --- |
| 구성요소 | 역할 | 연계 방식 | 핵심 기술 |
| **ETL** | 계열사·내부 데이터 정제 및 적재 | FOS, MFT, API | DataStage, Control-M |
| **통합 DB** | 분석용 DW/DM 저장소 | JDBC | Oracle HEX, Data Vault |
| **OLAP** | 다차원 분석 / KPI 연산 | JDBC | OLAP Server, Cube Engine |
| **BI 포털** | 시각화 리포트 및 대시보드 제공 | JDBC, REST | SpringBoot, MicroStrategy, Tableau |
| **사용자 영역** | 리포트 활용 / 고객 추출 | SSO + 권한 기반 접근 | SSO, RBAC, Web UI |

## ✅ **데이터 수집·정제·적재·분석·활용 흐름도 (텍스트 기반 도식)**

|  |
| --- |
| [1] 데이터 수집 (원천 데이터 확보 단계)  ─────────────────────────────────────────────  [계열사 Legacy 시스템]  ├─▶ 배치 (FILE) ─▶ [FOS] ─▶ [MFT] ─▶ [ETL 서버]  └─▶ 실시간 (API) ─▶ [대외 MCA] ─▶ [통합 데이터 AP] ─▶ [ETL DB Queue]  [은행 정보계 / 카드 정보계 / 외부 계열사]  └─▶ 제3자/제4자 동의 고객정보, 거래정보, 상품정보 등 전송  [2] 데이터 정제 및 전처리 (품질 확보 단계)  ─────────────────────────────────────────────  [ETL 서버 (IBM DataStage)]  ├─▶ 원천 파일 검증 (파일무결성, 필드유효성 체크)  ├─▶ 스키마 매핑 / 표준화 (공통코드 적용, 결측 처리)  ├─▶ 로그 적재 (정제 로그, 실패 건 별도 저장)  └─▶ 오류건 재처리 및 수동 확인 (반복 Job 관리)  [3] 데이터 적재 (통합 저장소 구성)  ─────────────────────────────────────────────  [ETL 서버]  └─▶ JDBC 기반 적재 ─▶ [통합 데이터 DB (Oracle HEX)]  ├─▶ 원천 데이터 저장 테이블  ├─▶ 표준화 마트 (Normalized Mart)  ├─▶ 분석전용 Raw Zone, Data Lake 구조  [4] 데이터 분석 및 모델링 (의사결정 인사이트 확보)  ─────────────────────────────────────────────  [빅스퀘어 / OLAP 시스템]  ├─▶ JDBC or Kafka Stream으로 데이터 수집  ├─▶ 머신러닝 모델 학습 및 예측  ├─▶ 다차원 분석 (정형 + 비정형)  └─▶ 결과값 JSON or File로 제공  [BI 포털 / 통합 데이터 포털]  └─▶ OLAP 연계 리포트 제공  └─▶ 메타 데이터 기반 KPI 조회  └─▶ 사용자 권한 기반 분석 리포트 구독  [5] 데이터 활용 및 피드백 (실행 기반 가치 전환)  ─────────────────────────────────────────────  [마케팅 시스템 / 캠페인 시스템]  ├─▶ 타겟 고객 추출 요청 (BI → AP → DB)  ├─▶ 캠페인 등록 및 실행 (e-마케팅, HUB)  └─▶ UMS 연계 메시지 발송 (SMS, 알림톡 등)  [성과 데이터 → ETL 재적재]  ├─▶ Push 반응률, 오픈율 등 수집  └─▶ 성과분석 결과 BI 포털로 재반영 |

구성요소 요약표

|  |  |  |  |
| --- | --- | --- | --- |
| 단계 | 주요 시스템 | 핵심 기술 | 전략적 의미 |
| 수집 | 계열사, 외부기관 | MFT, API, 대외MCA | 다양한 채널의 통합 수집 |
| 정제 | ETL 서버 (DataStage) | 정합성 검사, 스키마 정제 | 품질 기반 분석 정확도 확보 |
| 적재 | Oracle HEX | JDBC, TDE 암호화 | 통합저장소 및 마트 구성 |
| 분석 | OLAP, BI, 빅스퀘어 | Kafka, 모델링, Tableau | 인사이트 도출, 고급 분석 |
| 활용 | 마케팅 시스템, 캠페인, UMS | REST API, JWT 인증 | 분석→실행의 피드백 루프 |

## ✅ NH농협지주 데이터 플랫폼 To-Be 아키텍처 구성 요약

### 📌 ****1. 전체 데이터 흐름 영역 구분****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 주요 시스템 | 주요 기능 및 구성 | 연계방식 |
| **① 데이터 수집** | 은행 정보계, 카드 정보계, 계열사 시스템 등 | 고객, 거래, 상품 정보 수집 | MFT, API, 대외 MCA |
| **② 데이터 저장** | 빅스퀘어, 통합 데이터 저장소 | 수집/저장, 분석마트, 샌드박스, SAS 분석 등 | 내부 저장소 연계 |
| **③ 데이터 분석·활용** | 통합 분석, BI 포털 | OLAP, 통계, 리포트, 상품 추천, KPI 분석 | Kafka, Batch, API |
| **④ 마케팅 채널 전파** | 마케팅 시스템, 올원뱅크, UMS | 실시간/비동기 이벤트 연계, 성과 반영 | API, Push, E-Mail |

### 📌 ****2. 통합 데이터 플랫폼 주요 컴포넌트 역할****

|  |  |  |
| --- | --- | --- |
| 영역 | 컴포넌트 | 역할 요약 |
| 수집 | **MFT, API, 대외MCA** | 은행 및 계열사로부터 데이터 수집 (정형/비정형, 실시간/배치 포함) |
| 저장 | **빅스퀘어 / 통합 DB** | 원천데이터 저장, 가공마트/샌드박스 구성, 통합 고객번호 Key 생성 |
| 분석 | **통합분석, BI 포털, 통합 포털** | 공통분석, OLAP, 통계 SAS, 정책관리, 요청승인관리 포함 |
| 활용 | **마케팅 시스템, 올원뱅크, UMS** | 추천/이벤트/성과 정보 전파, 채널에 따른 개인화 전송 수행 |

### 📌 ****3. 주요 흐름도 기반 핵심 흐름 요약****

[계열사/은행 정보계]

↓ (MFT, API)

[통합데이터 ETL]

↓ (적재)

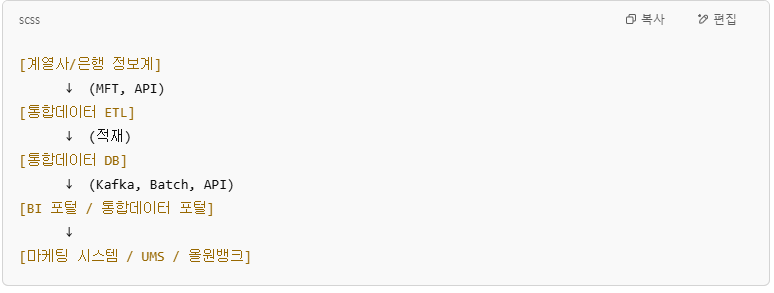
[통합데이터 DB]

↓ (Kafka, Batch, API)

[BI 포털 / 통합데이터 포털]

↓

[마케팅 시스템 / UMS / 올원뱅크]



### 📌 ****4. 아키텍처 전략적 구성 방향****

|  |  |
| --- | --- |
| 항목 | 전략 방향 |
| 데이터 수집 | 정형/비정형 및 실시간/배치 병행 수집 전략, 보안/동의 기반 분리처리 |
| 통합 DB | 은행 및 계열사 연계 고객정보의 통합 ID화 (CI 기준 통합키) |
| 분석 플랫폼 | 통합데이터 포털 + BI 포털 이중 포털 운영, OLAP + 통계 모델 동시 지원 |
| 실시간 연계 | Kafka 기반 상품추천 + API 기반 실시간 성과 전송 |
| 비동기 확산 | 마케팅 허브, UMS → MMS, E-Mail, Push 전송 구성 |
| 확장성 | 마케팅 시나리오 기반 확장 지원 (이벤트 기반 운영 구조 고려) |

# 주요 기능 정리

## 영역별 주요 기능 구성

|  |  |  |
| --- | --- | --- |
| 영역 | 주요 기능 항목 | 상세 기능 |
| **데이터 수집** | 계열사 배치 수집 실시간 수집 인터페이스 표준화 | - MFT 기반 계열사/은행 배치 수집 - API 기반 WebView 실시간 수집 - 은행 내부/외부 인터페이스 표준 준용 |
|  | 전송 구간 보호 | - 전송 구간 암호화 (HTTPS, SFTP 등) - 파일 암호화 및 암호해제 정책 - 암호화 알고리즘 및 인증서 기반 환경 구성 |
|  | 수집 파일 보관 | - 단기 보관을 위한 영역 구성 (스테이징 존) - 개인정보 포함 여부 기준 이중화 처리 |
| **데이터 저장** | 적재/통합/보호 | - 원천데이터 적재 및 목적별 마트 분리 - 통합고객번호 채번 및 데이터 연결 - 마스킹/익명화 적용 후 보호 등급 구분 |
|  | 표준화 수집 | - 계층 구조 저장 (원본, 요약, 정제, 통합 등) - 스키마 및 컬럼명 사용자 기준 정규화 |
|  | 데이터 처리 | - 삭제 대상 관리 기능 - 배치 로그 관리 - 개인정보 포함 데이터 분리 저장 |
| **데이터 분석 및 활용** | 분석 플랫폼 | - SQL/OLAP 기반 분석 환경 구성 - BI 포털 내 정형/비정형 리포트 지원 - ML/DL 모델 개발 및 서비스 연계 |
|  | 데이터 활용/제공 | - OLAP 기반 타겟 고객 예측 및 추출 - 분석결과 → 타 시스템 연계 API 지원 - 사후 결과 데이터 저장 및 추적 |
|  | 데이터 포털 기능 | - OLAP 환경 제공, 권한 및 메타관리 - 요청승인관리, 정책관리 기능 제공 - 이력조회/성과관리 통합 |
| **마케팅 시스템** | 캠페인/이벤트 실행 | - 시나리오 설계/대상선정/제외설정 - 실시간 및 자동 캠페인 실행 - Push, MMS, 이메일 전송 |
|  | 성과분석 | - 캠페인 리포트, 통계, 퍼널 분석 결과 제공 - UMS 및 올원뱅크와 실시간 연계 |

## 전략적 분석 포인트

|  |  |
| --- | --- |
| 분류 | 전략 방향 |
| 💡 **확장성 기반 수집** | 실시간 & 비동기 수집 공존 (MFT + API + Kafka) WebView 기반 마케팅 반응 데이터 실시간 수집 |
| 💡 **고객 통합 관리** | 고객 고유 키(CI) 기반 통합 저장 → 전사 CRM, 분석, 캠페인에 활용 |
| 💡 **표준화된 저장 체계** | 적재-정제-분석-보관의 4단계 데이터 흐름을 표준화 |
| 💡 **포털 통합 서비스** | OLAP + 캠페인 시나리오 + 결과 분석을 하나의 통합 포털에서 제공 |
| 💡 **보안 및 개인정보** | 수집–저장–전송 구간별 암호화 및 삭제, 분리저장 체계 수립 |
| 💡 **실행 최적화** | 통합된 타겟 관리 및 마케팅 실행 자동화 체계 수립 (UMS/올원뱅크 포함) |

## NH 농협지주 통합 데이터 수집 기능 체계 정리

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 1 | Level 2 | 상세 기능 설명 | 전략적 의미 / 비고 |
| **데이터 수집** | 데이터 수집 | 계열사 데이터 배치 수집 | - 제3자 제공 동의 고객 대상 데이터 수집 - 1단계: 은행, 카드, 증권, 생명, 손해 - 2단계: 캐피탈, 저축은행 등 - 은행 외 계열사 고객은 가맹고객으로 분류하여 별도 수집 | 고객 범주별 분리 수집 정책으로 보안/권한 통제 강화 가능 |
|  |  | 계열사 실시간 수집 (마케팅 동의 여부) | - 최초 수집 이후 동의 여부로 실시간 수집 반영 - D+1 일 배치 처리 방식 고려 - 실시간 반영 시 마케팅 활용 연계 | 마케팅 즉시성 요구 충족 위한 실시간 동기화 체계 필요 |
|  |  | 올원뱅크 Web View 실시간 수집 (행태 데이터) | - 고객의 Web 페이지 이동, 클릭 등 행동 데이터 수집 | 채널 내 비정형 사용자 행위 분석 고도화 가능 |
|  |  | 은행 내/외부 표준 인터페이스 준용 | - 계열사 연계를 위한 내부 표준 연동 방식 도입 | API 연계에 대한 보안 인증/로그 체계 연계 필요 |
|  |  | 전송 구간 암호화 | - 전송 시 구간 암호화 (SSL/TLS 등 은행 표준) 적용 | 데이터 유출 방지, 금융보안 대응 |
|  |  | 전송 파일 암호화 | - 파일 단위 암호화 (AES256 등) 및 무결성 검증 적용 | Batch/MFT 구간의 안정성 및 법적 요건 충족 |
|  |  | 수집 파일 암호화 해제/변환 등 환경 구성 | - 암호화 모듈 도입, 수신 즉시 복호화 및 전처리 기능 | ETL/적재 단계로 자동화 연계 필요 |
|  |  | 수집 파일 단기 보관 영역 구성 | - 보관정책: 월파일 기준 3개월 후 삭제 - 장기보관 필요시: 특정 폴더 7~14일 랜딩 저장 - MFT Local 디렉터리 내 보관 필요 | 복구, 재전처리 및 이중 검증 시 필요한 이력 확보 전략 |

### 아키텍처 연계 전략 요소

* **MFT → ETL 흐름 제어**: 수집 서버 내 전용 스테이징 영역 구성 후, ETL에서 정책 기반 이동 처리 (파일명, 폴더 기준)
* **실시간 수집 연계 (WebView/CI기반)**: API Gateway + Kafka + DB-Trigger 조합으로 실시간 데이터 반영 체계 구성
* **보안 연계 정책 (파일/통신)**:
  + 암호화 방식: AES256 + RSA Key 기반 이중 보안
  + 통신 채널: HTTPS(SNI)/TLS 1.3 기반 표준
  + 무결성 검증: SHA256 + HMAC 방식

### 도식 구성 (텍스트 기반)

[계열사/은행 정보계/계정계 시스템]

│

├─[배치 데이터]────┐

│ │

│ [MFT] ───→ [수집 서버]

│ ├─ 암호화 해제

│ └─ 스테이징 폴더 저장

│

└─[실시간 API 연계] ─→ [API GW] ─→ [Kafka] ─→ [실시간 적재서버]

[수집 서버] ─→ [ETL 적재] ─→ [통합 데이터 DB / 분석 마트]

## NH 농협지주 데이터 저장 기능 상세 정의

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 1 | Level 2 | 상세 설명 | 전략적 의미 / 비고 |
| **데이터 저장** | 원천 데이터 적재 및 목적별 마트 개발 | - 은행/계열사로부터 수집한 데이터 DB 적재 - 분석, 마케팅 용도의 마트(Mart) 구성 | 목적별로 사전 정의된 스키마로 분리되어 적재되어야 하며, DW/DM 전략과 연계됨 |  |
|  | 통합고객번호 채번 | - 최초 적재 시 CI 기준으로 동일 고객을 식별하여 채번 - 매핑 불가 시 신규 통합고객번호 부여 | 고객 360 분석, 통합 연계/분석의 핵심 인덱싱 체계 기반 |  |
|  | 데이터 저장 레이어 표준 수립 (원본, 정제/통합, 요약/분석) | - **L0 (Raw)**: 계열사 원천 데이터 수집 영역 - **L1 (Refined)**: 정제 후 통합 데이터 저장 영역 - **L2 (Summary)**: 통계/분석 목적 요약 데이터 | DW 3계층 구조 기반 데이터 신뢰성과 활용성 확보 |  |
|  | 정보제공 필수/선택에 따른 고객 데이터 분리/권한관리 | - 개인정보 유무에 따른 테이블 스키마/암호화/권한 분리 운영 - CI, 고객번호 등은 암호화/분리 필요 | 개인정보보호법 및 금융보안원 기준 대응 구조 |  |
|  | 분석(SI/사용자)용 DB 구성 | - 빅스퀘어 분석 후 결과 적재용 DB 구성 - 마케팅 성과 분석 결과도 별도 적재 | 실시간 결과 공유 및 타 시스템 연계 시 일관된 분석 환경 유지 |  |

### 레이어 구조 도식 (텍스트 기반)

[데이터 저장 3계층 구조]

1. L0 (Raw Data Layer)

└ 계열사별 원천 DB → 수집 그대로 저장

└ 무결성 체크, 적재 시점만 관리

2. L1 (Refined / 통합 Layer)

└ 중복 제거, 공통컬럼 표준화

└ 통합고객번호 기준으로 매핑 처리

└ 정제 Rule 적용 후 통합 저장

3. L2 (Summary / Insight Layer)

└ 마케팅, 분석, KPI, 시각화용 요약 DB

└ 일정 배치 주기로 Snapshot 구성

### 보안/권한 통제 전략 요약

|  |  |
| --- | --- |
| 항목 | 내용 |
| 개인정보 분리 | 암호화 후 별도 스키마 저장, 암호화 Key는 KeyVault에 저장 |
| 권한 분리 | 개인정보 열람은 관리자 그룹만 접근 가능, 읽기/쓰기 권한 분리 |
| 접근 제어 | DB 사용자 권한 분리 (ETL vs 분석 vs 일반사용자) |

## NH 농협지주 데이터 저장 기능 상세 정의 (2/3 반영)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 1 | Level 2 | 상세 설명 | 전략적 의미 / 비고 |
| **데이터 저장** | 스키마 및 테이블에 대한 사용자/그룹별 인증·접근 제어 표준 | - 계열사 통합데이터 접근 권한 정책 수립 - AP, ETL, 조회용 별도 계정 분리 사용 - AP: CRUD, 필요 시 개인정보 영역 제한적 접근 - ETL: CRUD, TRUNCATE, INDEX REBUILD 등 배치에 필요한 권한 부여 - 조회: SELECT 전용 계정 (BI 사용자) | 계열사별 Data Access 분리 및 사용자 그룹별 오라클 ROLE 설정을 통한 접근통제 강화 |  |
|  | 데이터 보관 주기 만료에 따른 삭제 기능 | - 계열사 고객 데이터 보관 주기 만료 시 고객정보 삭제 모듈 구성 - 내규 혹은 보안지침 기반 보관주기 정책 설정 가능 (예: 매일 삭제 후 재수신) - 예: 일반 데이터와 개인정보 테이블 구분 삭제 정책 | **개인정보 보호법**, 금융보안원 지침 등 데이터 라이프사이클 관리(LCM) 요구사항 대응 |  |

### 권한 분리 정책 예시 (Oracle DB 기준)

|  |  |  |
| --- | --- | --- |
| 역할 구분 | 권한 유형 | 권한 내용 |
| **ETL 계정** | CREATE, UPDATE, TRUNCATE, REBUILD INDEX, MERGE, GRANT | 대용량 적재/정제/정리 |
| **BI 계정** | SELECT ONLY, 제한된 View 접근 | OLAP 조회 전용 |
| **AP 계정** | SELECT, INSERT, (부분 UPDATE) | 서비스 중간계 계정 |
| **관리자** | Full DBA 권한 (운영자 전용) | 정책 변경/모니터링/사용자 관리 전용 |

### 논리적 구조 내 역할 구분도 (텍스트 기반)

┌──────────────────────────────┐

│ [데이터 저장구간 : Oracle] │

├────────────┬─────────────┤

│ ETL 계정 │ AP 계정 │

│ └ L0, L1, L2 적재 │ └ 연계용 뷰 조회/삽입 │

│ └ 데이터 삭제 관리 │ └ CI 매핑 조회 │

├────────────┴─────────────┤

│ BI 계정 (조회 전용) │

│ └ OLAP, BI 포털용 쿼리 수행 │

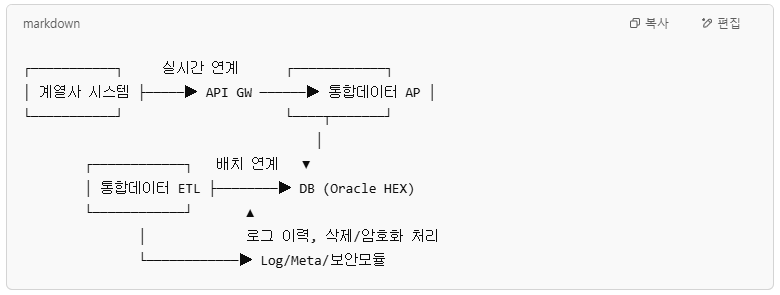
│ └ 개인정보 View 제외 or Masking │

|  |
| --- |
|  |

## NH 농협지주 데이터 저장 및 처리 기능 정리 (3/3 기반)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 설계 구현 전략 | 전략적 의미 |
| 데이터 처리 | 파기 대상 데이터 삭제 기능 | - 고객 철회 요청 또는 제3자 제공 철회 시 대상 고객 데이터 삭제 처리 - 파티션 테이블은 메타에서 직접 삭제 가능 - 비파티션 테이블은 전용 삭제 모듈 또는 적재 ETL 내장 처리 필요 | - 고객 ID 기준 삭제 처리 자동화 - 파기 이력 로깅 체계화 - 메타 연동으로 자동 파기 대상 조회 | 개인정보 보호법, 금융보안 규정 대응 및 고객신뢰도 확보 |
| 데이터 처리 | 정기 배치 작업 이력 로깅 | - 정기 배치 수행 내역 스케줄러 기반 관리 - 성공/실패 결과 이력 보관 및 모니터링 필요 | - 통합 스케줄러(예: Control-M, Airflow) 연동 - 로그 파일 → 통합 Log 관리 플랫폼 연계 | 배치 Job 품질 확보, 장애 추적 및 감사 대응력 강화 |
| 데이터 처리 | 개인정보 포함 필드 암호화/마스킹 | - DB 저장 시 암호화 (AES-256, Oracle TDE) - UI 출력 시 마스킹 적용 | - 민감 필드 자동 마스킹 정책 (이름, CI, 생년월일 등) - 계열사별 조회권한 분리 설정 | 보안 취약점 방어, 비인가 접근 차단, 정보 유출 예방 |
| 데이터 처리 | 타 시스템 실시간/배치 연계 서비스 | - 실시간 연계: API GW 기반 표준 REST/JSON 방식 - 외부 연계: 대외 MCA, FOS 기반 MFT 방식 - 배치 연계: MFT 또는 CDC/ETL 방식 지원 | - Webhook/Message Broker 연계도 병행 고려 (Kafka 등) - 연계 결과 로깅 및 재처리 모듈 구축 | 유연한 계열사 연계 확장성, 실패 복구 전략 내재화, 재처리 생산성 향상 |

### 아키텍처 연계 흐름 간략 도식 (텍스트 기반)

┌───────────┐ 실시간 연계 ┌────────────┐

│ 계열사 시스템 ├─────▶ API GW ──────▶ 통합데이터 AP │

└───────────┘ └────┬───────┘

│

┌────────────┐ 배치 연계 ▼

│ 통합데이터 ETL ├────────▶ DB (Oracle HEX)

└────────────┘ ▲

│ 로그 이력, 삭제/암호화 처리

└────────────▶ Log/Meta/보안모듈

### 정책 연계 요약

|  |  |  |
| --- | --- | --- |
| 항목 | 기준 정책 | 구현 기술 |
| 데이터 삭제 | 파기대상 목록 기반 삭제 프로세스 | Oracle Partition Drop + Python/Delete API |
| 배치 로깅 | 배치 스케줄 로그 수집 및 모니터링 | Prometheus + Grafana + Logstash |
| 개인정보 암호화 | 저장 시 암호화, 출력 시 마스킹 | Oracle TDE, SHA-256, UI Masking |
| 연계 인터페이스 | 실시간/API, 비동기/MFT 병행 | Spring Cloud Gateway + MFT (FOS) |

## ✅ NH 농협지주 데이터 분석 및 활용 기능 정의 (1/3 기반 정리)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 2 | 주요 내용 | 설계 구현 전략 | 전략적 의미 |
| 데이터 분석 | 쿼리 기반 분석 프로세스 개발 | - 은행 내부 쿼리 툴로 통합데이터 접근 분석 - 계열사는 보안 및 계정 이슈로 직접 쿼리 사용 불가 | - 은행은 BI 툴, SQL 기반 분석 환경 구성 - 계열사는 통합데이터 포털을 통해 제한적 조회 | 내부 데이터 분석 효율화 및 계열사 보안 분리 운영 |
| 데이터 분석 | ML/DL 모델 개발 | - 은행: 빅스퀘어 기반 모델 개발 가능 - 계열사: 미허용 (계정/접근 제한) | - ML Studio, Python, R 기반 환경 구성 - 통합데이터 모델링 셋과 연계 | 머신러닝 기반 예측 및 고객 리스크 탐지 확대 |
| 데이터 분석 | 상품추천 모델 확장 | - 통합데이터의 상품추천용 데이터 I/F - 빅스퀘어에서 고도화 또는 신규 모델 필요 | - 통합데이터 연동 후 빅스퀘어로 전송 - 모델 연산, 결과를 통합데이터로 다시 I/F | 그룹 차원의 Cross-Sell 기반 모델 확장 |
| 데이터 분석 | 분석 모델 운영 | - 빅스퀘어에서 모델 운영 수행 - 결과 데이터 통합데이터에 반영 가능 | - 결과 마트 구성, 모델 결과 이력 관리 - UMS/마케팅과 연계 가능한 포맷 설계 | 분석-마케팅 연계 자동화 및 활용도 증대 |

### 📌 설계 기준 요약

|  |  |  |
| --- | --- | --- |
| 항목 | 은행 | 계열사 |
| 쿼리 기반 분석 | BI툴, SQL 접근 가능 | 포털 통한 조회 UI |
| ML/DL 분석 | 빅스퀘어 기반 모델링 허용 | 불가 (보안 분리 및 계정 미제공) |
| 상품추천 분석 | 고도화 모델 연계 가능 | 계열사 전용 모델은 별도 개발 필요 |
| 모델 결과 활용 | OLAP 및 분석마트 반영 | BI포털로 결과 통합 조회 |

### 🔧 분석 기능 연계 구성도 (텍스트 기반)

css

복사편집

[통합데이터 저장소]

│

├─▶ [분석마트: BI, 상품추천, 고객세분화]

│ └─▶ 분석툴 (SQL, BI Dashboard)

│

├─▶ [빅스퀘어: ML/DL 모델 개발]

│ └─▶ Python, R, SAS, Spark

│

└─▶ [분석결과 → 통합데이터 포털]

└─▶ UMS / 마케팅 시스템 연계 (API, Batch)



## ✅ NH 농협지주 데이터 분석 및 활용 기능 정의 (2/3 기반 정리)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 2 | 주요 내용 | 설계 구현 전략 | 전략적 의미 |
| 데이터 활용/제공 | OLAP을 통한 타깃 고객 리스트 추출 및 제공 | - OLAP 분석 기반으로 마케팅 타깃 고객 리스트 도출 - 마케팅허브/e-마케팅 시스템으로 연계 | - OLAP 분석 결과를 Kafka 또는 Batch로 마케팅 시스템에 I/F - BI 포털 내 OLAP 대시보드 구성 | 개인화 마케팅 강화 및 분석–실행 자동화 체계 확보 |
| 데이터 활용/제공 | 분석환경에서 통합데이터를 조회/처리 기능 | - 은행: OLAP 포털 또는 빅스퀘어에서 통합데이터 직접 연결 분석 - 계열사: OLAP 포털을 통해 분석 환경 제공 | - BI 포털 기반의 OLAP 접근 권한 구분 - 계열사/은행 구분된 URI/접속 정책 적용 | 기관별 분석 자율성 확보와 보안 분리 동시에 실현 |

### 📌 기술 아키텍처 구성 요약

|  |  |  |
| --- | --- | --- |
| 항목 | 은행 | 계열사 |
| 분석도구 | BI 포털 OLAP, SAS, Sandbox 등 | BI 포털 OLAP |
| 분석 데이터 접근 | 통합데이터 직접 연결 | 통합 포털 연계 경유 |
| 마케팅 연계 | Kafka, API, Batch 연계 (성과/반응/추천 등) | 간접 연계 및 조회 전용 |
| 활용 데이터 유형 | 분석마트, 추천모델 결과, 통합고객번호 | 조회 기반 분석데이터 |

### 🔧 데이터 활용 흐름 구성도 (텍스트 기반)

css

복사편집

[통합데이터 DB]

└─▶ [OLAP 분석 마트]

├─▶ [BI 포털 대시보드: OLAP]

│ └─▶ 계열사 조회 (권한 기반 제한)

│ └─▶ 은행 분석자 직접 조회

└─▶ [Kafka or Batch Export]

└─▶ 마케팅 허브 / e-마케팅 시스템 (API 연계)



### 향후 고려 사항 및 설계 방향

|  |  |
| --- | --- |
| 항목 | 고려 사항 |
| OLAP 프로젝트 별도 분리 | OLAP 마트는 분석 주제/부서별로 구분하여 I/F 구성 필요 |
| 마케팅 시스템 연계 I/F | Kafka, REST API 등 실시간/비동기 메시징 기반 설계 권장 |
| 데이터 접근 권한 분리 | 은행/계열사 사용자의 OLAP 조회 및 분석 권한 차등 부여 |
| 보안정책 수립 | 계열사 접속 경로(URL), 로그, 권한 정책 등 별도 정책 수립 필요 |

## ✅ NH 농협지주 데이터 분석 포털 구성 전략

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 기능 세부 항목 | 설명 | 설계 전략 | 전략적 의미 |
| **OLAP 분석 환경 제공** | OLAP 분석 환경 제공 | 통합데이터 기반 정형/비정형 분석 보고서 작성 타깃 리스트 추출 결과는 마케팅 시스템에 연계 | - 기존 OLAP 툴 연계 - BI 포털에서 분석 환경 제공 (Cloud 기반) | 마케팅과 분석 연계를 통한 전사 전략 기반 의사결정 체계 |
| **메타데이터 관리** | 메타 데이터 등록/조회 | 업무/표준/코드 메타정보 등록 및 승인, 추가정보 등록 | - IT메타 연동 자동화 - Biz메타-업무 메타 분리 등록 체계 구축 | 데이터 신뢰성과 표준화 확보, 분산 메타 유지 최소화 |
| **품질관리 체계** | 데이터 품질 등록 및 점검 | 품질기준 등록 및 오류이력 관리 | - 품질기준 자동 점검 - ETL 또는 분석 시점 품질로그 집계 | 데이터 활용 정확도 확보 및 통합품질 관리체계 정립 |
| **권한 및 사용자 관리** | 사용자 계정/권한 통제 | 은행/계열사별 계정 통제, 메뉴 접근 권한 별도 적용 | - 메뉴권한 템플릿 정의 - 접속 URL/계정 이중 맵핑 처리 | 정보보호 및 접근통제 정책 대응 (ISMS-P, 내부통제) |
| **마케팅 커뮤니케이션** | 마케팅 기획 게시판 | 마케팅 타깃 시나리오, 배너 내용, 컨텐츠 설계 가이드 등 게시 | - UMS/캠페인/BI포털 간 KPI 연계 - 마케팅 UX 커뮤니케이션 정형화 | 부서 간 협업 효율성과 배너 효과 분석 가능 |
| **성과 분석 결과 제공** | 반응 분석 리포트 | 고객별 클릭/행동 이력 분석, 타깃 반응 결과 제공 | - 배너별 ID 추적 - OLAP 분석을 통한 반응 리포트 생성 | 고객 행동기반 분석 고도화 및 개인화 마케팅 지원 |

### 포털 활용 기능 흐름 구성도 (텍스트 기반)

[통합 데이터 저장소]

└─▶ [OLAP 분석 마트]

├─▶ [BI 포털]

│ ├─ OLAP 분석 환경

│ ├─ 사용자별 권한 관리

│ ├─ 메타 조회/등록

│ ├─ 마케팅 기획 게시판

│ └─ 반응분석 리포트 제공

└─▶ [마케팅 시스템 (EBM/e-마케팅)]

├─ 배너 클릭 결과 수신

├─ 고객 반응 분석 요청

└─ 캠페인 실행 / UMS 전송



### 기술 구성 및 고려 요소 요약

|  |  |  |
| --- | --- | --- |
| 항목 | 구성 기술 | 비고 |
| 분석 포털 인프라 | PaaS 기반 Cloud BI 포털 (분리형 Pod 구성) | 계열사/은행 분리 운영 |
| 메타 통합 연계 | IT메타 (용어/컬럼 등) + Biz메타 (업무정의) | 표준 등록/조회 기능 내장 |
| 품질 점검 체계 | ETL 품질 로그 + 수기 품질 등록 | 포털 내 품질점검 탭 제공 |
| 마케팅 연계 | Kafka, REST API, Batch | BI 분석 결과를 마케팅 시스템과 연결 |
| 보안 및 권한 | Role 기반 RBAC, 사용자별 접근제어 정책 | 내부통제 대응체계 |

## ✅ NH 농협지주 마케팅 시스템 설계 요소 요약

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | Level 2 기능 | 주요 내용 | 설계/연계 전략 | 전략적 의미 |
| **캠페인 설정** | 개인화 마케팅 시나리오 작성 | 요청 기반 계열사 마케팅 실행정보를 은행 마케팅 시스템에 등록 | - 시나리오별 캠페인 요청/승인 Workflow 연계 필요 - 통합데이터 기준 이력관리 구조 도입 | 계열사 자율성과 통제권 조화를 위한 구조 |
|  | 마케팅 대상/제외자 선정 | BI 포털 OLAP로 추출된 타깃을 마케팅 시스템으로 직접 전송 | - OLAP 결과를 Kafka 또는 Batch API로 실시간 전송 - 타깃 기준 저장 시 통합 고객번호 기반 필요 | 정확한 대상 관리 및 배포 최적화 |
|  | 피로도 정책 설정 | 노출 빈도 및 기간 조절 정책 등록 | - 프로파일 기반 실시간 피로도 체크 - 개인화 룰 기반 노출 제어 로직 필요 | 고객 반응도 기반 효율적 노출 제어 |
|  | 캠페인 프로모션 처리 및 타깃 | 등록된 캠페인과 매칭하여 대상·A/B 그룹 설정 | - BI 포털 타깃 정보 + 캠페인매핑관리 DB 활용 | 반복 마케팅 효율 분석 가능 |
| **캠페인 실행** | 실시간 서비스 지원 | 실시간 데이터 수신 후 이벤트 트리거 → 메시지 발송 | - Kafka 기반 실시간 메시징 처리 - 실시간 API 호출 구조 필요 | 이벤트 기반 실시간 마케팅 대응 |
|  | 자동화 서비스 지원 | 데이터 적재 후 자동 트리거 실행 | - 통합데이터 기준 조건 → API → 캠페인 실행 - 조건 관리 룰엔진 필요 (예: Drools, RuleBook 등) | 지속 가능한 자동 타깃 대응 구조 |
|  | 개인화 마케팅 실행 | 개인별 ID 기준 맞춤 메시지 발송 및 실시간 응답 추적 | - 배너/컨텐츠 ID 기준 반응 추적 DB 연계 - 고객별 메시지 전환 경로 기록 필요 | 옴니채널 기반의 고도화된 맞춤 마케팅 |
| **이벤트 실행** | 이벤트 설계 및 실행 | UMS 및 앱 배너 노출 기반 이벤트 설계 및 이력 관리 | - 랜딩 이력과 함께 설계 - 이벤트 반응 수집 및 BI 포털 전달 | 배너/이벤트 효과 분석을 위한 폐쇄 루프 구현 |

### 📌 시스템 간 연계 흐름 요약 (텍스트 기반 구성도)

css

복사편집

[BI 포털 OLAP 분석]

└─ 타깃 고객 리스트 생성

└─▶ Kafka / Batch API

└─▶ [은행 마케팅 시스템]

├─ 타깃 캠페인 매핑

├─ 피로도 정책 검증

├─ 캠페인 실행 (App, SMS, Push)

└─ 반응결과 저장 및 BI 포털 전달



### 🛠️ 기술 요소 및 도입 전략

|  |  |  |
| --- | --- | --- |
| 항목 | 도입 기술 | 연계 포인트 |
| 타깃 전송 | Kafka, REST API, Batch API | BI 포털 → 마케팅 시스템 |
| 실시간 실행 | Kafka Stream, EventTrigger | 캠페인 조건 충족 시 즉시 실행 |
| 피로도 관리 | Rule Engine (Drools 등) | 개인별 노출횟수 기준 평가 |
| 시나리오 작성 | Workflow + UMS 연계 | 계열사 → 중앙 실행 요청 흐름 |
| 반응 수집 | 배너ID + 고객ID 기반 DB | UMS + 분석용 DB 연계 |
| 이력 관리 | 통합 로그 저장소, Audit Trail | 보안 감사 및 내부통제 대응 |

## NH 농협지주 마케팅 성과 분석 체계 요약

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 기능 | 주요 설명 | 통합 연계 전략 | 전략적 의미 |
| **성과 분석** | 과거 이력 관리 및 성과 관리 | 계열사 마케팅 실행 이력 및 결과 저장·분석 | - 마케팅 시스템의 캠페인 로그를 통합데이터 포털로 전송 - 성과 이력 기준 타겟 세분화 가능 | 반복 마케팅 성과 추적 기반 고도 분석 가능 |
|  | 오퍼/시나리오별 성과 분석 | 각 캠페인별 오퍼(A/B/n) 및 시나리오별 효과 비교 분석 | - 마케팅 시스템 → 통합데이터로 결과 연계 - KPI 중심 통계 기반 대시보드 구성 | 콘텐츠/전략별 ROI 분석 및 최적화 |
|  | 퍼널 분석 결과 제공 | 고객 유입–노출–클릭–전환 단계별 흐름 분석 | - BI 포털 내 퍼널 분석 대시보드 개발 - 이벤트 클릭정보 + 노출 컨텐츠 ID 기준 통합 | 채널별 반응 경로 시각화 및 개선 가이드 제공 |

### 🧩 아키텍처 연계 흐름도 (텍스트 기반)

scss

복사편집

[캠페인 실행]

└─ 고객 ID + 컨텐츠 ID + 채널 정보 + 반응 결과 (클릭/전환/미응답)

└─▶ [은행 마케팅 시스템] → 로그 적재

└─▶ 통합데이터 플랫폼 수집

└─▶ BI 포털 (성과 대시보드 / 퍼널 분석)



### 🛠️ 연계 기술 및 컴포넌트 대응

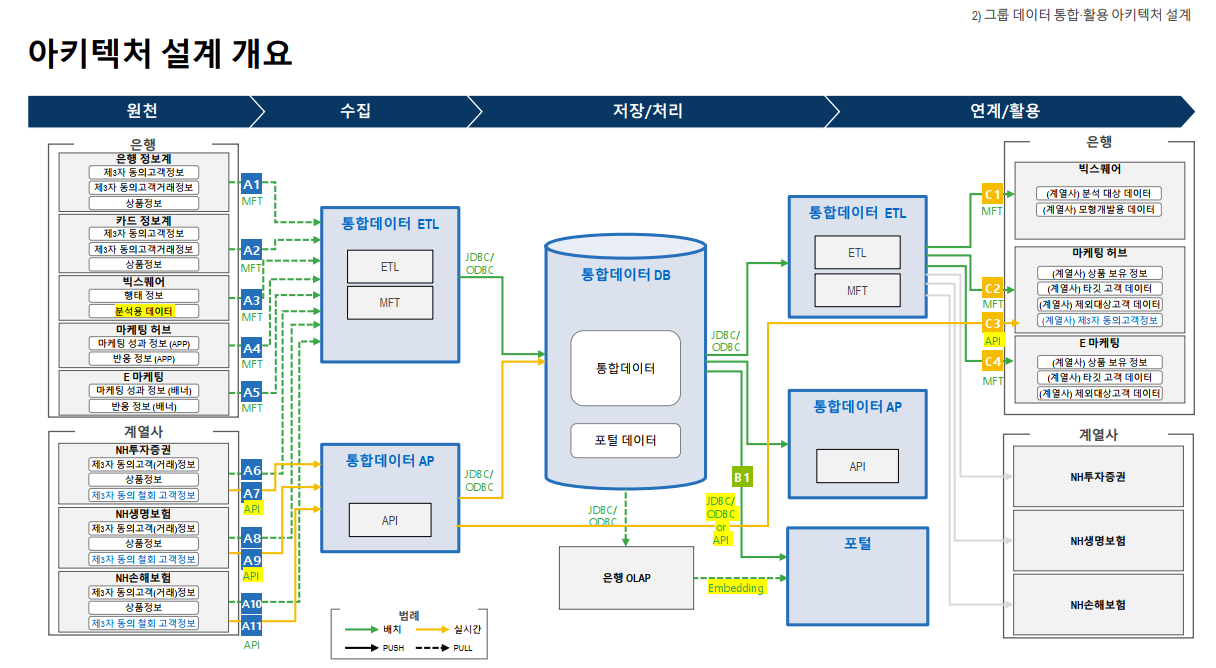
|  |  |  |  |
| --- | --- | --- | --- |
| 연계 대상 | 기술 스택 | 연계 구조 | 설명 |
| 마케팅 시스템 → 포털 | Kafka, REST API, Batch | 비정형 로그 전달 | 실시간 이벤트 결과 또는 정기 집계 연동 |
| 성과분석 DB | Oracle / Hive / DWH | BI 포털 OLAP 또는 정형통계 시스템 | 클릭, 전환률, 노출 대비 반응률 통계 처리 |
| 퍼널 분석 엔진 | Grafana / Metabase / Tableau | BI 연동 대시보드 구성 | 단계별 지표 분해: 노출→클릭→상담/구매 |
| 이벤트 로그 수집 | UMS Log, Web Click Log, App 이벤트 | DB + 로그 수집 | 이벤트 ID 기반 고객 반응 전체 추적 |
| KPI 분석 기준 | 캠페인코드, 타깃코드, 컨텐츠 ID | 통합 기준정보 연계 | 정규화된 마케팅 결과 분석을 위한 표준코드 설정 필요 |

### 📌 비즈니스 전략 관점 요약

|  |  |
| --- | --- |
| 전략 항목 | 설명 |
| 🎯 정밀 성과 분석 | 통합 포털에서 고객별 성과 이력 + 캠페인 반응 연계 |
| 🧭 캠페인 전략 조정 | A/B 테스트 결과 기반으로 최적 채널/컨텐츠 도출 |
| 🔄 자동화 흐름 분석 | 퍼널 기반 자동 최적화 모델링 적용 가능 |
| 🔐 보안 및 로그 규정 | 캠페인별 반응정보는 개인정보 보호 기준에 따라 익명화 또는 제한 조회 필요 |

# 핵심 아키텍처 전략

## 아키텍처 전체 개요



### 아키텍처 설계 테스트 기반 구성도

|  |
| --- |
| 1. [원천 데이터 소스: 은행/계열사]  └─ 은행 정보계  ├─ 제3자 동의 고객정보  ├─ 제4자 동의 고객정보  ├─ 상품정보, 거래정보  └─ 카드 정보계  └─ 제3자 동의 고객정보, 거래정보  └─ 분석성 데이터 (행태정보)  └─ 마케팅 허브 (성과정보, 반응정보)  └─ e-마케팅 (배너/팝업 클릭정보)  └─ 계열사 (증권/생명/손보)  └─ 제3자 고객정보 / 철회정보  2. [데이터 수집 Layer]  ├─ 📦 통합데이터 ETL 서버  ├─ (MFT) A1 ~ A5: 은행/카드/마케팅 허브/이벤트 배치 전송  ├─ ETL 처리 모듈 포함  └─ 🔌 통합데이터 AP 서버  ├─ (API + MFT) A6 ~ A11: 계열사 데이터 수집  ├─ 실시간 고객정보 요청/철회 대응 (MCA 연계)  3. [데이터 저장 Layer]  └─ 🗄️ 통합데이터 DB  ├─ 통합데이터 저장소 (고객, 거래, 상품, 공통정보)  ├─ 포털 데이터 저장소 (UI 분석 요청, 정책관리 등)  4. [데이터 활용/연계 Layer]  ├─ 📊 통합데이터 ETL 서버 (연계용)  ├─ C1: 빅스퀘어 분석 데이터 연계 (MFT)  ├─ C2: 마케팅 허브 상품/반응 데이터 연계  ├─ C3: 마케팅 타깃 고객 연계 (API)  └─ C4: e-마케팅 배너 반응 연계 (MFT)  ├─ 🔌 통합데이터 AP 서버 (API)  └─ 실시간 고객정보 제공용 API 서비스  ├─ 🧠 은행 OLAP  └─ B1: 통합데이터 기반 분석 시스템 (JDBC/ODBC/Embedding)  └─ 🌐 통합 포털  └─ 사용자 요청 인터페이스, 대시보드, 정책관리, 결과조회 등 |
|  |

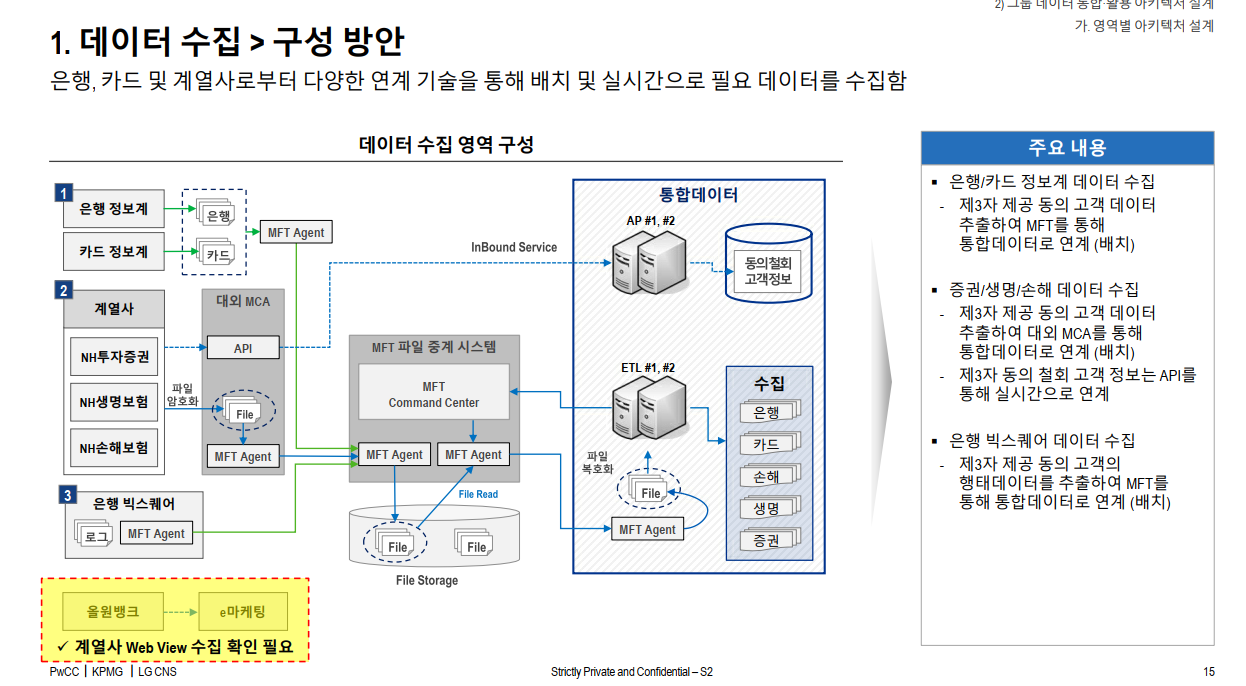
### 🔄 연계 유형별 흐름 구분

|  |  |  |  |
| --- | --- | --- | --- |
| 연계 방식 | 사용 예시 | 구분 | 적용 위치 |
| **MFT (배치 전송)** | 비정형 대용량 파일 | 비정기 / 예약 전송 | A1~A5, A6, C1, C2, C4 |
| **API (실시간 호출)** | 고객정보 조회/철회 | 실시간 서비스 | A7~A11, C3 |
| **ETL (정형 처리)** | 정제/적재 | 배치 처리 | 통합 ETL |
| **JDBC/ODBC** | OLAP / BI 분석 | 직접 질의 | OLAP ↔ DB / 포털 ↔ DB |

### 🧭 전략적 시사점 요약

|  |  |
| --- | --- |
| 항목 | 내용 |
| 고객 식별 | 모든 연계 흐름의 기준은 **통합 고객번호 (CI)** 기준 |
| 마케팅 연계 | 실시간 추천은 Kafka 연계 가능성 내포 (별도 도입 필요 시 명시) |
| 분석 연계 | OLAP 또는 빅스퀘어, BI 포털은 모두 JDBC 기반 직접 연계 |
| 계열사 확장 | MCA 기반 API 및 MFT 연계 구조로 확장 가능성 확보 |
| 포털 연동 | 통합 포털은 OLAP / ETL / AP 서비스 모두 참조 가능해야 함 (Embedding 고려) |

## 데이터 수집 아키텍처 구성방안



### 텍스트 기반 데이터 수집 아키텍처 구성도

|  |
| --- |
| [1] 은행/카드 정보계 수집 (배치 - MFT 방식)  └─ 은행 정보계  └─ 카드 정보계  └→ MFT Agent (은행/카드 측)  └→ MFT 파일 중계 시스템 (MFT Command Center)  └→ MFT Agent (통합 데이터 ETL 서버)  └→ File Storage → ETL 처리 → 통합 데이터 DB 적재  [2] 계열사 수집 (배치 및 실시간 혼합 방식)  ├─ NH투자증권, NH생명보험, NH손해보험  │ ├─ 제3자 제공 고객정보 → MFT 전송  │ ├─ 철회정보 등 고객정보 → 대외 MCA 연계 API 호출  │ └→ 통합 데이터 AP 서버 수신 (API + MFT Agent 수신)  │ └→ 통합 데이터 DB 적재  [3] 은행 빅스퀘어 수집 (배치)  └─ 빅스퀘어 로그 (행태 데이터)  └→ MFT Agent → MFT 중계 → 통합데이터 ETL 서버  [4] 통합 수신 인프라 (통합 영역)  ├─ ETL 서버 (ETL #1, #2)  │ └→ 파일 수신 및 복호화  │ └→ 수집 디렉토리 구분: 은행 / 카드 / 손해 / 생명 / 증권  │ └→ DB 적재  └─ AP 서버 (AP #1, #2)  └→ 실시간 Inbound API 수신  └→ 동의/철회 고객정보 처리  └→ DB 적재  [5] 기타 수집 고려 항목  └─ 올원뱅크 · e마케팅 Web View 수집 필요  └→ (고객 Web 행동 로그: 클릭/뷰/스크롤 등)  └→ 추가 수집 모듈 구성 필요 |
|  |

### 주요 내용 요약 (이미지 기반 해석)

|  |  |
| --- | --- |
| 항목 | 설명 |
| **은행/카드** | 배치 기반 MFT 전송 (제3자 제공 고객/거래/상품정보) |
| **계열사** | MCA API 호출 기반 실시간 + MFT 배치 동시 구성 |
| **빅스퀘어** | MFT 기반 로그성 데이터 전송 (행태 분석용) |
| **WebView 로그** | 올원뱅크 / e마케팅에서 발생하는 사용자 Web 행동 수집 필요 (현재 수집 여부 미정) |
| **중계 시스템** | 중앙 MFT Command Center를 중심으로 통합 수신체계 구성 |
| **저장 위치** | 통합 데이터 DB 내부의 ‘수집 디렉토리’ 또는 ‘포털데이터’ 구분 저장 |

### 전략적 시사점

|  |  |
| --- | --- |
| 구분 | 전략적 고려사항 |
| 전송 방식 이중화 | 계열사 등 외부 기관은 MFT와 API를 병행 사용하여 데이터 신뢰성과 실시간성 확보 |
| Web 행동 수집 | e-마케팅 성과/배너 클릭 등 개인화 서비스 고도화를 위한 WebView 수집 체계 신규 필요 |
| MFT 보안 정책 | 파일 암호화/복호화, 저장지 보관정책(보관주기 3개월), 전용 폴더 구성 필요 |
| 수집 로그 추적 | 수신/복호화/적재 로그 이력 관리를 위한 통합 Logging 시스템 필요 (ETL/Command Center 단위) |

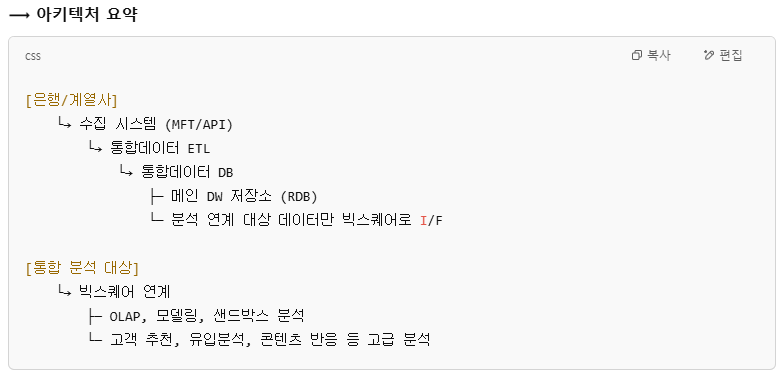
## 데이터 적재 아키텍처 구성방안



### 통합데이터 수집 및 적재 방안 비교 요약표

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | [1안] 은행 DW 활용 | [2안] 은행 빅데이터 플랫폼 활용 | [3안] 은행 빅데이터 내 별도 AP 추가 구성 (**추천안**) |
| **구현 구조** | ADW/ RDW 내 통합 적재 | 기존 빅스퀘어에 통합 적재 | 기존 빅스퀘어 + 별도 AP 구성 (RDB 구조) |
| **수집 대상** | 은행/카드 + 생명/손해/증권 고객 | 은행/카드 + 생명/손해/증권 고객 | 은행/카드 + 생명/손해/증권 고객 |
| **저장소 타입** | RDB (정형 DW) | Hadoop 기반 파일 시스템 | RDB (AP 내 DB) + 파일 분석 연계 |
| **특징** | - 경영/리스크/보고용 통합 DW 적재 - 분석/보고 중심 | - 다양한 분석데이터 적재 가능 - 비정형 기반 처리 가능 | - API 기반 실시간 업무 대응 구조 - 실시간 철회/삭제 이력 관리 |
| **고려사항** | - DW 의존성 높음 - 마케팅 시스템에 직접 영향 - DR 구성 필요 | - 실시간 고객 데이터 관리 불가 - 고객 Profile 일부 불완전 | - 분석 대상만 빅데이터 I/F - 나머지 정보는 RDB에서 처리 - 보안 영역 격리 용이 |
| **검토 의견** | ❌ 그룹 전체 데이터 수집/처리 위치로 부적합 | ❌ 실시간 업무/철회/추적 부적합 | ✅ 적재/처리/분석 모두 안정적 ✅ 실시간 마케팅/철회 대응 가능 |

### 전략적 추천 구성 (3안 기준)



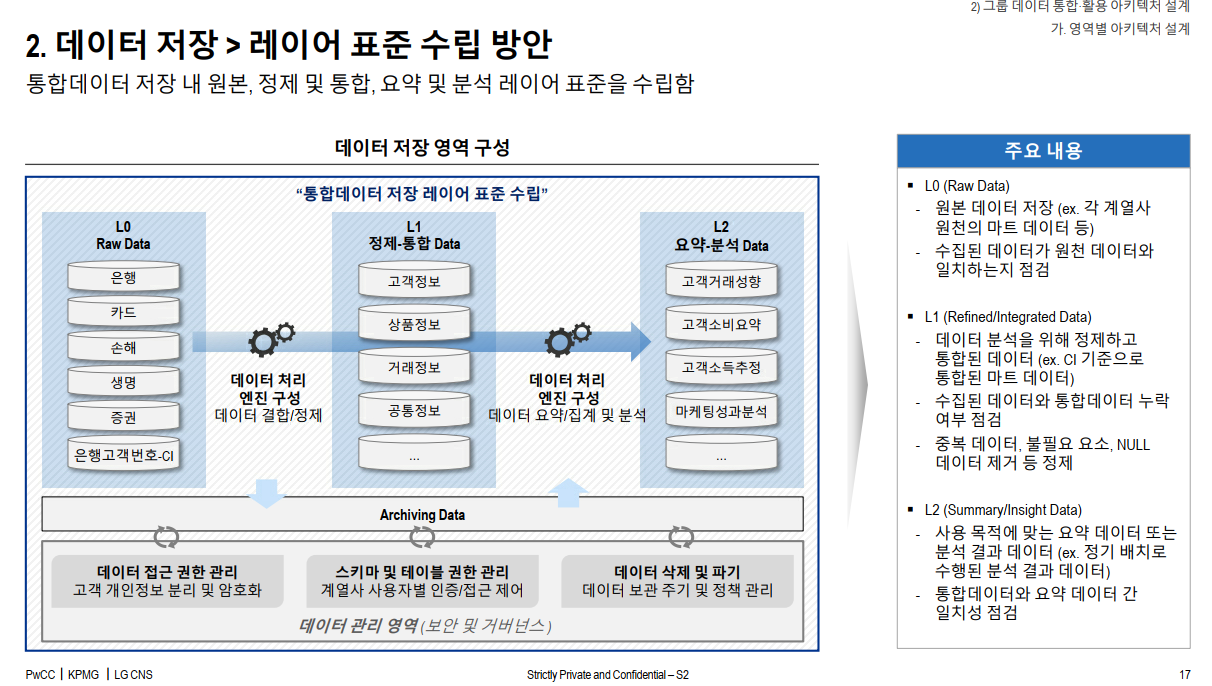
**⟶ 설계 전략 키워드**

|  |  |
| --- | --- |
| 항목 | 전략 |
| 데이터 분리 저장 | 정형 정보 (거래/CI/권한 등) → RDB, 비정형/로그형 → 빅데이터 |
| 보안 정책 반영 | 제3자 제공, 마케팅 동의 철회 등 실시간 반영 위해 별도 DB 구성 필수 |
| 분석 연계 구조 | 빅스퀘어와는 단방향 연계 (분석 요청 기준) |
| 포털/BI 연계 | 통합 데이터 포털 → RDB 연동, OLAP 및 SAS 연계 |
| 고객정보 연계 | OLAP 분석 → 통합 고객번호 기반 연계로 통일 |

### 시사점 (EA 아키텍처 관점)

* **RDB 기반 저장소**와 **Hadoop 기반 분석 플랫폼**의 **역할 분리 전략**을 통해 가용성과 보안성 동시 확보
* 실시간 처리가 중요한 철회/삭제 대상은 반드시 별도 저장소에 존재해야 하며, 이를 위해 **[3안] 별도 AP 구성안**이 가장 합리적
* 향후 통합 마케팅, 통합 포털 분석 요구 증가에 따라 **통합 RDB 기반 설계**를 기본 구조로 보고, 분석 연계만 분기하여 구성하는 것이 적절함

## 데이터 저장 표준 레이어 저장 방안



### ✅ 통합 데이터 저장 구조 레이어 표준화

#### 🔷 1. 레이어 정의 구조

|  |  |  |  |
| --- | --- | --- | --- |
| 레이어 | 명칭 | 주요 내용 | 전략적 역할 |
| **L0** | Raw Data Layer | - 계열사 원천 데이터를 그대로 수집하여 저장 (은행, 카드, 손해, 생명, 증권 등) - 은행고객번호/CI 기준 수집 - 원본 데이터의 정합성 유지 | - 소스 데이터 보존 및 변경이력 추적 - 이상 탐지 및 오류 검증 기준 제공 |
| **L1** | 정제·통합 Data Layer | - L0 기반 정제/통합 수행 - 고객·상품·거래·공통정보 등 통합구조로 구성 - 중복/불필요/NVL 제거 후 품질 보정 | - 통합 고객 분석의 기준 데이터셋 - BI/분석 시스템과 직접 연동 가능 |
| **L2** | 요약·분석 Data Layer | - 마케팅·성과·고객가치 등의 분석 결과를 저장 - L1 기반 집계 및 인사이트 데이터 구성 - 예: 고객성향/소비요약/이탈율 등 | - 마케팅 대상 추출, 성과 리포트, UMS 메시지 발송 등 실시간 활용 |

### 2. 저장 계층별 엔진 구성 및 처리 흐름

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

| L0 Raw Data |

| (은행,카드 등) |

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

|

[ETL] 데이터 정제/변환 엔진 (품질 점검, 필드 정리, 중복 제거)

↓

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

| L1 통합 데이터 |

| 고객/상품/거래정보 |

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

|

[AGG] 요약/분석 엔진 (통계 집계, 마케팅 성과 추출, 리스크 산출 등)

↓

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

| L2 요약/분석 Data |

| 고객성향/소비/성과 등 |

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



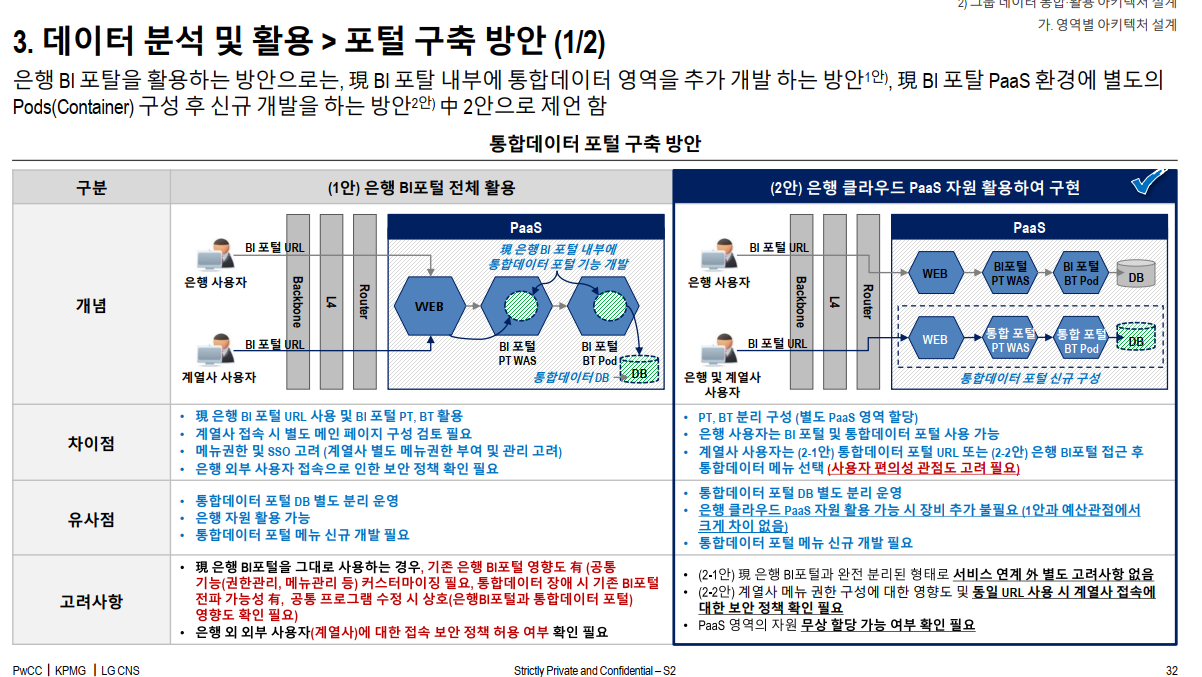
### 🔷 3. 보안 및 거버넌스 구성

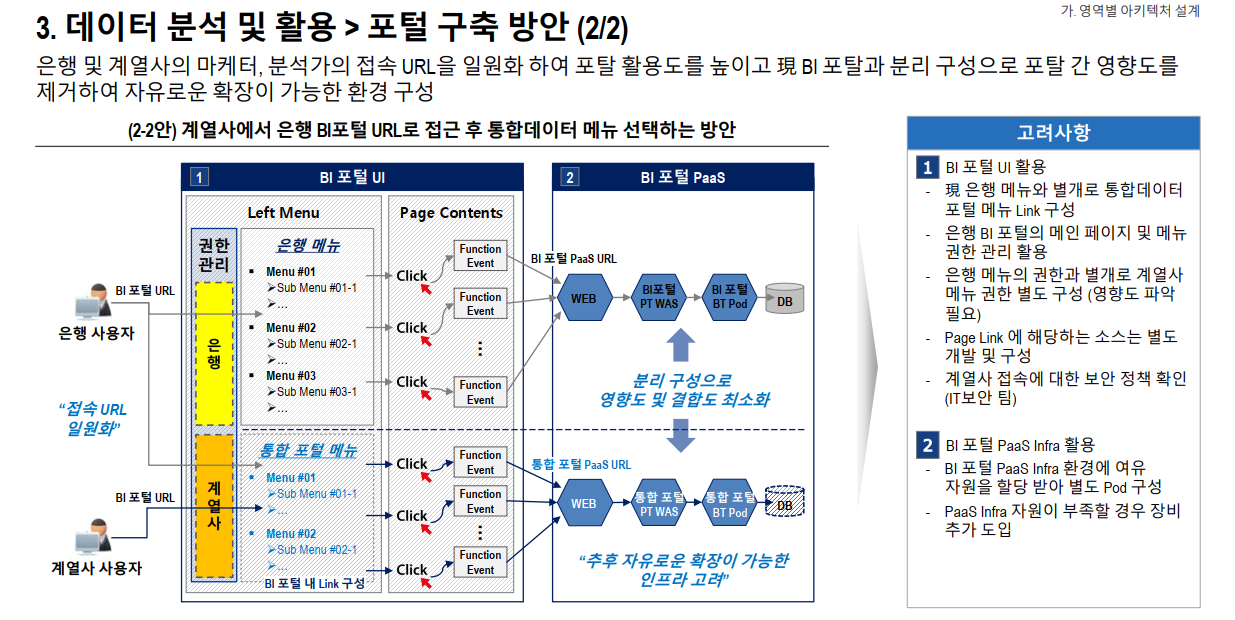
|  |  |
| --- | --- |
| 항목 | 내용 |
| **접근권한** | 고객 개인정보는 L0/L1에서 분리 저장되며, L2에서는 암호화 후 활용 가능 |
| **계정별 권한** | 시스템/사용자/계정별 접근 권한을 테이블/스키마 단위로 설정 |
| **데이터 삭제** | 정책 기반의 보관주기/파기 모듈 운영 (예: 3개월 주기, 배치 삭제 등) |
| **감사/이력관리** | 삭제 이력, 배치 수행이력 등 Logging 시스템 별도 운영 필요 |
| **Archiving 관리** | L0 및 L1 단계에서 일정 기간 후 별도 Archive 저장소로 이관 가능 |

### 📌 EA 전략적 시사점

|  |  |
| --- | --- |
| 분석 항목 | 전략적 의미 |
| **계층 분리 저장** | 데이터 처리 효율성과 정합성 확보, 업무별 분리 처리 용이 |
| **분석 대응 유연성** | OLAP/ML 분석 대상은 L1/L2, BI/리포팅 대상은 L2로 분리 활용 가능 |
| **보안/거버넌스** | 계층별 정책 분리로 민감정보 보호 및 실명 추적 체계 구축 |
| **품질 관리 체계 수립** | L0→L1→L2 간 일관된 품질 점검 및 적재표준 운영 필요 |
| **계열사 확장성 확보** | 레이어 표준이 정해지면 타 계열사 신규 도입 시 확장 적용 용이 |

## BI 은행 /계열사 접근 관리 아키텍처 전략 요약





### 📌 아키텍처 구성 개요

[BI 포털 사용자]

├──> 동일 BI 포털 URL 접근 (은행/계열사 공통 URL)

└─▶ 권한 기반 Left Menu 구성

├─▶ 은행 메뉴 (Menu #01 ~ #03)

│ └─▶ Page Content : Function Event → BI 포털 PT/BT

└─▶ 통합 포털 메뉴 (Menu #01 ~ #02)

└─▶ Page Content : Function Event → 통합 포털 PT/BT

[BI 포털 UI] [BI 포털 PaaS 인프라]

▲ ▲

Web UI ┌───────────┬────────────┐

Menu 구성 │ BI 포털 PT WAS ─▶ BI 포털 BT Pod ─▶ DB

│ 통합 포털 PT WAS ─▶ 통합 포털 BT Pod ─▶ DB



### 📌 아키텍처 특징

|  |  |
| --- | --- |
| 항목 | 설명 |
| **URL 통합 접근 방식** | 사용자(은행/계열사) 모두 동일한 BI 포털 URL을 통해 접근 |
| **UI 분기 방식** | UI 내 Left Menu는 사용자 권한 기반으로 "은행 메뉴"와 "통합 포털 메뉴"를 분기 |
| **PT/BT 구성** | BI 포털과 통합 데이터 포털은 각각 PT Pod, BT Pod를 별도 구성하여 서비스 경계를 명확히 분리 |
| **메뉴 기반 권한 처리** | 메뉴와 페이지 구성은 각각의 권한 정책에 따라 연계하며, 계열사 접근 시 별도 권한 구성이 필요 |
| **확장성 고려** | 통합 포털은 PaaS 자원의 여유 여부에 따라 별도 Pod를 할당하거나 장비 증설 필요 |

### 🧩 고려 사항 정리 (표 기반)

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 세부 항목 | 주요 내용 | 전략적 의미 |
| BI 포털 활용 | 메뉴 구성 분리 | BI 포털 내 은행 메뉴 ↔ 통합 포털 메뉴 분리 구성 | 사용자의 업무 영역에 따른 메뉴 권한 및 접근 제한 설정 가능 |
| 메뉴 링크 구조 | Page Link 별로 별도 WAS 연계 | 메뉴 선택 시 URL은 내부적으로 별도 서비스로 연결되어 있음 | 각 Pod 간 결합도 최소화로 시스템 확장성 및 장애 격리 확보 |
| 권한 정책 분리 | 계열사 사용자 권한은 별도 설정 | 은행과 다른 접근 정책 필요 | 인증 정책 다중화 및 Keycloak 등의 연계 설계 필요 |
| 보안 정책 | URL은 통합이나 사용자 정책은 분리 필요 | L4, ACL, Role 인증기반 분리 | API Gateway + JWT 기반 정책 적용 고려 |
| 인프라 활용 | Pod 및 자원 분리 | PaaS 인프라 내 자원 부족 시 장비 확장 | Kubernetes 기반 탄력적 확장 정책 연계 필수 |

### 🔧 향후 설계 적용 방향

* **① PaaS 기반 Pod 분리 정책**
  + 은행 사용자: BI 포털 PT/BT POD 접근
  + 계열사 사용자: 통합 포털 PT/BT POD 접근
  + 둘 다 동일한 URL을 통해 접근하되 **메뉴 기반 분기** 및 **라우팅 방식**은 분리
* **② Role 기반 인증 정책 구성**
  + Keycloak 또는 OAuth2 기반 인증/인가 연계
  + 메뉴별 접근 제어 정책(메뉴 코드/권한 그룹 매핑 DB 구축)
* **③ 서비스 확장 및 이중화 고려**
  + 각 포털의 PT/BT Pod는 **독립 구성** → 고가용성(HA) 이중화 필요
  + WAS 간 Load Balancing 및 Liveness Check 설정

# 제약사항

## ****아키텍처 제약사항 요약표****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **구분** | **제약사항 ID** | **세부 항목** | **주요 내용** | **설계 구현 전략** | **전략적 의미** |
| 포털 구조 | 확인 01 | PaaS 내 포털 구성 | NH 금융지주 전용 BI 포털은 PaaS 내 별도 구성 | 독립 Pod 운영 (BT/WEB 별도) | 조직 분리 및 운영 분리 대응 |
| 접속 구성 | 확인 02 | 통합 URL 구성 | 은행/계열사 접속은 동일 URL 사용 | Proxy + Reverse Mapping, Routing 기반 분기 | 사용자 혼동 최소화, 운영 효율 증대 |
| 아키텍처 | 확인 03 | PT/BT 분리 | PT WAS와 BT WAS는 아키텍처 상 분리 | Spring Boot 기반 이중 POD 설계 | 가용성 확보 및 장애 전파 차단 |
| DB 연계 | 확인 04 | 포털 DB 공유 여부 | 은행과 DB 공유 여부 미결정 | 논리적 DB 스키마 분리 또는 물리적 이중화 옵션 제안 | 보안/성능 정책에 따라 유연 대응 가능 |
| 보안 정책 | 확인 05 | 계열사 보안 정책 | 통일 URL 사용 시 계열사 접속 정책 필요 | IP 인증, VPN, Header 기반 분기, MFA 연계 | 계열사 보안 규제 및 ISMS 대응 가능 |
| 권한 관리 | 확인 06 | 포털 메뉴 권한 | 메뉴별 은행/계열사 분리 및 접속 제어 필요 | 메뉴 접근권한 + URI 기반 ACL 구성 | 업무 도메인 분리 및 정보 유출 방지 |
| 메타 연계 | 확인 07 | 테이블/컬럼 메타 연계 | IT 메타시스템과 테이블/컬럼 연계 필요 | 실시간 API or 배치 기반 연동 설계 | 메타 최신화 유지, 데이터 카탈로그 구성 |
| 요청 관리 | 확인 08 | 데이터 요청 워크플로우 | 요청서 결재 흐름 필요, 자체 개발 또는 솔루션 검토 | 자체 BPM 개발 or 오픈소스 Workflow(Activiti, Camunda 등) | 요청 이력 관리 및 결재 추적 가능 |