|  |  |
| --- | --- |
| **전사 EA 표준 Profile정의서** | |
| **사내 표준 정의 사업**  NH 농협지주 데이터 플랫폼 구축  **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. 아키텍처 3](#_Toc195542829)

[1.1. 도입 목적 3](#_Toc195542830)

[1.2. 도입 전 선결과제 3](#_Toc195542831)

[1.3. 핵심 아키텍처 기술 요소 4](#_Toc195542832)

[1.4. 핵심 특징 및 장단점 비교 4](#_Toc195542833)

[2. 시장 상황 6](#_Toc195542834)

[2.1. 시장 동향 및 기술 연계 영향도 6](#_Toc195542835)

[2.1.1. 시장 상황 및 동향 6](#_Toc195542836)

[2.1.2. 기술동향 영향도 평가 6](#_Toc195542837)

[2.2. 시장 점유율 및 주요 기업 채택 사유 7](#_Toc195542838)

[2.3. 주요 API 문서화툴 완성도 평가 8](#_Toc195542839)

[2.4. 기술 트렌드에 따른 영향도 및 이유 9](#_Toc195542840)

[3. 기술 동향 10](#_Toc195542841)

[3.1. 전통적인 API 문서화툴(도구)들의 한계점과 시사점 10](#_Toc195542842)

[3.1.1. 아키텍처 한계점 10](#_Toc195542843)

[3.2. 최신 핵심 아키텍처 기술요소 및 구현 사례 10](#_Toc195542844)

[3.3. 핵심 아키텍처 사상 (텍스트 기반) 11](#_Toc195542845)

[3.4. API 문서화툴의 한계점에 대한 종합적인 시사점 (정리) 12](#_Toc195542846)

[3.4.1. 기술적 한계 측면 12](#_Toc195542847)

[3.4.2. API 문서화툴 도입 시 기술 과제 12](#_Toc195542848)

[3.4.3. 기업환경에서 API 문서화툴 선택 시 고려 요소 12](#_Toc195542849)

[3.4.4. API 문서화툴 도입 시 미래 대응을 위한 전략적 시사점 12](#_Toc195542850)

[3.5. 기술 트렌드별 영향 API 문서화툴 및 기술 연계 설명 13](#_Toc195542851)

[4. 요구사항 정의 14](#_Toc195542852)

[4.1. API 문서화툴 선택 요구사항 정의표 14](#_Toc195542853)

[4.2. 자가진단 체크 14](#_Toc195542854)

[4.3. 자가진단 기반 선택 가이드 15](#_Toc195542855)

[5. 도구 선정 가이드 17](#_Toc195542856)

[5.1. 선정조건 정리 17](#_Toc195542857)

[5.2. API 문서화툴 선정을 위한 평가기준 및 가중치 구성 17](#_Toc195542858)

[5.3. API 문서화툴 평가항목 점수표 18](#_Toc195542859)

[5.4. 프로젝트 유형별 · 상황별 · 규모별 맞춤 선정표 19](#_Toc195542860)

[5.4.1. 프로젝트 유형별/상황별 추천표 19](#_Toc195542861)

[5.4.2. 프로젝트 규모별 적용 가이드 19](#_Toc195542862)

[6. 부록 21](#_Toc195542863)

[6.1. 선정조건과 평가기준의 관계 21](#_Toc195542864)

# 개요 및 도입전략

## ****프로젝트 배경 및 목적****

NH 농협지주 통합 데이터 플랫폼 구축 사업은 금융권 디지털 전환 및 데이터 기반 의사결정 강화를 목표로 추진됩니다.  
금융 데이터의 실시간 처리, 빅데이터 분석, 마케팅 자동화, 고객 맞춤형 서비스 제공을 통해 금융권의 경쟁력을 극대화하고,  
다양한 계열사(NH농협은행, NH농협카드, NH투자증권, NH생명보험, NH손해보험)의 데이터를 통합하여 중앙 집중식 데이터 관리 체계를 구축합니다.

**주요 추진 목적**

1. **데이터 통합 관리 및 분석 최적화**
   * 계열사 간 분산된 데이터를 통합 관리하여 분석 효율성을 극대화
   * 실시간 분석 환경을 통해 빠르고 정확한 의사결정 지원
2. **비즈니스 민첩성 확보**
   * MSA(Microservice Architecture) 기반의 유연한 확장성 보장
   * 이벤트 기반의 실시간 마케팅 및 고객 맞춤형 서비스 구현
3. **보안 및 규제 준수 강화**
   * 금융 데이터의 안정성 및 보안 강화를 위한 E2E(End-to-End) Encryption 적용
   * 개인정보보호법 및 금융 관련 규제에 완벽히 대응

## ****제안 아키텍처 도입전략****

NH 금융지주의 통합 데이터 플랫폼은 최신 클라우드 네이티브 기술을 기반으로 설계되며, 다음의 전략을 따릅니다.

|  |  |
| --- | --- |
| 전략 구분 | 주요 내용 |
| **클라우드 네이티브** | Kubernetes 기반의 MSA(Microservice Architecture) 구조 설계 |
| **Data Lake 구축** | HDFS, AWS S3를 활용한 대용량 데이터 통합 관리 |
| **실시간 스트림 처리** | Kafka 및 Spark Streaming을 통한 실시간 분석 환경 구축 |
| **보안 강화 및 규제 대응** | OAuth2.0, Keycloak, RBAC 기반의 접근 제어 및 TLS 1.3 전송 구간 암호화 적용 |
| **API Gateway 도입** | Spring Cloud Gateway를 통한 통합 API 관리와 보안 정책 통합 |
| **CI/CD 자동화** | Jenkins, ArgoCD를 통한 지속적 배포와 자동화된 업데이트 관리 |

## ****기대효과 및 주요 특징****

* **데이터 통합의 일원화**
  + 기존에 분산된 금융 데이터를 통합하여 관리함으로써 분석 시간 단축 및 일관성 유지
* **실시간 마케팅 및 고객 분석**
  + Spark Streaming 및 OLAP 연동을 통한 실시간 타겟 마케팅과 고객 세분화
* **확장성과 유연성 보장**
  + Kubernetes 기반의 오토스케일링으로 트래픽 급증 시에도 안정적인 서비스 제공
* **보안 및 규제 대응 최적화**
  + 데이터 전송 및 저장 시 E2E Encryption을 통한 보안성 강화
  + 금융 데이터에 대한 규제 준수 (금융감독원, 개인정보보호법 등)
* **지속적 배포 및 무중단 운영**
  + ArgoCD를 통한 Blue-Green Deployment, Canary Release 등 무중단 배포 전략 적용

# ****요구사항 분석****

## ****기능 요구사항 분석****

|  |  |
| --- | --- |
| 구분 | 요구사항 내용 |
| **데이터 수집** | - 계열사 간 통합 데이터 수집 (NH농협은행, NH농협카드, NH투자증권 등) |
|  | - 실시간 스트리밍 데이터 수집 (Kafka, Spark Streaming) |
|  | - 정기 배치 수집 (DataStage, ETL 파이프라인 구성) |
| **데이터 처리** | - 실시간 이벤트 처리 및 비동기 메시지 처리 (RabbitMQ, Kafka) |
|  | - 정형/비정형 데이터 처리 및 저장 (HDFS, S3) |
|  | - 데이터 정제 및 변환 (Spark SQL, DataStage) |
| **데이터 저장** | - 원천 데이터 적재 및 통합 관리 (Data Lake) |
|  | - 목적별 마트 개발 및 실시간 분석용 DB 생성 (Data Mart, OLAP) |
|  | - 통합 고객번호 생성 및 관리 |
| **데이터 분석** | - OLAP 환경 제공 및 실시간 분석 대시보드 생성 |
|  | - 메타데이터 조회 및 관리 (Data Catalog) |
|  | - 데이터 품질 관리 및 검증 |
| **마케팅 분석** | - 캠페인 설정 및 실시간 EBM (Event-Based Marketing) 실행 |
|  | - 실시간 개인화 마케팅 및 고객 타겟팅 분석 |
| **데이터 제공** | - 타겟 고객 추출 및 분석 결과 제공 |
|  | - 금융 리포트 생성 및 시각화 (Tableau, PowerBI) |

## ****비기능 요구사항 분석****

|  |  |
| --- | --- |
| 구분 | 요구사항 내용 |
| **보안** | - TLS 1.3, IPSec을 통한 데이터 전송 구간 암호화 |
|  | - AES-256 암호화 알고리즘을 통한 데이터 저장 보안 |
|  | - OAuth2.0 및 Keycloak을 통한 인증 및 권한 관리 |
| **성능** | - 초당 10,000 TPS 처리 성능 보장 |
|  | - API 응답 시간 3초 이내 유지 |
|  | - 대량 트랜잭션 처리 시 비동기 메시지 큐 활용 (Kafka, RabbitMQ) |
| **가용성** | - 99.99% SLA 보장 |
|  | - 무중단 배포 (Blue-Green, Canary Release) |
|  | - Kubernetes 기반 오토스케일링 적용 |
| **확장성** | - 수평적 확장 (Horizontal Scaling) 및 클러스터 확장 최적화 |
|  | - Data Lake와 Data Mart의 유연한 확장성 |
| **유지보수성** | - CI/CD 자동화 배포 (Jenkins, ArgoCD) |
|  | - IaC(Infrastructure as Code) 기반의 자동화된 인프라 관리 |
| **감사 및 로깅** | - ELK Stack을 통한 실시간 로그 수집 및 분석 |
|  | - 데이터 접근 로그 및 감사 기록 보존 (3년 이상) |

## ****법적/규제적 요구사항 대응****

|  |  |
| --- | --- |
| 구분 | 요구사항 내용 |
| **개인정보 보호법** | - 개인정보 암호화 및 저장 시 AES-256 알고리즘 적용 |
|  | - 개인정보 마스킹 처리 (조회 시 비식별화 처리) |
|  | - 파기 시 데이터 완전 삭제 (Zero Fill 처리) |
| **금융감독원 규제** | - 금융 데이터에 대한 접근 권한 통제 및 감사 로그 보존 |
|  | - 금융거래 기록 5년 이상 보관 |
|  | - 금융기관 간의 데이터 전송 시 전송 구간 암호화 |
| **전자금융거래법** | - API 접근 시 인증 및 권한 검사 (OAuth2.0, JWT 기반 인증) |
|  | - 전자 거래의 무결성과 기밀성을 보장하는 트랜잭션 처리 |

# 기술 요소 정의 및 아키텍처 전략맵

## ****기술 스택 구성 분석****

|  |  |  |
| --- | --- | --- |
| 기술 영역 | 기술 스택 | 주요 역할 및 설명 |
| **인프라** | Kubernetes, OpenShift, VMware | Private Cloud 기반의 컨테이너 오케스트레이션 및 클러스터 관리 |
| **웹 서버** | Apache, Nginx | 정적 파일 제공, 로드 밸런싱 및 보안 처리 |
| **WAS 서버** | Tomcat, Spring Boot | 애플리케이션 실행 환경 제공 및 REST API 관리 |
| **API Gateway** | Spring Cloud Gateway | API 호출 관리 및 트래픽 제어, 보안 정책 적용 |
| **데이터베이스** | Oracle HEX, MySQL, HDFS, AWS S3 | 원천 데이터 저장 및 실시간 분석, Data Lake 구성 |
| **ETL 도구** | IBM DataStage, Apache Spark | 대용량 배치 처리, 실시간 스트림 분석 |
| **메시징 시스템** | Kafka, RabbitMQ | 비동기 메시지 전달 및 실시간 이벤트 처리 |
| **보안 솔루션** | Keycloak, OAuth2.0, Spring Security | SSO 인증, RBAC, OAuth2.0 기반의 접근 통제 |
| **모니터링 도구** | Prometheus, Grafana, ELK Stack | 실시간 모니터링, 로그 수집 및 시각화 |
| **CI/CD 도구** | Jenkins, ArgoCD, Docker | 지속적 배포 및 버전 관리, 컨테이너화 배포 최적화 |

## ****주요 기술 요소 정의****

1. **MSA (Microservice Architecture)**
   * Spring Boot 기반의 독립적인 마이크로서비스로 설계
   * 도메인 단위 분리 및 API Gateway(Spring Cloud Gateway) 연동
   * 각 서비스는 독립적으로 배포 및 확장이 가능하며, 장애 격리 (Circuit Breaker) 구조를 적용
2. **Data Lake 및 Data Mart**
   * HDFS와 AWS S3를 활용한 대용량 비정형 데이터 저장
   * Data Mart로 OLAP 환경을 구성하여 실시간 분석과 리포팅 최적화
   * DataStage를 통한 데이터 클렌징 및 전처리 작업 수행
3. **실시간 스트림 처리**
   * Kafka 및 Spark Streaming을 활용하여 실시간 데이터 스트림 처리
   * 실시간 데이터 분석 및 로그 수집, 이벤트 처리 최적화
   * 금융 거래 및 고객 행동 데이터를 실시간으로 집계 및 분석
4. **API Gateway와 인터페이스 연계**
   * Spring Cloud Gateway를 통해 내부 API 및 외부 연계 API 표준화
   * OAuth2.0과 JWT 인증 방식을 통한 보안 처리
   * 트래픽 부하 분산 및 장애 격리 구조 적용
5. **보안 및 인증 체계**
   * Keycloak 기반의 통합 인증 및 RBAC(Role-Based Access Control)
   * 전송 구간(TLS 1.3) 및 저장 구간(AES-256) 암호화
   * MFT(Managed File Transfer) 및 SFTP를 활용한 보안 전송

## ****제안 아키텍처 전략맵****

### ****전략적 아키텍처 구성****

1. **Data Ingestion Layer**
   * 수집된 데이터를 Data Lake에 저장 (HDFS, AWS S3)
   * 실시간 스트림 처리는 Kafka 및 Spark Streaming으로 처리
2. **Data Processing Layer**
   * DataStage 및 Spark를 활용한 대용량 배치 처리
   * 데이터 정제 및 통합 작업 진행
3. **Data Storage Layer**
   * OLAP 및 Data Mart에 최적화된 데이터 저장
   * 분석 목적의 구조화된 데이터 관리
4. **Business Logic Layer**
   * Spring Boot 기반의 Microservices 배포
   * 비즈니스 로직 처리 및 데이터 연계
5. **API Gateway Layer**
   * Spring Cloud Gateway를 통한 API 관리
   * 인증 및 권한 관리 (OAuth2.0, JWT)
6. **Presentation Layer**
   * Tableau 및 PowerBI를 활용한 실시간 분석 대시보드
   * 사용자 인터페이스와 API 호출 처리
7. **Security & Monitoring Layer**
   * Keycloak 및 Spring Security를 통한 인증 관리
   * Prometheus & Grafana를 통한 실시간 모니터링
   * ELK Stack을 활용한 로그 수집 및 분석

# ****아키텍처 설계****

## ****AA (Application Architecture) 설계****

### ****서비스 도메인 구조****

* **Microservice 기반 서비스 구성**
  + 서비스별 도메인 분리: 고객 관리, 상품 관리, 거래 관리, 분석 관리 등
  + Spring Boot 기반의 독립적인 서비스 구조
  + 각 도메인은 API Gateway(Spring Cloud Gateway)와 연계하여 접근 제어
* **서비스 간 연계 구조**
  + Kafka를 활용한 비동기 이벤트 연계
  + RESTful API 표준화로 서비스 호출 간 데이터 일관성 보장
  + Circuit Breaker 패턴을 통해 장애 격리

### ****내부 구조 및 설계 패턴****

* **Layered Architecture**
  + Presentation Layer: Spring MVC로 사용자 요청 처리
  + Business Logic Layer: Service와 Facade 패턴을 활용한 비즈니스 로직 분리
  + Data Access Layer: JPA/Hibernate를 통한 데이터 관리
  + Integration Layer: Kafka와 RabbitMQ를 통한 비동기 통합 처리
* **핵심 설계 패턴**
  + **Repository Pattern**: 데이터 접근 계층을 모듈화하여 재사용성 확보
  + **Circuit Breaker Pattern**: 장애 전파 방지 및 자동 복구
  + **Event Sourcing**: Kafka 기반의 이벤트 로그 수집 및 복원

### ****트랜잭션 및 오류 처리 구조****

* **트랜잭션 관리**
  + JTA(Java Transaction API) 기반 분산 트랜잭션 관리
  + 2PC(Two-Phase Commit) 적용으로 데이터 일관성 보장
* **오류 처리 구조**
  + Global Exception Handler를 이용하여 통합 오류 관리
  + Kafka Dead Letter Queue(DLQ)를 활용한 실패 메시지 처리
  + 오류 로그는 ELK Stack으로 수집하여 분석

### ****API 설계 및 운영방식****

* **API 설계 원칙**
  + RESTful 설계 규칙 준수 (HTTP 메서드: GET, POST, PUT, DELETE)
  + URI 설계: /api/v1/{resource}/{id} 형식으로 규격화
* **운영 방식**
  + Swagger를 이용한 API 명세 자동화
  + API 버전 관리 정책 수립 (예: v1, v2 등)
  + Gateway를 통한 API 트래픽 제어 및 라우팅

### ****유효성 검증 및 공통 예외처리****

* **입력 데이터 검증**
  + @Valid 어노테이션을 통한 필드 유효성 체크
  + Bean Validation을 이용하여 데이터 형식 검증
* **공통 예외 처리**
  + Custom Exception 정의: InvalidInputException, ServiceUnavailableException
  + 예외 처리 컨트롤러: @ControllerAdvice로 전역 예외 처리
  + HTTP Status Code 매핑: 400 (Bad Request), 500 (Internal Server Error)

### ****테스트 전략****

* **단위 테스트**
  + JUnit과 Mockito를 이용하여 Service와 Controller 테스트
  + MockMvc를 이용한 API 통합 테스트
* **통합 테스트**
  + Spring Boot Test를 활용하여 MSA 간 연계 테스트
  + Kafka 컨슈머와 프로듀서 테스트를 통한 이벤트 흐름 검증
* **성능 테스트**
  + JMeter를 활용한 TPS 테스트
  + 실시간 데이터 스트리밍 테스트 (Kafka Producer/Consumer 성능 측정)

## ****TA (Technical Architecture) 설계****

### ****실행 환경 및 플랫폼 구조****

* **Kubernetes 기반 컨테이너 관리**
  + 클러스터 구성: Master-Worker 노드로 분리
  + Helm Chart를 활용한 애플리케이션 배포 자동화
* **클라우드 인프라 구성**
  + Private Cloud (VMware)와 퍼블릭 클라우드 (AWS) 혼합 사용
  + IaaS(PaaS) 환경에서 애플리케이션 자동 확장

### ****CI/CD 자동화 파이프라인****

* **CI 파이프라인**
  + Jenkins를 이용한 코드 빌드 및 정적 분석
  + SonarQube를 통한 코드 품질 점검
* **CD 파이프라인**
  + ArgoCD를 활용한 GitOps 기반 배포
  + Blue-Green Deployment 및 Canary Release 전략

### ****시스템 확장성 및 가용성 설계****

* **오토스케일링(Auto-Scaling) 전략**
  + Kubernetes Horizontal Pod Autoscaler(HPA) 적용
  + CPU, Memory 사용량에 따른 동적 확장
  + AWS Auto Scaling Group과 연동하여 노드 확장
* **로드 밸런싱 설계**
  + **HAProxy**와 **Nginx Ingress Controller**를 이용한 부하 분산
  + API Gateway(Spring Cloud Gateway)를 통해 서비스별 트래픽 제어
  + 글로벌 트래픽 관리를 위한 **AWS Global Accelerator** 적용
* **멀티 리전(Multi-Region) 아키텍처**
  + 지역 간 재해복구(Disaster Recovery) 설정
  + Active-Active 또는 Active-Passive 구성 지원
  + RPO(Recovery Point Objective): 15분 이내
  + RTO(Recovery Time Objective): 10분 이내
* **데이터 복제 및 백업 전략**
  + HDFS 및 S3 버킷 간 Cross-Region Replication 설정
  + Ceph를 활용한 실시간 Snapshot 생성
  + Aurora Global Database로 다중 리전 데이터 동기화
* **지속적인 무중단 배포**
  + Blue-Green Deployment와 Canary Release를 통한 배포
  + ArgoCD를 활용한 GitOps 방식의 실시간 배포 관리
  + 트래픽 점진적 전환으로 사용자 영향 최소화

### ****보안 인프라 및 정책****

* **네트워크 보안**
  + VPC(Virtual Private Cloud)와 Subnet으로 네트워크 격리
  + Security Group과 Network ACL을 통한 방화벽 설정
  + **IPSec VPN**과 **TLS 1.3**을 이용한 전송 구간 암호화
* **API 및 서비스 보안**
  + Spring Security를 활용한 인증 및 권한 관리
  + OAuth2.0 기반 인증과 JWT 토큰 발급
  + Keycloak을 통한 Role-Based Access Control(RBAC) 설정
* **데이터 보안 및 암호화**
  + AES-256 암호화를 통한 데이터 저장 보안
  + HashiCorp Vault를 활용한 API Key 및 비밀정보 관리
  + TLS(Transport Layer Security)로 데이터 전송 보안 강화
* **감사 및 접근 제어**
  + ELK Stack을 활용한 실시간 모니터링 및 감사 로그 수집
  + SIEM(Security Information and Event Management) 통합
  + RBAC(Role-Based Access Control)로 접근 권한 관리

### ****모니터링 및 장애 대응****

* **모니터링 시스템 구성**
  + Prometheus와 Grafana를 통한 실시간 모니터링
  + 주요 메트릭: CPU, Memory, Disk I/O, Network Latency
  + 커스텀 메트릭 수집: Kafka Lag, API 응답시간, ETL 처리시간
* **알림 및 대응 체계**
  + AlertManager와 Slack 연동으로 실시간 알림
  + 장애 발생 시 Incident Response Plan(IRP) 자동 실행
* **장애 복구 및 Auto Healing**
  + Kubernetes의 Self-Healing을 통한 장애 복구
  + Failed Pod 자동 재시작 및 Rollback 처리
  + Ceph Snapshot 및 HDFS Backup을 통한 데이터 복구

### ****운영 배포 정책****

* **Blue-Green Deployment**
  + 두 개의 환경(Blue, Green)으로 나누어 무중단 배포
  + 신규 버전 배포 후 검증 완료 시 트래픽 전환
* **Canary Deployment**
  + 일부 사용자에게만 신규 버전 배포 후 모니터링
  + 문제 발생 시 즉시 롤백 가능
* **Rolling Update**
  + 서비스 중단 없이 단계적 배포
  + Pod 단위의 순차적 업데이트로 Downtime 최소화
* **Helm Chart 배포 관리**
  + 버전 관리와 의존성 관리를 Helm을 통해 수행
  + ConfigMap 및 Secret 동기화 자동화

## ****DA (Data Architecture) 설계****

### ****데이터 저장소 구성****

* **Data Lake**
  + HDFS와 AWS S3를 기반으로 대용량 데이터 저장
  + 원천 데이터, 정제된 데이터, 분석 목적 데이터를 계층화하여 관리
  + 실시간 스트림 데이터는 Kafka Connect를 통해 Data Lake로 적재
* **Data Mart**
  + 분석 목적에 최적화된 데이터 마트 구성
  + OLAP 분석을 위한 다차원 데이터 모델링
  + 주요 데이터 마트:
    - 고객 마트 (Customer Mart)
    - 거래 마트 (Transaction Mart)
    - 마케팅 분석 마트 (Marketing Analysis Mart)
    - 리스크 관리 마트 (Risk Management Mart)
* **Operational Data Store (ODS)**
  + 실시간 거래 처리 및 분석을 위한 중간 저장소
  + 이벤트 처리 및 빠른 조회를 위한 Redis Cache 활용
  + 빠른 응답이 요구되는 API 호출에 최적화

### ****데이터 흐름 구조****

* **Batch Data Pipeline**
  + IBM DataStage를 활용한 ETL 작업
  + 정기적 배치 수행 (일/주/월 단위)
  + 주요 처리 항목:
    - 고객정보 통합 및 정제
    - 금융 거래 내역 수집 및 분석
* **Real-Time Data Pipeline**
  + Kafka와 Spark Streaming을 통한 실시간 데이터 처리
  + 데이터 생성 시점부터 Data Lake까지 실시간 연동
  + 주요 처리 항목:
    - 실시간 금융 거래 감시
    - 이벤트 기반 고객 분석
    - 마케팅 캠페인 실행
* **Data Synchronization**
  + CDC(Change Data Capture)를 통한 실시간 데이터 동기화
  + PostgreSQL 및 Oracle DB와의 데이터 복제
  + 변경 발생 시 Kafka Topic으로 전송 후 처리

### ****데이터 접근 및 조회 전략****

* **Data Access Layer (DAL)**
  + Spring Data JPA를 통한 CRUD 처리 최적화
  + 복잡한 조회 시 QueryDSL 사용
  + 대량 데이터 조회 시 Pagination 처리
* **CQRS(Command Query Responsibility Segregation)**
  + 읽기(Read)와 쓰기(Write) 연산을 분리하여 성능 최적화
  + Kafka Streams를 통한 비동기 메시지 전송
  + 분석 업무와 트랜잭션 업무를 별도로 관리
* **Caching 전략**
  + Redis와 Hazelcast를 활용한 캐싱 처리
  + 세션 관리와 빈번한 조회에 대한 빠른 응답 제공

### ****배치 및 실시간 처리 전략****

* **Batch Processing**
  + DataStage와 Spring Batch를 활용한 대량 데이터 처리
  + 배치 작업의 스케줄링: Control-M을 이용한 시간 단위 실행
  + 배치 실패 시 자동 재시도(3회) 및 로깅 처리
* **Stream Processing**
  + Kafka Streams와 Spark Streaming을 통한 실시간 이벤트 처리
  + 이벤트 기반 처리 구조로 빠른 응답성과 확장성 보장
  + 데이터 파이프라인 내 복원성 확보

### ****데이터 보안 및 감사 대응****

* **데이터 암호화**
  + AES-256을 이용한 저장 데이터 암호화
  + 전송 시 TLS 1.3을 통한 구간 암호화
  + 중요 데이터는 HashiCorp Vault를 이용한 비밀 정보 관리
* **접근 제어 및 감사 로깅**
  + RBAC(Role-Based Access Control) 적용
  + 모든 데이터 접근은 ELK Stack을 통해 실시간 로깅
  + 접근 내역에 대한 감사 보고서 생성
* **Masking 및 Tokenization**
  + 개인정보에 대해 마스킹 처리 후 저장
  + 고객 식별 정보는 Tokenization을 통해 보호

### ****데이터 백업 및 복구 구조****

* **Data Backup**
  + HDFS Snapshot 및 AWS S3 버킷 동기화
  + 데이터 보관 주기: 일일 Incremental Backup, 주간 Full Backup
  + 오프사이트 백업을 통한 재해 복구 대비
* **Disaster Recovery (DR) 전략**
  + Multi-Region 배포를 통한 장애 복구
  + Ceph Replication을 통한 실시간 데이터 복제
  + DR 테스트 주기: 월 1회 정기 테스트
* **데이터 복구 전략**
  + RPO(Recovery Point Objective): 15분 이내
  + RTO(Recovery Time Objective): 10분 이내
  + 복구 시점 선택 지원 (Point-in-Time Recovery)

## ****IA (Integration Architecture) 설계****

### ****인터페이스 구조 개요****

* **내부 인터페이스**
  + NH농협은행, NH카드, NH투자증권 등 계열사 간 데이터 송수신
  + Spring Cloud Gateway를 통한 API 통합 관리
  + Kafka 및 RabbitMQ를 활용한 비동기 메시지 처리
* **외부 인터페이스**
  + 금융결제원, 한국은행, 신용평가사 등 외부 금융기관 연동
  + FEP(Front-End Processor)를 통한 실시간 데이터 전송
  + MFT(Managed File Transfer)로 대용량 파일 안전 전송
* **표준 연계 프로토콜**
  + REST API (JSON): CRUD 작업 및 실시간 트랜잭션 처리
  + SOAP API (XML): 금융 표준 연동 및 외부 금융기관 송수신
  + MQ(Message Queue): 대량의 트랜잭션 처리

### ****연계 기술 스펙 및 구성도****

* **연계 기술 구성**
  + API Gateway(Spring Cloud Gateway): 모든 API 트래픽을 중앙 집중화하여 관리
  + Kafka Connect: 데이터 스트리밍 및 실시간 데이터 연동
  + FEP: 외부 금융기관과의 안정적인 데이터 송수신
* **구성도**

[Client] → [API Gateway] → [Microservices] → [Database]

↓

[Kafka Stream]

↓

[Data Lake / Data Mart]

* **데이터 전송 경로**
  1. 클라이언트 요청 → API Gateway로 전달
  2. Microservices에서 비즈니스 로직 처리
  3. Kafka를 통해 실시간 스트리밍 처리
  4. Data Lake와 Data Mart에 저장

### ****연계 표준 설계****

* **REST API 설계 표준**
  + HTTP Methods: GET, POST, PUT, DELETE
  + URI 구조: /api/v1/{service}/{resource}/{id}
  + 응답 형식: JSON (JavaScript Object Notation)
  + 예시: /api/v1/customer/12345
* **MQ 통신 설계 표준**
  + RabbitMQ: 대량 메시지 처리 및 비동기 통신
  + Kafka: 실시간 이벤트 스트리밍
  + Durable Queue 설정을 통한 데이터 유실 방지
* **파일 전송 설계 표준**
  + MFT(Managed File Transfer)
  + SFTP 및 FTPS를 통한 파일 암호화 전송
  + 전송 이력 관리 및 복구 기능 포함

### ****연계 안정화 및 오류 대응 전략****

* **Failover 및 Load Balancing**
  + API Gateway 레벨에서 Circuit Breaker 적용
  + Retry Logic 설정을 통해 일시적 네트워크 장애 극복
  + 메시지 재처리를 위한 Dead Letter Queue 설정
* **에러 로깅 및 트랜잭션 복구**
  + ELK Stack을 통한 실시간 에러 모니터링
  + Kafka Streams의 Exactly Once Processing 보장
  + 장애 발생 시 Rollback 및 트랜잭션 복구
* **연계 테스트 및 검증**
  + RestAssured와 WireMock을 통한 API 연계 테스트
  + JMeter를 활용한 성능 테스트 및 부하 검증

### ****인터페이스 버전 관리 및 확장 전략****

* **API 버전 관리**
  + URI에 버전 명시 (/api/v1 → /api/v2)
  + 기존 API와의 호환성 유지
  + Swagger를 통해 API 명세 자동화
* **확장 전략**
  + MSA(Microservice Architecture) 기반의 독립적 확장 가능
  + Spring Cloud Config를 통한 설정 관리
  + API Gateway에서 신규 서비스 라우팅 추가 시 무중단 반영

# 보안·인증·성능 설계

## ****보안 설계 전략****

* **네트워크 보안**
  + VPC(Virtual Private Cloud)와 Subnet을 활용한 네트워크 격리
  + Security Group과 NACL(Network ACL)을 통한 방화벽 설정
  + 외부 API 접근 시 **IPSec VPN**을 통한 암호화 전송
* **시스템 보안 (K8s 보안 포함)**
  + Kubernetes RBAC(Role-Based Access Control)로 접근 제어
  + Pod Security Policy(PSP) 적용으로 권한 최소화
  + Helm Chart에 보안 정책 정의 및 배포 시 검증
* **API 및 서비스 보안**
  + Spring Security와 Keycloak을 통한 OAuth2.0 인증 관리
  + JWT(JSON Web Token) 기반의 API 접근 제어
  + Mutual TLS(Transport Layer Security)로 서비스 간 안전한 통신
* **데이터 보안**
  + AES-256을 통한 데이터 저장 시 암호화
  + TLS 1.3을 적용한 전송 구간 암호화
  + HashiCorp Vault를 이용한 비밀 정보 관리 (Database Credentials, API Keys)
* **로그 보안**
  + ELK Stack (Elasticsearch, Logstash, Kibana)을 통한 실시간 로그 수집
  + 보안 이벤트와 접근 내역의 Audit Log 생성
  + 3년간 로그 보존 정책 및 주기적 아카이빙
* **운영 보안**
  + Bastion Host를 통한 접근 관리
  + MFA(Multi-Factor Authentication) 적용으로 보안 강화
  + Ansible을 통한 보안 패치 및 정책 업데이트

## ****인증 및 권한 관리****

* **OAuth2.0 및 Keycloak 기반 인증**
  + 중앙 인증 서버(Keycloak)에서 OAuth2.0을 통한 Single Sign-On(SSO) 제공
  + Access Token 및 Refresh Token을 통한 세션 관리
* **RBAC(Role-Based Access Control) 기반 접근 통제**
  + 계정 그룹 별 권한 정의: Admin, Manager, User
  + 서비스 및 데이터베이스에 대한 최소 권한 원칙 적용
* **MFA(Multi-Factor Authentication) 적용**
  + OTP(One-Time Password) 생성기를 통한 이중 인증
  + 금융 거래 및 민감한 정보 접근 시 MFA 필수 적용

## ****성능 설계 전략****

* **성능 목표 정의 (TPS, Latency, Throughput)**
  + 목표 TPS: 10,000 TPS
  + Latency: 3초 이내 응답 시간
  + Throughput: 초당 10,000 트랜잭션 처리
* **최적화된 API 게이트웨이 구조**
  + Spring Cloud Gateway에 Rate Limiting 설정
  + Redis와 연동한 API 요청 캐싱 처리
  + Circuit Breaker를 통해 장애 전파 방지
* **대량 트랜잭션 처리 구조**
  + Kafka를 통한 비동기 메시지 큐 처리
  + 분산 트랜잭션 관리(TCC, SAGA 패턴) 적용
  + 대규모 데이터 조회 시 Elasticsearch 활용
* **비동기 처리 및 이벤트 스트리밍 최적화**
  + Kafka Stream과 Spark Streaming을 통한 실시간 처리
  + CQRS 패턴으로 Read와 Write 경로 최적화
  + 메시지 중복 방지를 위한 Exactly Once Processing 보장

# ****장애 대응 및 복구 전략****

## ****장애 탐지 및 대응 전략****

* **실시간 장애 탐지**
  + Prometheus를 이용한 주요 메트릭 수집 (CPU, Memory, Disk I/O, Network Latency)
  + Grafana 대시보드를 통해 실시간 장애 탐지 및 시각화
  + ELK Stack (Elasticsearch, Logstash, Kibana)으로 로그 모니터링
* **자동 알림 설정**
  + AlertManager와 Slack 연동을 통한 즉각적인 알림 전송
  + 주요 장애 발생 시 이메일 및 SMS를 통한 관리자 통보
  + 장애 임계치 설정 (CPU 80%, Memory 70%, Disk I/O 85%)
* **자동 복구 프로세스**
  + Kubernetes의 **Self-Healing** 기능 활성화
  + Pod 장애 시 자동 재시작 및 상태 복구
  + ReplicaSet을 통한 이중화 및 무중단 서비스 유지

## ****장애 복구 및 Auto Healing 구조****

* **Auto Healing 구조**
  + HPA(Horizontal Pod Autoscaler)로 부하에 따른 동적 확장
  + VPA(Vertical Pod Autoscaler)로 리소스 최적화
  + Kubernetes Cluster Autoscaler로 노드 자동 추가
* **Failover 전략**
  + Multi-AZ(Availability Zone) 구조를 통한 이중화
  + 로드 밸런서를 활용한 트래픽 우회 처리
  + Nginx Ingress와 HAProxy를 통한 L4/L7 Failover
* **Rollback 및 복구 프로세스**
  + ArgoCD를 활용한 GitOps 기반 Rollback 처리
  + Helm Chart 버전 관리로 빠른 배포 복구
  + DataStage ETL 실패 시, 재처리 큐로 전송 후 복구 시도

## ****DR(Disaster Recovery) 설계****

* **DR 구축 전략**
  + Primary-Secondary 방식으로 DR 센터 운영
  + AWS Global Accelerator를 통한 글로벌 트래픽 관리
  + Active-Passive 구조에서의 데이터 동기화
* **복구 시나리오 정의**
  + 지역 장애(Region Failure) 시 DR 센터로 트래픽 자동 전환
  + Ceph 기반의 실시간 데이터 복제와 복구
  + DR 테스트 주기: 월 1회 정기 테스트
* **백업 및 복구 프로세스**
  + HDFS Snapshot 및 AWS S3 버킷 복제
  + Ceph RBD(Replicated Block Device) 기반의 실시간 백업
  + HDFS Incremental Backup을 통해 주기적 데이터 보존

## ****RPO/RTO 최적화 방안****

* **Recovery Point Objective (RPO)**
  + 데이터 유실 최소화를 목표로 15분 이내 복구
  + Kafka Streams와 HDFS Snapshot을 통해 데이터 일관성 유지
* **Recovery Time Objective (RTO)**
  + 서비스 복구 목표 시간: 10분 이내
  + Kubernetes의 Auto Healing과 Helm Rollback을 통한 신속 복구
* **최적화 방안**
  + Multi-Region 배포 및 Active-Passive 설정
  + DR 센터와의 실시간 Sync를 통한 무중단 서비스

## ****실시간 탐지와 Auto-Healing 구조 적용****

* **Self-Healing Mechanism**
  + Kubernetes Node Failure 감지 시 Pod 자동 복구
  + StatefulSet 기반의 애플리케이션 무중단 처리
* **Auto-Scaling 적용**
  + HPA로 트래픽 증가에 따른 실시간 확장
  + Node Auto Scaling으로 서버 자원 최적화
* **예측 기반 모니터링**
  + Prometheus의 AI 기반 예측 분석 활성화
  + 트래픽 패턴 분석을 통한 사전 리소스 증설

# ****가용성 및 확장성 설계****

## ****Multi-Region 아키텍처 설계****

* **Active-Active 및 Active-Passive 구성**
  + 주요 금융 서비스는 Active-Active 구조로 이중화하여 무중단 운영
  + 백엔드 데이터 스토리지는 Active-Passive 구조로 안정성 확보
  + Global Load Balancer를 활용하여 각 Region 간 트래픽 분산
* **Region 간 데이터 동기화**
  + AWS Global Accelerator를 통해 빠르고 안정적인 전송 보장
  + Ceph Replication과 HDFS Sync를 통해 실시간 데이터 동기화
  + 장애 시 DR(Disaster Recovery) 센터로 트래픽 자동 전환
* **멀티 클러스터 관리**
  + Rancher를 활용하여 Multi-Cluster 환경의 통합 관리
  + Prometheus와 Grafana로 멀티 리전의 메트릭 실시간 모니터링

## ****무중단 배포(Blue-Green, Canary Deployments)****

* **Blue-Green Deployment**
  + 두 개의 동일한 환경(Blue, Green)을 유지하며 배포 시 Zero Downtime
  + Green 환경에서 배포 후 정상 작동 확인 시 트래픽 전환
* **Canary Deployment**
  + 일부 트래픽만 신규 버전으로 전달하여 안전성 테스트
  + 장애 발생 시 즉시 이전 버전으로 롤백 가능
  + Spring Cloud Gateway와 API Gateway에서 트래픽 제어
* **Rolling Update**
  + Kubernetes Rolling Update를 통해 서비스 중단 없이 점진적 배포
  + ReplicaSet을 이용하여 순차적 업데이트 적용

## ****수평적 확장 구조(Horizontal Scaling)****

* **오토스케일링 정책 설정**
  + HPA(Horizontal Pod Autoscaler)로 CPU, Memory 사용량에 따라 동적 확장
  + VPA(Vertical Pod Autoscaler)로 리소스 최적화
  + 클러스터의 노드 수 증감을 AWS Auto Scaling Group과 연동
* **데이터베이스 수평 확장**
  + Sharding을 통한 데이터 분산 처리
  + Aurora Cluster Read Replica를 이용한 대량 조회 처리
* **Cache Layer 확장**
  + Redis Cluster와 Hazelcast를 활용한 캐싱 확장
  + 세션 클러스터링을 통해 사용자 세션 유실 방지

## ****오토스케일링 정책 및 자동 복구 전략****

* **Pod Auto Healing**
  + Kubernetes의 Self-Healing 기능을 통해 Pod 장애 시 자동 복구
  + Node Failure 시 다른 Node로 재배치
* **Node Auto Scaling**
  + AWS Auto Scaling Group을 통해 EC2 인스턴스 자동 증설
  + 클러스터 리소스 부족 시 Node 추가 배포
* **Ceph 및 HDFS 자동 복구**
  + Ceph의 Replicated Pool을 통한 데이터 이중화
  + HDFS의 Block Replication 설정으로 손실 방지
* **Disaster Recovery 연동**
  + DR 센터와의 실시간 Sync로 데이터 무결성 유지
  + RPO(15분 이내), RTO(10분 이내)를 보장

# ****운영 및 자동화 전략****

## ****운영 자동화 체계 개요****

* **Infrastructure as Code (IaC) 적용**
  + Terraform을 통한 인프라 프로비저닝 자동화
  + Kubernetes Cluster, VPC, Subnet 생성 및 관리
  + Ansible을 활용한 서버 설정 및 패치 관리
* **모니터링 및 로깅 자동화**
  + Prometheus와 Grafana를 통한 실시간 메트릭 수집
  + ELK Stack (Elasticsearch, Logstash, Kibana)으로 로그 관리
  + AlertManager와 Slack 연동을 통한 실시간 알림 전송
* **자동 복구 프로세스**
  + Kubernetes의 Self-Healing 기능으로 Pod 장애 시 자동 복구
  + ReplicaSet 및 StatefulSet을 통한 이중화 보장
  + Disaster Recovery 시나리오에 따른 자동 복구

## ****운영 자동화 전략****

* **CI/CD 파이프라인 최적화**
  + Jenkins와 ArgoCD를 통한 GitOps 기반 자동화
  + Maven 및 Gradle을 활용한 빌드 및 의존성 관리
  + SonarQube를 통한 코드 품질 분석
* **Blue-Green Deployment 및 Canary Release**
  + ArgoCD Rollout을 통한 무중단 배포
  + API Gateway 레벨에서 Canary Traffic Routing 설정
* **Configuration Management**
  + Spring Cloud Config를 활용한 중앙 설정 관리
  + Consul과 HashiCorp Vault로 보안 정보 저장 및 공유

## ****CI/CD 구조 전략적 설계****

* **Continuous Integration (CI)**
  + GitLab Runner를 통한 자동 빌드 및 테스트
  + SonarQube를 통한 정적 코드 분석 및 품질 관리
  + Docker 이미지 생성 후 Docker Hub에 자동 푸시
* **Continuous Deployment (CD)**
  + ArgoCD를 활용한 GitOps 기반의 실시간 배포
  + Helm Chart를 통해 Kubernetes 배포 관리
  + Canary Release 및 Blue-Green Deployment 전략 적용
* **자동화 테스트 전략**
  + JUnit, Mockito를 활용한 단위 테스트
  + Postman 및 RestAssured를 통한 API 테스트
  + JMeter를 통한 성능 및 부하 테스트

## ****GitOps 및 모니터링 연계****

* **GitOps 적용 전략**
  + Git Repository에 저장된 Manifest 파일을 ArgoCD가 실시간 동기화
  + Git Commit 시 자동 배포 트리거 설정
  + Rollback 및 Rollout 이력 관리
* **모니터링 연계 체계**
  + Prometheus와 Grafana를 통한 메트릭 실시간 시각화
  + ELK Stack으로 로그 통합 수집 및 분석
  + 주요 이벤트 발생 시 Slack 및 이메일 알림

## ****운영배포 체계 및 조직 연계 전략****

* **DevOps 운영 체계 확립**
  + 개발팀, 운영팀, DevOps 팀 간 협업 구조 최적화
  + DevSecOps 적용을 통한 보안 검증 자동화
  + 매일 자동화된 테스트 및 빌드 결과 공유
* **배포 프로세스 최적화**
  + Rolling Update와 Blue-Green 전략을 활용한 안정적 배포
  + 주요 금융 이벤트 전후 Pre-Deployment Test 수행
* **장애 대응 프로세스**
  + 실시간 AlertManager 연동으로 장애 감지
  + Prometheus Alert Rule 설정을 통한 자가 복구

## ****운영 품질 보증 방안****

* **SLA (Service Level Agreement) 관리**
  + 가용성: 99.99% 이상 보장
  + 응답 시간: 3초 이내
  + 데이터 복구 시간: 10분 이내
* **운영 품질 모니터링**
  + Prometheus를 활용한 실시간 Health Check
  + Kibana Dashboard를 통한 트래픽 및 장애 시각화
* **정기 점검 및 유지보수 계획**
  + 월간 보안 업데이트 및 취약점 분석
  + 주간 트래픽 분석 및 스케일링 최적화

# ****전략적 차별화 포인트****

## ****핵심 전략 차별화 포인트 요약표****

|  |  |  |
| --- | --- | --- |
| 구분 | 차별화 요소 | 주요 특징 |
| **기술 프레임워크** | Spring Cloud, Kafka, Kubernetes 기반 MSA | 실시간 데이터 처리 및 확장성 확보 |
| **성능 최적화** | Kafka Streams, Redis 캐싱 구조 | 대용량 데이터 처리 및 실시간 응답성 보장 |
| **보안 강화** | OAuth2.0, Keycloak, JWT 인증 | API 보안 강화 및 데이터 접근 통제 강화 |
| **운영 자동화** | GitOps 기반 CI/CD 파이프라인 | 무중단 배포 및 자동화된 운영 관리 |
| **데이터 관리** | Data Lake와 Data Mart 분리 | 실시간 분석과 장기 데이터 관리 병행 |
| **장애 대응** | Auto Healing 및 Self-Healing 클러스터 구성 | 장애 발생 시 자동 복구 및 서비스 연속성 보장 |

## ****기술 프레임워크 기반 경쟁우위****

* **MSA 기반 모듈화**
  + Spring Cloud와 Spring Boot를 활용하여 서비스 간 결합도 최소화
  + API Gateway를 이용하여 트래픽 관리 및 서비스 가용성 강화
* **실시간 데이터 처리 최적화**
  + Kafka를 활용하여 초당 10,000 TPS 이상의 데이터 처리 성능 확보
  + Spark Streaming을 통한 실시간 데이터 분석
* **데이터 통합 관리**
  + Data Lake와 Data Mart를 활용하여 데이터 수집, 정제, 분석의 일관성 유지
  + HDFS와 S3를 통한 대용량 비정형 데이터 저장 및 관리

## ****제안 설계 실행력 기반 차별화 포인트****

* **GitOps 기반 CI/CD 구조**
  + ArgoCD를 통해 선언형 배포 관리
  + Git 기반 버전 관리로 환경 일관성 유지
* **DevSecOps 도입으로 보안 검증 자동화**
  + Jenkins 파이프라인에 SonarQube와 Trivy를 포함하여 코드 보안 검토
  + CI/CD와 통합된 취약점 스캔으로 신속한 보안 이슈 대응
* **지속적 성능 모니터링 체계**
  + Prometheus와 Grafana로 실시간 메트릭 모니터링
  + Kafka Lag 모니터링을 통한 데이터 처리 지연 감지

## ****경쟁사 대비 차별화 요소****

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | NH 농협지주 데이터 플랫폼 | 경쟁사 A | 경쟁사 B |
| **데이터 처리 성능** | Kafka와 Spark 기반 실시간 데이터 스트리밍 | 기존 배치 처리 중심 | 실시간 처리 미흡 |
| **아키텍처 구조** | MSA 기반 확장성 보장 | Monolithic 구조 | Hybrid 구조 (일부 MSA 도입) |
| **보안 체계** | OAuth2.0, RBAC, E2E 암호화 | 단일 인증 체계 | 복잡한 인증 정책 |
| **운영 자동화** | GitOps 및 ArgoCD 기반 운영 자동화 | 수동 배포 및 모니터링 | 일부 자동화 도입 |
| **장애 복구** | Auto Healing 및 Multi-Region 구성 | 단일 데이터 센터 운영 | DR 구조 미비 |

## ****고객 특화 전략****

* **금융 데이터에 특화된 데이터 분석 플랫폼**
  + 고객 금융 거래 분석을 통해 맞춤형 상품 제안
  + 고객 이탈 예측 모델 기반의 리스크 관리
* **개인화 마케팅 강화**
  + AI/ML 기반의 타겟 마케팅 모델 적용
  + 실시간 데이터 분석으로 캠페인 성과 모니터링
* **API 중심 서비스 구조**
  + 기존 레거시 시스템과의 무중단 연계 구조
  + API 게이트웨이를 활용한 실시간 데이터 제공

## ****POC 기반 실행 설득 전략****

* **POC(Proof of Concept) 단계별 진행**
  + 1단계: MSA 기반 API Gateway 구축 (Spring Cloud Gateway)
  + 2단계: Kafka를 활용한 실시간 데이터 스트리밍 처리 검증
  + 3단계: Data Lake 기반 대용량 데이터 저장 및 분석 성능 검증
* **POC 검증 지표**
  + TPS 성능: 10,000 TPS 이상 처리 가능 여부
  + 데이터 처리 시간: 실시간 분석 지연 시간 1초 이내
  + 장애 복구 시간: 5분 이내 복구 완료 여부
* **POC 결과 보고 체계**
  + 실시간 대시보드로 POC 성과 시각화
  + 성능 지표와 장애 발생 시 복구 테스트 결과 포함

# 제안 아키텍처 검증 및 점검 체계

## ****아키텍처 검토 체크리스트****

|  |  |  |
| --- | --- | --- |
| **검토 항목** | **주요 내용** | **검증 방법** |
| **MSA 구조 적정성** | MSA 기반의 도메인 설계와 서비스 간 독립성 | Spring Cloud Config, API Gateway |
| **데이터 흐름 및 통합 구조** | Kafka, Data Lake, Data Mart 간의 데이터 연동 | Kafka Streams 및 ELK 모니터링 |
| **보안 구조** | OAuth2.0, Keycloak, RBAC 구조의 적정성 | Security Scan (SonarQube, Trivy) |
| **CI/CD 파이프라인 최적화** | ArgoCD 및 Jenkins를 통한 GitOps 배포 전략 | GitLab CI/CD Pipeline 테스트 |
| **장애 대응 및 복구 전략** | Auto Healing, Multi-Region DR 전략 | DR 테스트 시나리오 실행 |
| **모니터링 및 로깅 체계** | Prometheus, Grafana, ELK Stack 연동 | 실시간 대시보드 확인 |
| **API 연동 및 인터페이스 표준화** | REST API 표준 설계 및 Versioning 관리 | Postman 및 Swagger 검증 |

## ****유형별 검증 체크리스트****

|  |  |  |  |
| --- | --- | --- | --- |
| 유형 | 검증 항목 | 검증 기준 | 도구/기술 |
| **성능 검증** | API 응답 시간, TPS 처리량 | 3초 이내 응답, 10,000 TPS | JMeter, Locust, Prometheus |
| **보안 검증** | 데이터 암호화, 전송 구간 보안 | AES-256, TLS 1.3 적용 | SonarQube, Trivy, Nessus |
| **장애 대응 검증** | 장애 복구 시간, Auto Healing 동작 | 10분 이내 복구 | Kubernetes Self-Healing, ELK |
| **연계 검증** | 내부 API 및 외부 시스템 연동 | Kafka Streams, RabbitMQ | Postman, WireMock, REST Assured |
| **데이터 일관성 검증** | Data Lake와 Data Mart의 데이터 정합성 | HDFS와 S3 간의 데이터 동기화 | Apache Spark, HiveQL |
| **확장성 검증** | Horizontal Scaling, Multi-Region | Node 증가 시 Latency 최소화 | Kubernetes HPA, AWS Auto Scaling |

## ****기술 발표 준비 및 QA 정리****

* **발표 전략**
  + 주요 아키텍처 설계의 차별성 및 경쟁 우위 강조
  + 실시간 데이터 처리 구조(Kafka, Spark Streaming) 시연
  + GitOps 기반의 무중단 배포 전략 설명 (ArgoCD)
  + Multi-Region DR 구조의 무결성 및 가용성 보장 사례
* **질의응답 예상 항목**
  + Q1: MSA 구조에서의 데이터 일관성 문제는 어떻게 해결합니까?
    - A1: SAGA 패턴과 Eventual Consistency를 통해 서비스 간 데이터 정합성을 유지합니다.
  + Q2: 장애 시 DR 센터로의 전환 시간은 어느 정도입니까?
    - A2: Active-Passive 구조로 10분 이내 전환을 목표로 설계하였습니다.
  + Q3: 금융 데이터 보안은 어떻게 보장합니까?
    - A3: AES-256 암호화와 TLS 1.3을 통한 전송 구간 보호, RBAC 기반의 접근 통제를 적용합니다.
* **시연 준비 항목**
  + API Gateway를 통한 트래픽 분산 테스트
  + Kafka 기반 실시간 데이터 처리 흐름 시연
  + Grafana 대시보드에서의 실시간 모니터링 현황

# 추진체계 및 조직 전략

## ****인력 투입 계획****

* **프로젝트 단계별 인력 투입 전략**
  + 요구사항 분석 단계: 아키텍트, 비즈니스 분석가 (BA), 데이터 엔지니어
  + 설계 단계: 소프트웨어 아키텍트, 데이터 아키텍트, 보안 아키텍트
  + 개발 단계: 프론트엔드 개발자, 백엔드 개발자, 데이터 엔지니어
  + 테스트 및 검증 단계: QA 엔지니어, 테스트 매니저
  + 운영 및 유지보수 단계: DevOps 엔지니어, 시스템 관리자
* **투입 인력 규모 (예시)**

|  |  |  |
| --- | --- | --- |
| 역할 | 인원 수 | 주요 책임 |
| 프로젝트 매니저 (PM) | 1명 | 프로젝트 일정 관리 및 전체 통제 |
| 아키텍트 (Architect) | 2명 | 시스템 아키텍처 설계 및 기술 결정 |
| 데이터 엔지니어 | 3명 | 데이터 수집, ETL 처리 및 저장소 설계 |
| 백엔드 개발자 | 4명 | API 개발 및 비즈니스 로직 구현 |
| 프론트엔드 개발자 | 2명 | 사용자 인터페이스 및 화면 개발 |
| DevOps 엔지니어 | 2명 | CI/CD 자동화 및 클러스터 관리 |
| QA 엔지니어 | 2명 | 테스트 계획 수립 및 품질 검증 |
| 보안 엔지니어 | 1명 | 보안 정책 수립 및 점검 |

## ****역할/책임/산출물 매트릭스****

|  |  |  |
| --- | --- | --- |
| 역할 | 주요 책임 | 주요 산출물 |
| **프로젝트 매니저** | 일정 관리, 위험 관리, 이해관계자 커뮤니케이션 | 프로젝트 일정표, 리스크 관리 보고서 |
| **아키텍트** | 시스템 아키텍처 설계, 기술 스택 결정, 표준 정의 | 아키텍처 설계서, 기술 명세서, 표준 가이드 |
| **데이터 엔지니어** | 데이터 모델링, ETL 파이프라인 개발 및 최적화 | 데이터 모델링 문서, ETL 설계 문서 |
| **백엔드 개발자** | API 개발, 비즈니스 로직 구현, 데이터베이스 연동 | API 설계서, 비즈니스 로직 명세서 |
| **프론트엔드 개발자** | UI/UX 개발, API 연동, 사용자 인터페이스 구현 | 화면 설계서, 사용자 매뉴얼 |
| **DevOps 엔지니어** | CI/CD 파이프라인 구축, Kubernetes 클러스터 관리 | 배포 파이프라인 명세서, Helm Chart |
| **QA 엔지니어** | 기능 테스트, 성능 테스트, 보안 테스트 | 테스트 계획서, 테스트 결과 보고서 |
| **보안 엔지니어** | 시스템 보안 검토, 취약점 분석 및 대응 | 보안 점검 보고서, 취약점 분석 결과 |

## ****교육 및 역량 강화 계획****

* **프로젝트 이해를 위한 교육**
  + MSA(Microservices Architecture) 이해와 구현 전략
  + Kafka를 활용한 실시간 데이터 스트리밍
  + Spring Cloud Gateway 기반 API 설계
* **기술 역량 강화를 위한 교육**
  + Kubernetes 클러스터 관리와 오토스케일링 최적화
  + DevOps와 GitOps를 활용한 배포 자동화 전략
  + Prometheus와 Grafana를 활용한 모니터링 및 로깅
* **보안 및 규제 대응 교육**
  + 금융 데이터 보호를 위한 보안 정책 이해 (OAuth2.0, Keycloak)
  + 정보보호 관리체계(ISMS)와 GDPR 준수 사항
* **정기적 세미나 및 기술 워크숍**
  + 매월 기술 세미나 진행 (내부 전문가 발표)
  + 매 분기 DevOps Best Practice 공유

# 레퍼런스 및 유사사례

## ****유사 구축 사례 분석****

|  |  |  |  |
| --- | --- | --- | --- |
| 프로젝트명 | 주요 기술 스택 | 특징 및 성공 포인트 | 구축 기간 |
| **K은행 데이터 플랫폼 구축** | Spring Boot, Kafka, Kubernetes | 금융거래 실시간 스트리밍, 대용량 데이터 분석, MSA 구조 | 2023년 6월 ~ 2024년 3월 |
| **H증권 빅데이터 플랫폼** | Spark, HDFS, Cloudera Manager | 실시간 리스크 분석, 사용자 패턴 추적, 고속 데이터 처리 | 2022년 4월 ~ 2023년 2월 |
| **S카드 마이데이터 구축** | Spring Cloud, Redis, RabbitMQ | 고객 맞춤형 데이터 제공, 실시간 마케팅 분석, 고성능 API | 2023년 1월 ~ 2023년 12월 |

* **K은행 데이터 플랫폼**
  + Spring Boot와 Kafka를 활용한 MSA 구조로 금융 거래의 실시간 스트리밍 처리
  + Multi-Region 배포를 통해 지역 장애에도 무중단 서비스 운영
  + ELK Stack을 통한 실시간 로그 모니터링과 장애 대응
* **H증권 빅데이터 플랫폼**
  + Spark와 HDFS를 통해 대용량 금융 데이터를 실시간 분석
  + Cloudera Manager를 통해 배치 작업 최적화 및 스케일링 대응
  + 고객 리스크 분석을 통한 사전 대응 시나리오 구축
* **S카드 마이데이터 구축**
  + Spring Cloud Gateway를 통한 API Gateway 구조 확립
  + Redis Cache를 활용하여 고속 응답 처리 최적화
  + RabbitMQ를 통한 실시간 이벤트 메시징 관리

## ****주요 레퍼런스 사례 정리****

|  |  |  |
| --- | --- | --- |
| 고객사 | 구축 솔루션 | 주요 특징 및 기대효과 |
| **A은행** | Microservices 기반 인터넷 뱅킹 시스템 | 금융 거래 처리 성능 향상, 실시간 데이터 분석 도입 |
| **B증권** | Data Lake 기반 투자 분석 플랫폼 | 투자 리스크 분석, 고속 조회 서비스 제공 |
| **C보험사** | 고객 360 뷰 통합 플랫폼 | 고객 데이터 통합 관리, 개인화 마케팅 강화 |

* **A은행 인터넷 뱅킹 시스템**
  + MSA(Microservices Architecture)로 시스템을 모듈화하여 확장성 및 유지보수 용이성 증대
  + Spring Cloud Config와 API Gateway를 통한 중앙 집중형 관리 체계
  + 실시간 금융 거래 추적과 이상 감지 시스템 구축
* **B증권 투자 분석 플랫폼**
  + Spark와 Kafka를 통해 투자 데이터를 실시간으로 수집 및 분석
  + Data Lake 기반으로 장기 보존 및 실시간 조회 지원
  + 투자 전략 최적화를 위한 Machine Learning 모델 도입
* **C보험사 고객 360 뷰 통합 플랫폼**
  + 고객 정보를 통합 관리하여 마케팅 전략 최적화
  + AI 기반 고객 행동 분석을 통해 상품 추천 정확도 향상
  + 대규모 고객 데이터를 효율적으로 관리하기 위한 클러스터링 구조 도입

## ****고객사 적용사례와 기대효과****

* **고객사 적용 사례**
  + NH 농협지주의 데이터 통합 플랫폼에 적용 시, 기존 금융 데이터의 실시간 분석과 중앙 집중화가 가능
  + 마이데이터 사업 확장 시, 고객의 금융 데이터를 통합하여 고객 맞춤형 서비스 제공
  + 신속한 API 응답 처리와 실시간 금융 트랜잭션 분석이 가능
* **기대효과**
  + 금융 데이터의 통합 관리로 업무 효율성 증대
  + 실시간 트랜잭션 처리로 고객 응답 시간 단축
  + 금융 리스크 분석 및 대응 속도 향상
  + API Gateway와 MSA 구조를 통한 확장성 보장

# 부록

## ****제품 카탈로그****

|  |  |  |  |
| --- | --- | --- | --- |
| 제품명 | 주요 기능 | 제조사 | 버전 |
| **Spring Boot** | MSA 기반의 독립 서비스 개발 | Pivotal | 2.6.x |
| **Kafka** | 실시간 데이터 스트리밍 처리 | Apache | 3.0.x |
| **Kubernetes** | 컨테이너 오케스트레이션 및 스케일링 | CNCF | 1.23.x |
| **DataStage** | ETL 작업 자동화 및 데이터 파이프라인 | IBM | 11.7 |
| **Redis** | 캐싱 및 세션 클러스터링 | Redis Labs | 6.x |
| **Grafana** | 실시간 모니터링 대시보드 | Grafana Labs | 8.x |
| **Prometheus** | 메트릭 수집 및 경보 처리 | CNCF | 2.x |
| **ArgoCD** | GitOps 기반의 지속적 배포 관리 | CNCF | 2.1.x |

## ****질의응답서****

|  |  |
| --- | --- |
| 질문 내용 | 답변 내용 |
| **Q1. MSA 기반 아키텍처에서 데이터 일관성은 어떻게 유지하나요?** | SAGA 패턴 및 Eventual Consistency를 적용하여 서비스 간 데이터 일관성 보장 |
| **Q2. 금융 거래 시 데이터 보안은 어떻게 보장하나요?** | AES-256 암호화 및 TLS 1.3 보안을 통해 안전한 전송을 보장합니다. |
| **Q3. Kubernetes 클러스터 장애 발생 시 대응 전략은 무엇인가요?** | Auto Healing과 Multi-AZ 구조를 통해 무중단 서비스를 유지합니다. |
| **Q4. CI/CD 파이프라인의 무중단 배포는 어떻게 수행되나요?** | ArgoCD와 Blue-Green, Canary Deployment를 활용합니다. |
| **Q5. API Gateway에서의 보안 정책은 어떻게 설정되나요?** | OAuth2.0과 Keycloak을 활용하여 인증 및 권한 관리를 수행합니다. |

## ****인증서 보안문서****

* **ISO/IEC 27001:2013** - 정보보호 관리체계 인증
* **ISMS-P** - 금융 보안 및 개인정보보호 인증
* **FISC 금융보안기준** - 금융기관의 정보시스템 보안 지침 준수
* **PCI-DSS** - 금융 결제 시스템의 보안 인증

## ****주요 API 명세서****

|  |  |  |  |
| --- | --- | --- | --- |
| API 명칭 | HTTP Method | Endpoint | 설명 |
| **고객 정보 조회 API** | GET | /api/v1/customer/{customerId} | 고객 기본 정보 조회 |
| **계좌 거래 내역 조회 API** | GET | /api/v1/account/{accountId}/transactions | 특정 계좌의 거래 내역 조회 |
| **고객 데이터 업데이트 API** | PUT | /api/v1/customer/{customerId} | 고객의 개인정보 업데이트 |
| **대출 신청 API** | POST | /api/v1/loan/apply | 고객 대출 신청 처리 |
| **마케팅 캠페인 조회 API** | GET | /api/v1/marketing/campaigns | 진행 중인 마케팅 캠페인 목록 조회 |

## ****시스템 구성도 및 네트워크 구조****

* **네트워크 구성도**
  + Private Cloud (VMware)와 퍼블릭 클라우드 (AWS) 혼합 사용
  + VPC와 Subnet을 통한 네트워크 격리
  + API Gateway를 중심으로 내부망 및 외부망 분리
* **시스템 아키텍처 구성도**
  + Application Layer: Spring Boot, Spring Cloud
  + Integration Layer: Kafka, RabbitMQ, API Gateway
  + Data Layer: Oracle DB, DataStage, HDFS
  + Monitoring Layer: Prometheus, Grafana, ELK Stack
* **데이터 흐름 구조**
  + 실시간 데이터: Kafka → Spark Streaming → Data Lake
  + 배치 데이터: DataStage → Data Mart → 분석 플랫폼

# ■■■

# 프로젝트 개요

## 도입 전 안내말

NH 금융지주의 통합 데이터 플랫폼 구축은 기존의 시스템 인프라를 확장하여 실시간 분석, 안정적인 거래 처리, 확장성 있는 마이크로서비스 아키텍처를 실현하기 위한 목적으로 설계되었습니다.  
본 아키텍처는 고성능 하드웨어, 최신 소프트웨어, 클라우드 네이티브 인프라를 활용하여 금융 거래의 안정성과 확장성을 보장합니다.  
특히, Private Cloud 기반의 Kubernetes 클러스터, Data Lake 통합 관리, Microservices 기반의 금융 서비스 처리를 통해 고객 중심의 금융 서비스 제공과 디지털 금융 전환을 가속화합니다.

## 프로젝트 개요

### 로젝트 명칭

* NH 농협지주 통합 데이터 플랫폼 구축 사업

### 대상 계열사

* NH농협은행
* NH농협카드
* NH투자증권
* NH생명보험
* NH손해보험

### 주요 요구사항 정의

|  |  |
| --- | --- |
| 구분 | 요구사항 |
| 데이터 수집 | 계열사 데이터 배치 수집, 올원뱅크 행동 데이터 수집, 표준 인터페이스 적용 |
| 데이터 처리 | 파기 대상 데이터 삭제, 정기 배치 이력 로깅, 암호화/마스킹/파기 처리 |
| 데이터 저장 | 원천 데이터 적재, 목적별 마트 개발, 통합 고객번호 채번, 데이터 보관 관리 |
| 데이터 분석 | OLAP 환경 제공, 메타데이터 관리, 품질 관리, 계열사 권한 관리 |
| 마케팅 분석 | 캠페인 설정 및 실행, 실시간 EBM 지원, 개인화 마케팅 실행, 이벤트 실행 |
| 데이터 제공 | 타깃 고객 추출 및 제공, 분석 DB 구성, 데이터 포털 제공 |

### 추진 배경 및 목적

NH 금융지주는 고객 중심의 금융 서비스와 데이터 기반 의사결정을 강화하기 위해 통합 데이터 플랫폼을 도입합니다.  
이를 통해 분산된 데이터를 중앙에서 통합 관리하고, 실시간 분석 및 마케팅 최적화가 가능하도록 설계하였습니다.  
주요 목표는 다음과 같습니다:

1. **실시간 거래 처리 및 분석**: 분산 데이터의 실시간 분석을 통해 금융 거래의 신속한 처리가 가능하도록 설계합니다.
2. **데이터 거버넌스 및 품질 관리**: 계열사 간 데이터 품질을 일관성 있게 유지하고 보안성을 강화합니다.
3. **확장성과 유연성 보장**: 클라우드 네이티브 아키텍처를 기반으로 시스템 확장이 용이합니다.
4. **마케팅 최적화**: 통합 고객 정보를 기반으로 타겟 마케팅을 실시간으로 실행합니다.

### 주요 추진 전략

1. **클라우드 네이티브 기반의 MSA 도입**
   * Private Cloud 및 Kubernetes 기반의 확장성 있는 MSA 구조 설계
   * Spring Boot, Kafka, Spring Cloud Gateway를 활용한 비동기 처리 최적화
2. **Data Lake 및 분석 마트 통합**
   * HDFS 및 AWS S3 기반의 대용량 데이터 저장
   * 실시간 스트리밍 분석 및 Data Mart 구축
3. **보안 강화를 위한 E2E Encryption 및 RBAC 적용**
   * 데이터 전송 구간 및 저장 시 AES-256 암호화 적용
   * Role-Based Access Control (RBAC) 기반의 사용자 접근 관리
4. **실시간 분석 및 캠페인 관리 최적화**
   * Spark Streaming과 Kafka를 활용한 실시간 이벤트 분석
   * 마케팅 캠페인 실행 및 성과 분석을 위한 OLAP 연계
5. **통합 데이터 포털 구축**
   * 메타 데이터 관리, 품질 관리, 데이터 카탈로그 제공
   * 계열사 간 데이터 접근 권한 및 관리 기능 제공

# ****아키텍처 설계****

## ****기술 아키텍처 (TA)****

본 문서는 전사 EA 표준 사업을 위한 솔루션 표준 프로파일을 도식화하고 상세에 대해 기술한 문서이다.

### ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼 구축은 기존의 시스템 인프라를 확장하여,  
**실시간 분석, 안정적인 거래 처리, 확장성 있는 마이크로서비스 아키텍처**를 실현하기 위한 목적으로 설계되었습니다.  
본 아키텍처는 **고성능 하드웨어, 최신 소프트웨어, 클라우드 네이티브 인프라**를 활용하여 금융 거래의 안정성과 확장성을 보장합니다.

### ****인프라 환경 정의****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **구성 요소** | **기술 스택** | **설명** |
| **클라우드 환경** | Private Cloud | OpenStack, VMware | NH 금융지주 전용 클라우드 환경 구성, 고가용성 및 보안성 보장 |
| **가상화 기술** | Hypervisor | KVM, VMware ESXi | 서버 자원의 효율적 사용 및 격리된 환경 제공 |
| **컨테이너 오케스트레이션** | Kubernetes | OpenShift, Rancher | MSA 아키텍처 지원 및 자동 확장 설정 |
| **스토리지** | Distributed Storage | Ceph, GlusterFS | 대용량 데이터 처리 및 고가용성 보장 |
| **네트워크 구성** | Software Defined Network | Calico, Flannel | 동적 네트워크 설정 및 마이크로서비스 간 트래픽 관리 |
| **로드 밸런싱** | L4, L7 Load Balancer | HAProxy, Nginx | 트래픽 분산 처리 및 무중단 서비스 제공 |
| **보안 네트워크** | Firewall, VPN | Fortinet, Cisco ASA | 외부 공격 방어 및 데이터 전송 보안 |

### ****HW, SW, 인력 계획****

|  |  |  |
| --- | --- | --- |
| **구분** | **내용** | **비고** |
| **하드웨어 계획** | - 256 Core, 1TB RAM, 40TB Storage (Cluster 단위 구성) | 연간 확장 계획 반영 |
| **소프트웨어 계획** | - OpenShift, Kubernetes, Docker, VMware, Ceph | 오픈소스 기반 최적화 |
| **네트워크 장비** | - Cisco Nexus 9000 시리즈, FortiGate Firewall | 보안 및 트래픽 최적화 |
| **인력 계획** | - 시스템 엔지니어 5명, 네트워크 엔지니어 2명, 보안 전문가 1명 | DevOps 포함 |

### ****예산 계획****

|  |  |  |
| --- | --- | --- |
| **항목** | **설명** | **비용(백만 원)** |
| **인프라 구축 비용** | 서버 및 네트워크 장비 구매 | 2,000 |
| **소프트웨어 라이선스** | VMware, RedHat, FortiGate 라이선스 | 800 |
| **컨테이너 오케스트레이션** | Kubernetes 및 OpenShift 클러스터 구성 | 600 |
| **스토리지 구축** | Ceph 및 Distributed Storage 도입 | 500 |
| **보안 장비 도입** | Firewall, VPN 구축 | 300 |
| **기타 비용** | 유지보수 및 운영 인력 배치 비용 | 200 |
| **총 비용** |  | **4,400** |

### ****스토리지 및 임시 보관 영역 설계****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **스토리지 타입** | **기술 스택** | **설명** |
| **단기 보관 영역** | SSD 기반 스토리지 | NVMe, Ceph | 실시간 트랜잭션 처리 및 분석을 위한 고속 데이터 저장소 |
| **장기 보관 영역** | Object Storage | AWS S3, GlusterFS | 장기 보관이 필요한 금융 데이터 (10년 이상 보관 기준) |
| **백업 및 복구 영역** | Snapshot 및 DR 센터 저장소 | AWS S3, HDFS | 스냅샷 기반의 복구 전략 적용, 재해 발생 시 복구 시간 15분 이내 목표 |
| **ETL 처리 영역** | Distributed File System | HDFS, GFS | 배치 처리 및 대용량 파일 전송 최적화 |

### ****설계 및 구현 방안****

#### ****인프라 구축 단계****

1. **Private Cloud 환경 구성:**
   * OpenStack 기반의 Private Cloud를 구축하여 금융 데이터 보안성 강화
   * 네트워크 분리 및 DMZ 설정으로 외부 침입 차단
2. **컨테이너 오케스트레이션 구성:**
   * Kubernetes 클러스터를 구성하여 Microservice 기반의 확장성을 확보
   * Auto Scaling 설정으로 트래픽 증가 시 자동 확장
3. **스토리지 최적화:**
   * Ceph를 통해 이중화된 분산 스토리지 구성
   * Object Storage(S3)를 활용하여 장기 보관 데이터 관리
4. **네트워크 및 로드 밸런싱 설정:**
   * HAProxy 및 Nginx를 통해 L4/L7 레벨의 로드 밸런싱 처리
   * SDN(Software Defined Network)을 통해 동적 네트워크 구성
5. **보안 설계:**
   * FortiGate Firewall과 Cisco ASA를 통해 네트워크 방화벽 설정
   * VPN 설정으로 외부 접근 시 보안 터널링 처리

## ****애플리케이션 아키텍처 (AA)****

### ****도입 전 안내말****

NH 금융지주의 애플리케이션 아키텍처는 **Microservice Architecture**를 중심으로 설계되었으며,  
각 금융 서비스는 독립적인 마이크로서비스로 구성되어 유연성과 확장성을 극대화하였습니다.  
또한, 실시간 금융 거래 처리와 대량 데이터 수집 및 분석이 가능하도록 **Spring Boot**, **Kafka**,  
**Spring Cloud Gateway**를 통해 분산된 트랜잭션 관리와 안정적인 인터페이스를 제공합니다.

### ****시스템 연계 및 수집 프로세스****

#### ****시스템 연계 구조****

|  |  |  |
| --- | --- | --- |
| **연계 대상 시스템** | **주요 기능** | **프로토콜 및 기술 스택** |
| **NH 농협은행** | - 금융 거래 처리 - 고객 정보 조회 - 캠페인 데이터 수집 | REST API, FEP, MCA |
| **NH 농협카드** | - 카드 결제 정보 연계 - 마케팅 데이터 수집 - 실시간 결제 승인 | REST API, ISO 8583 |
| **NH 투자증권** | - 금융 투자 내역 관리 - 거래 내역 분석 | FIX Protocol, MQ |
| **NH 생명보험** | - 보험 계약 관리 - 보험금 청구 및 결제 처리 | SOAP, REST API |
| **NH 손해보험** | - 손해보험 계약 및 청구 관리 - 고객 정보 동기화 | SOAP, FEP |
| **대외 금융기관** | - 공공기관 연계, 금융감독원 보고 - KYC 인증, AML 규제 대응 | SFTP, MFT, WebSphere MQ |
| **외부 데이터 제공자** | - 실시간 환율 정보, 대출 금리 정보 제공 | REST API, WebSocket |
| **BI 및 분석 플랫폼** | - 고객 행동 분석, 리스크 관리, 금융 상품 분석 | Spark, Tableau, PowerBI |

#### ****데이터 수집 프로세스 설계****

|  |  |  |
| --- | --- | --- |
| **수집 데이터 유형** | **수집 방식** | **기술 스택** |
| **금융 거래 정보** | - 실시간 거래 연동 (REST API) - 배치 처리 (DataStage) | Spring Boot, DataStage |
| **고객 행동 데이터** | - 올원뱅크 및 인터넷 뱅킹에서 실시간 수집 | Kafka, Spring Cloud |
| **카드 결제 정보** | - NH 농협카드 결제 트랜잭션 실시간 수집 | WebSphere MQ, ISO 8583 |
| **대외 금융기관 정보** | - 공공 데이터 포털 및 금융감독원에서 배치 수집 | MFT, WebSphere MQ |
| **환율 및 금융 지수 정보** | - 외부 데이터 제공자의 API를 통해 실시간 수집 | REST API, WebSocket |
| **보험 계약 및 청구 정보** | - SOAP 기반 인터페이스를 통해 실시간 조회 | SOAP, JMS |
| **투자 및 증권 거래 정보** | - FIX Protocol을 통해 실시간 거래 내역 수집 | FIX, MQ |
| **메타 데이터 수집** | - 각 데이터 항목의 스키마 및 변경 내역 추적 | Schema Registry, Kafka |

### ****설계 및 구현 방안****

#### ****애플리케이션 계층 구조 설계****

[Channel Layer]

- Internet Banking

- Mobile Banking

- API Gateway (Spring Cloud Gateway)

|

[Interface Layer]

- FEP, MCA, MFT

|

[Business Logic Layer]

- Microservices (Spring Boot 기반)

- Customer Service

- Transaction Service

- Marketing Service

- Insurance Service

- Securities Service

- Campaign Service

- Analytics Service

|

[Data Access Layer]

- Oracle DB

- MySQL DB

- HDFS (Data Lake)

- AWS S3 (장기 보관)

|

[Integration Layer]

- Kafka Streaming

- ETL (DataStage)

- Spark Streaming

|

[Monitoring & Logging Layer]

- Prometheus

- Grafana

- ELK Stack (Elasticsearch, Logstash, Kibana)

#### ****주요 서비스 설계****

|  |  |  |
| --- | --- | --- |
| **서비스 명** | **기능 설명** | **기술 스택** |
| **Customer Service** | - 고객 정보 조회 및 수정 - 고객 등급 관리 | Spring Boot, MySQL |
| **Transaction Service** | - 실시간 거래 처리 - 결제 승인 및 전송 | Spring Boot, Oracle DB |
| **Marketing Service** | - 캠페인 설정 및 실행 - 고객 대상 타겟 마케팅 실행 | Spring Boot, Kafka |
| **Insurance Service** | - 보험 계약 관리 - 보험금 청구 처리 | Spring Boot, SOAP |
| **Securities Service** | - 투자 상품 거래 내역 관리 - 실시간 시세 조회 | Spring Boot, FIX Protocol |
| **Campaign Service** | - 고객 캠페인 생성 및 참여 관리 - 이벤트 기반 마케팅 실행 | Spring Boot, Kafka |
| **Analytics Service** | - 고객 행동 분석 - 리스크 예측 및 금융상품 추천 | Spark, Tableau, PowerBI |

#### ****트랜잭션 처리 설계****

* 모든 금융 거래는 **ACID (Atomicity, Consistency, Isolation, Durability)** 보장을 위한 설계
* Spring Boot에서 JTA (Java Transaction API)를 사용하여 **2PC (Two-Phase Commit)** 처리
* **Kafka**를 통해 트랜잭션 로그를 비동기적으로 저장하고, 장애 발생 시 복구 가능

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **단일 거래 처리** | - API 요청 시 트랜잭션 시작 → DB Commit or Rollback | Spring Boot, Oracle |
| **다중 거래 처리** | - 복수 계좌 간 거래 처리 시 2PC 적용 | JTA, Spring Data |
| **비동기 메시지 처리** | - Kafka Producer를 통해 비동기 메시지 전송 | Kafka, Spring Boot |
| **트랜잭션 로그 저장** | - HDFS 및 Kafka에 트랜잭션 로그 저장 | HDFS, Kafka |

#### ****장애 복구 및 DR(Disaster Recovery) 전략****

* 각 Microservice는 **Kubernetes Auto Healing**을 통해 장애 발생 시 자동 복구
* DB는 \*\*Oracle RAC (Real Application Clusters)\*\*를 통해 Active-Standby 구조로 무중단 전환
* Kafka 메시지 로그는 HDFS와 AWS S3에 실시간 백업
* DR 센터에 **15분 이내** 복구 목표로 설정

|  |  |  |
| --- | --- | --- |
| **복구 대상** | **복구 전략** | **복구 시간 목표 (RTO)** |
| **Microservice** | Kubernetes Auto Healing | 5분 이내 |
| **Database** | Oracle RAC, MySQL Replication | 10분 이내 |
| **Message Queue** | Kafka 스트림 복구, HDFS에서 재처리 | 3분 이내 |
| **ETL 작업** | DataStage Job 재시작 | 10분 이내 |

## ****비즈니스 아키텍처 (BA)****

### ****도입 전 안내말****

NH 금융지주의 비즈니스 아키텍처는 금융 거래와 마케팅 활동을 **개인화된 고객 중심 서비스**로 확장하기 위한 설계입니다.  
고객의 금융 거래 패턴, 마케팅 반응 데이터를 실시간으로 분석하여 최적의 금융 상품과 맞춤형 캠페인을 제공하는 것이 목표입니다.  
이를 통해, 고객 중심의 \*\*CRM(Customer Relationship Management)\*\*을 강화하고,  
실시간 캠페인 실행을 통해 고객 만족도를 극대화하는 아키텍처로 설계되었습니다.

### ****개인화 마케팅 서비스 설계****

#### ****개인화 마케팅 주요 서비스 정의****

|  |  |  |
| --- | --- | --- |
| **서비스 명** | **기능 설명** | **기술 스택** |
| **Target Marketing** | - 고객의 금융 거래 데이터를 분석하여 타겟 고객 선별 - 마케팅 캠페인 대상자 추출 | Spark, DataStage, Spring Boot |
| **Campaign Management** | - 마케팅 캠페인 생성 및 실행 - 이메일, 문자, 알림 메시지 발송 관리 | Spring Boot, Kafka, Redis |
| **Customer Segmentation** | - 고객의 행동 데이터를 기반으로 세분화 - 연령대, 금융상품 사용량, 거래 빈도별 분류 | Spark, Tableau, PowerBI |
| **Event Trigger Service** | - 특정 조건 발생 시 실시간 마케팅 이벤트 실행 - 예: 카드 결제 후 할인 쿠폰 발행 | Spring Cloud Gateway, Kafka |
| **Omni-Channel Integration** | - 모바일, 인터넷뱅킹, 영업점에서 동일한 캠페인 실행 및 관리 | Spring Boot, API Gateway |
| **Loyalty Program Service** | - 고객 리워드 및 포인트 관리 - 마케팅 참여도에 따른 보상 지급 | Spring Boot, Redis |
| **Performance Analysis** | - 캠페인 성과 분석 및 ROI(Return On Investment) 측정 | Tableau, PowerBI, Spark |

#### ****개인화 캠페인 실행 프로세스****

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 고객 세분화 분석** | - Spark Streaming을 통해 실시간으로 고객 데이터를 분석 - 금융 거래 패턴, 사용 빈도에 따른 분류 | Spark, DataStage |
| **2. 타겟 고객 추출** | - 특정 캠페인에 적합한 고객을 추출 - 예: 고액 자산가, 신용 카드 사용자 | Spark, Spring Boot |
| **3. 마케팅 캠페인 생성** | - Spring Boot로 캠페인 생성 - 웹 UI를 통해 마케팅 팀이 설정 | Spring Boot, Redis |
| **4. 실시간 캠페인 실행** | - Kafka 이벤트를 통해 실시간 메시지 전송 - Omni-Channel 통합 전송 | Kafka, Spring Cloud |
| **5. 응답 데이터 수집** | - 캠페인 참여 여부 및 반응 데이터를 실시간 수집 - Redis와 HDFS에 로그 기록 | Redis, HDFS |
| **6. 성과 분석 및 리포팅** | - Tableau와 PowerBI를 통해 캠페인 성과 분석 - 고객 반응 및 전환율 측정 | Tableau, PowerBI, Spark |

#### ****Omni-Channel 마케팅 설계****

* NH 금융지주의 모든 채널 (모바일, 인터넷뱅킹, 영업점)에서 **동일한 마케팅 메시지 전달**
* Spring Cloud Gateway를 활용하여 트래픽을 효율적으로 분배하고,  
  고객이 어느 채널을 통해 접속하든 동일한 캠페인 경험을 제공합니다.

|  |  |  |
| --- | --- | --- |
| **채널** | **설명** | **기술 스택** |
| **Mobile Banking** | - 실시간 푸시 알림, 캠페인 메시지 전송 | Firebase, Spring Boot |
| **Internet Banking** | - 로그인 시 마케팅 배너 표시, 할인 이벤트 노출 | Spring MVC, Redis |
| **Branch Service** | - 영업점에서 고객이 방문 시, 맞춤형 금융상품 안내 | Spring Boot, Redis |
| **ATM Service** | - ATM 사용 시 마케팅 메시지 출력 | Spring Boot, API Gateway |
| **Chatbot Service** | - 마케팅 관련 문의에 대해 실시간 응답 제공 | DialogFlow, Spring Boot |

### ****설계 및 구현 방안****

#### ****마케팅 캠페인 관리 시스템 구조****

[Channel Layer]

- Mobile Banking

- Internet Banking

- Branch Service

- ATM Service

- Chatbot Service

|

[API Gateway]

- Spring Cloud Gateway

|

[Campaign Management Layer]

- Target Marketing

- Customer Segmentation

- Event Trigger Service

- Loyalty Program Service

|

[Data Access Layer]

- Oracle DB

- Redis Cache

- HDFS (Data Lake)

- AWS S3 (장기 보관)

|

[Integration Layer]

- Kafka Streaming

- ETL (DataStage)

- Spark Streaming

|

[Monitoring & Logging Layer]

- Prometheus

- Grafana

- ELK Stack (Elasticsearch, Logstash, Kibana)

#### ****트랜잭션 관리 및 이벤트 트리거 설계****

* 캠페인 실행 시 이벤트 트리거가 발생하여 **Kafka Streaming**을 통해 실시간 데이터가 전송됩니다.
* 캠페인 메시지 발송은 **Spring Cloud Gateway**를 거쳐 다중 채널에 전달됩니다.
* 마케팅 데이터는 **Redis Cache**에 임시 보관되며, 장기 분석을 위해 **HDFS**로 저장됩니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **실시간 트리거 처리** | - 고객의 금융 거래 발생 시 자동으로 캠페인 실행 | Spring Boot, Kafka |
| **다중 채널 동기화** | - 모바일, 웹, 영업점에 동일한 마케팅 메시지 전달 | Spring Cloud Gateway |
| **로그 및 분석 처리** | - 캠페인 참여 및 응답 결과를 Redis에 캐싱 후 분석 처리 | Redis, HDFS |
| **데이터 분석 및 리포팅** | - Tableau를 통해 실시간 분석 대시보드 제공 | Tableau, Spark |

#### ****장애 복구 및 DR(Disaster Recovery) 전략****

* **Kubernetes Auto Healing**을 통해 마케팅 서비스가 중단 없이 실행됩니다.
* 캠페인 메시지는 **Kafka 스트림**에 기록되며, 장애 발생 시 복구 가능하도록 설계되었습니다.
* 마케팅 관련 데이터는 **HDFS**와 **AWS S3**에 백업되며, DR 센터에서 **15분 이내** 복구가 목표입니다.

|  |  |  |
| --- | --- | --- |
| **복구 대상** | **복구 전략** | **복구 시간 목표 (RTO)** |
| **Campaign Service** | Kubernetes Auto Healing | 5분 이내 |
| **Data Storage** | HDFS, AWS S3, Redis Cache 복구 | 10분 이내 |
| **Message Queue** | Kafka 스트림 복구, HDFS에서 재처리 | 3분 이내 |

## ****데이터 아키텍처 (DA)****

### ****4.1 도입 전 안내말****

NH 금융지주의 데이터 아키텍처는 **전사 데이터를 통합 관리하고 실시간 분석이 가능한 구조**로 설계되었습니다.  
특히 금융 거래, 고객 정보, 마케팅 데이터를 **Data Lake**에 통합하여,  
\*\*HDFS(Hadoop Distributed File System)\*\*와 **AWS S3**를 기반으로 대용량 데이터 처리와  
장기 보관이 가능하도록 구성하였습니다.  
또한, **ETL 프로세스**를 통해 정형/비정형 데이터를 목적별 마트로 분리하고,  
Spark 기반의 실시간 분석이 가능하도록 최적화되었습니다.

### ****데이터 주제 영역 정의****

|  |  |  |
| --- | --- | --- |
| **주제 영역** | **설명** | **구성 요소** |
| **고객 정보** | - 고객의 기본 정보 및 거래 이력 관리 - 신용 등급 및 마케팅 수신 동의 정보 | 고객 ID, 이름, 생년월일, 주소, 연락처, 신용 등급 |
| **상품 정보** | - 금융 상품에 대한 기본 정보 및 조건 관리 | 상품 ID, 상품명, 이자율, 만기일 |
| **거래 정보** | - 예금, 대출, 카드 결제, 투자 내역 등 금융 거래 정보 관리 | 거래 ID, 거래 일자, 거래 금액, 거래 유형 |
| **채널 정보** | - 인터넷 뱅킹, 모바일 뱅킹, 영업점 거래 이력 | 채널 ID, 접속 일시, 거래 유형 |
| **마케팅 정보** | - 캠페인 참여 이력 및 반응 정보 관리 | 캠페인 ID, 참여 일자, 응답 여부 |
| **분석 정보** | - 금융 리스크, 마케팅 성과 분석 결과 | 분석 모델 ID, 예측 값, 리스크 지표 |
| **메타 데이터** | - 각 데이터의 스키마 및 변경 이력 관리 | 테이블명, 필드명, 데이터 타입, 변경 일시 |

### ****원천 데이터 적재 및 목적별 마트 구성****

#### ****원천 데이터 적재 구조****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **저장소 타입** | **기술 스택** | **설명** |
| **실시간 수집 영역** | In-Memory Storage | Redis, Kafka | - 금융 거래 및 고객 이벤트 실시간 수집 |
| **정형 데이터 적재 영역** | RDBMS | Oracle, MySQL | - 거래 이력, 고객 정보, 카드 결제 정보 |
| **비정형 데이터 적재 영역** | Distributed Storage | HDFS, S3 | - 로그 파일, 트랜잭션 로그, 이벤트 데이터 |
| **ETL 처리 영역** | Distributed Storage | DataStage, Spark | - 정형 및 비정형 데이터를 목적별 마트로 적재 |

#### ****목적별 마트 구성****

|  |  |  |  |
| --- | --- | --- | --- |
| **마트 명** | **설명** | **적재 대상 데이터** | **기술 스택** |
| **고객 분석 마트** | - 고객 행동 분석, 거래 패턴 분석 | 고객 정보, 금융 거래 정보, 행동 데이터 | HDFS, MySQL |
| **마케팅 분석 마트** | - 캠페인 성과 분석, 고객 반응 추적 | 마케팅 정보, 캠페인 참여 이력 | HDFS, AWS S3 |
| **금융 리스크 분석 마트** | - 리스크 평가 모델, 투자 위험 분석 | 금융 거래 정보, 리스크 평가 데이터 | HDFS, Oracle DB |
| **상품 관리 마트** | - 금융 상품별 거래 현황 분석, 고객별 상품 사용 패턴 분석 | 상품 정보, 거래 내역 | Oracle DB |
| **메타 데이터 마트** | - 스키마 변경 이력, 데이터 변경 추적 | 메타 데이터, 변경 이력 | MySQL, Redis |

### ****메타 데이터 관리 및 조회****

#### ****메타 데이터 관리 구조****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Schema Registry** | - Kafka와 연동하여 데이터 스키마 관리 | Confluent Schema Registry |
| **Data Catalog** | - 데이터의 정의, 소유권, 변경 이력을 관리하며 탐색 가능 | Apache Atlas |
| **Data Lineage** | - 데이터의 생성부터 분석까지의 흐름을 추적 | Apache Atlas, Spark |
| **Data Governance** | - 데이터 표준화 및 보안 규정을 준수하며 관리 | Apache Ranger |

#### ****메타 데이터 조회 방식****

* **API 기반 조회:**
  + Spring Boot로 메타 데이터 API 개발
  + GET 요청 시 메타 데이터 정보를 JSON 형식으로 반환
* **UI 기반 조회:**
  + React 기반의 관리 화면 제공
  + 데이터 스키마 변경 이력, 컬럼 설명, 관계 정보를 시각화
* **분석 플랫폼 연계:**
  + Tableau와 PowerBI에서 메타 데이터를 직접 조회 가능
  + 데이터 변경 시 실시간 반영

### ****분석용 목적 DB 구성****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **구성 요소** | **기술 스택** | **설명** |
| **실시간 분석 DB** | In-Memory DB | Redis, Memcached | - 실시간 거래 처리, 고객 반응 분석 |
| **대용량 분석 DB** | Columnar Storage | HDFS, AWS Redshift | - 대규모 배치 처리, 금융 리스크 분석 |
| **예측 분석 DB** | NoSQL DB | MongoDB, Cassandra | - 비정형 데이터 처리, 행동 분석 |
| **분석 결과 저장 DB** | RDBMS | Oracle, MySQL | - 분석 모델 결과 저장, 통계 데이터 관리 |
| **데이터 레이크** | Distributed Storage | HDFS, AWS S3 | - 원천 데이터, 정제 데이터, 분석 데이터 통합 저장 |

### ****데이터 보관 주기 및 파기 정책****

|  |  |  |  |
| --- | --- | --- | --- |
| **데이터 유형** | **보관 주기** | **파기 조건** | **파기 방식** |
| **고객 정보** | 10년 보관 | 계약 종료 및 고객 요청 시 | 안전 삭제 후 인증서 발급 |
| **금융 거래 정보** | 7년 보관 | 금융 거래 종료 및 법적 보관 기간 만료 시 | 암호화 후 파기 |
| **마케팅 캠페인 정보** | 5년 보관 | 캠페인 종료 후 5년 이후 | 물리적 삭제 |
| **로그 및 이벤트 데이터** | 1년 보관 | 시스템 이벤트 종료 후 1년 이후 | 자동 삭제 |
| **메타 데이터** | 변경 시 3년 보관 | 데이터 모델 변경 후 3년 보관 후 파기 | 데이터 재정비 후 삭제 |

## ****보안 아키텍처 (SA)****

### ****도입 전 안내말****

NH 금융지주의 보안 아키텍처는 **금융 데이터의 안전성, 기밀성, 무결성**을 보장하기 위해 설계되었습니다.  
고객의 민감한 금융 정보를 보호하고, 외부 공격 및 데이터 유출을 방지하기 위해  
**암호화 기술, 인증 체계, 접근 제어, 전송 보안**을 강화하였습니다.  
특히, **전송 구간 암호화, 저장 데이터 암호화, 권한 관리**를 통해 금융 기관의 보안 규정을 준수합니다.

### ****제3자 동의 기반 통합 데이터 인증****

#### ****인증 방식 정의****

|  |  |  |
| --- | --- | --- |
| **인증 유형** | **설명** | **기술 스택** |
| **OAuth2.0** | - 외부 API 및 마이크로서비스 간 인증 관리 - 금융사 및 외부 협력사 간 데이터 연동 | Spring Security, Keycloak |
| **JWT (JSON Web Token)** | - 사용자의 인증 정보와 권한 정보를 포함한 토큰 생성 - 무결성 검증을 통한 신뢰성 확보 | Spring Security, JWT |
| **SSO (Single Sign-On)** | - NH 금융지주 통합 인증을 통한 단일 로그인 - 계열사 간 인증 정보 공유 | SAML, OpenID Connect |
| **Multi-Factor Authentication (MFA)** | - 2단계 인증 (OTP, SMS 인증 등) 추가 적용 | Google Authenticator, Duo |
| **Role-Based Access Control (RBAC)** | - 역할 기반 권한 제어 - 업무별 최소 권한 원칙 적용 | Apache Ranger, Keycloak |

#### ****제3자 동의 프로세스****

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 동의 요청** | - 사용자가 외부 기관(타 금융사)에 데이터 연동 요청 | Spring Boot, OAuth2.0 |
| **2. 인증 및 승인** | - 사용자 인증 후 외부 기관으로 승인 요청 전달 | Keycloak, JWT |
| **3. 데이터 접근 허가** | - Role-Based Access Control을 통해 데이터 접근 허가 | Apache Ranger |
| **4. 데이터 연동 및 처리** | - API Gateway를 통해 외부 기관으로 데이터 전송 | Spring Cloud Gateway, Zuul |
| **5. 로깅 및 감사 처리** | - 모든 접근 기록을 ELK Stack에 저장, 추적 가능 | ELK (Elasticsearch, Logstash, Kibana) |

### ****전송구간 암호화 설계****

#### ****전송 구간 보안 구조****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **TLS 1.3** | - 모든 데이터 전송 구간에 대한 암호화 - 최신 암호화 방식 사용 | OpenSSL, Nginx, HAProxy |
| **HTTPS 적용** | - 모든 API 통신에 HTTPS 적용 - SSL 인증서 사용 | Spring Security, Let's Encrypt |
| **MFT (Managed File Transfer)** | - 금융 기관 간 대용량 파일 전송 시 보안 통신 설정 | EZGator MFT, FOS MFT |
| **IPSec VPN** | - 내부 네트워크와 외부 네트워크 간 전송 구간 암호화 | Cisco VPN, Fortinet VPN |
| **SSH/SFTP** | - 서버 간 데이터 전송 시 SSH Key 기반 보안 적용 | OpenSSH, FileZilla |

#### ****전송구간 암호화 프로세스****

1. **클라이언트 요청 → API Gateway:**
   * 사용자 요청이 API Gateway를 통해 전송
   * Spring Cloud Gateway에서 TLS 1.3 암호화 적용
2. **Gateway → Microservice:**
   * Microservice와의 통신 또한 HTTPS로 암호화
   * 인증 토큰(JWT) 검증 후 데이터 처리
3. **Microservice → Database:**
   * DB와의 통신도 TLS로 암호화하여 무결성 보장
4. **파일 전송 → MFT:**
   * 대용량 데이터 전송 시 MFT 솔루션 사용
   * IPSec VPN을 통한 전송 경로 보호
5. **로그 기록 및 감사:**
   * 모든 통신 기록은 ELK Stack에 저장하여 추적 및 분석

### ****전송파일 암호화 설계****

#### ****일 암호화 전략****

|  |  |  |
| --- | --- | --- |
| **암호화 대상** | **암호화 방식** | **기술 스택** |
| **고객 정보 파일** | AES-256 대칭 키 암호화 | OpenSSL, AWS KMS |
| **금융 거래 파일** | RSA-2048 공개키 암호화 | OpenSSL, GPG |
| **마케팅 캠페인 파일** | AES-128 대칭 키 암호화 | AWS S3 Server-Side Encryption |
| **정기 배치 데이터** | PGP 암호화 적용 | GnuPG, OpenPGP |
| **로그 파일** | AES-256 + Base64 Encoding | OpenSSL, Apache Kafka |

#### ****파일 암호화 처리 프로세스****

1. **파일 생성 시 암호화:**
   * Microservice에서 생성된 파일은 AES-256으로 암호화 후 S3에 업로드
2. **전송 시 암호화:**
   * IPSec VPN 및 SFTP 전송 시 추가 암호화 처리
3. **복호화 시점:**
   * 권한이 검증된 사용자만 복호화 키 접근 가능
   * AWS KMS를 통해 복호화 키를 안전하게 관리
4. **파일 접근 로깅:**
   * 모든 파일 접근 이력은 ELK Stack에 저장

### ****개인정보 처리 시 암호화/마스킹/파기 설계****

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **암호화 처리** | - 고객 민감 정보(AES-256 암호화 적용) | AWS KMS, OpenSSL |
| **마스킹 처리** | - 웹 UI 및 내부 조회 시 개인정보 마스킹 처리 | Spring Security, Maskify |
| **파기 정책** | - 보관 주기 만료 시 안전 삭제 - 영구 삭제 시 Secure Wipe 적용 | Apache Ranger, HDFS |
| **접근 제어 관리** | - Role-Based Access Control로 관리 | Apache Ranger, Keycloak |
| **감사 및 로깅 처리** | - 모든 접근 이력을 실시간 기록 - Kibana에서 분석 및 시각화 | ELK Stack |

### ****계정 및 권한 관리 체계****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **SSO 인증 체계** | - SSO(Single Sign-On)를 통해 통합 로그인 관리 | Keycloak, SAML |
| **Role-Based Access Control** | - 역할 기반의 권한 관리 (Admin, Manager, User 등) | Apache Ranger |
| **Multi-Factor Authentication** | - OTP 인증, 생체 인증 등 다단계 인증 적용 | Google Authenticator, Duo |
| **감사 로깅 및 추적** | - 접근 이력 추적 및 실시간 분석 | ELK Stack, Kibana |

## ****인터페이스 아키텍처 (IA)****

### ****도입 전 안내말****

NH 금융지주의 인터페이스 아키텍처는 **계열사 간 데이터 통합, 외부 금융기관 연계, 실시간 데이터 수집**을  
안전하고 신속하게 처리할 수 있도록 설계되었습니다.  
**FEP (Front End Processor)**, **MCA (Message Communication Adapter)**,  
\*\*MFT (Managed File Transfer)\*\*를 통해 금융 거래의 안정성과 보안을 보장하며,  
모든 인터페이스는 **표준 프로토콜**을 기반으로 통합 관리됩니다.

### ****계열사 데이터 수집 인터페이스 설계****

#### ****계열사 연계 구조****

|  |  |  |  |
| --- | --- | --- | --- |
| **계열사 명** | **연계 방식** | **프로토콜 및 기술 스택** | **주요 데이터 유형** |
| **NH 농협은행** | - 금융 거래 데이터 실시간 전송 - 고객 정보 업데이트 | REST API, FEP, MCA | 거래 내역, 계좌 정보, 고객 정보 |
| **NH 농협카드** | - 카드 결제 정보 실시간 연동 - 승인 및 취소 내역 전달 | REST API, ISO 8583 | 카드 거래 내역, 승인 상태, 포인트 적립 |
| **NH 투자증권** | - 투자 상품 거래 정보 연동 - 시세 조회 및 거래 내역 전송 | FIX Protocol, WebSphere MQ | 증권 거래 내역, 시세 정보, 투자상품 정보 |
| **NH 생명보험** | - 보험 계약 정보 연동 - 보험금 청구 내역 전송 | SOAP, WebSphere MQ | 계약 정보, 청구 내역, 만기 정보 |
| **NH 손해보험** | - 손해보험 정보 연동 - 보험금 지급 및 손해 청구 내역 전달 | SOAP, FEP | 손해보험 계약 정보, 청구 내역 |
| **대외 금융기관** | - 공공기관 연계, 금융감독원 보고 - KYC 인증 및 AML 규제 대응 | SFTP, MFT, WebSphere MQ | 금융 감독 보고서, 고객 인증 정보 |

#### ****데이터 수집 프로세스****

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 데이터 수집 요청** | - FEP 및 MCA를 통해 계열사로부터 데이터 수집 요청 | FEP, MCA, REST API |
| **2. 실시간 수집 처리** | - Kafka를 통해 수집된 데이터를 Streaming 처리 | Kafka, Spring Cloud |
| **3. 배치 수집 처리** | - DataStage를 통한 일괄 수집 및 클렌징 작업 | IBM DataStage, Control-M |
| **4. 임시 보관 및 적재** | - HDFS에 임시 저장 후, ETL 과정을 통해 목적별 마트로 적재 | HDFS, AWS S3, Spark |
| **5. 데이터 마트 배치** | - 고객정보 마트, 거래정보 마트, 마케팅정보 마트로 분류 | Oracle DB, MySQL |

### ****올원뱅크 행동 데이터 수집 설계****

#### ****올원뱅크 연계 구조****

|  |  |  |
| --- | --- | --- |
| **연계 대상** | **설명** | **프로토콜 및 기술 스택** |
| **모바일 뱅킹** | - 올원뱅크 사용자의 행동 데이터 실시간 수집 - 로그인, 조회, 결제 이력 | Kafka, Spring Cloud Stream |
| **인터넷 뱅킹** | - 웹 상에서의 금융 거래 및 이벤트 추적 | REST API, Zuul Gateway |
| **ATM 거래 연동** | - ATM 기기 사용 시 발생하는 거래 및 인출 내역 수집 | WebSocket, REST API |
| **고객 반응 이벤트** | - 마케팅 메시지 수신 시 반응 데이터를 실시간 수집 | Kafka, Redis Cache |

#### ****데이터 수집 및 적재 프로세스****

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 이벤트 수집** | - 올원뱅크 사용자의 행동이 발생할 때 Kafka Topic에 전송 | Kafka, Spring Boot |
| **2. 실시간 스트리밍** | - Spring Cloud Stream을 통해 데이터 스트림 처리 | Spring Cloud Stream |
| **3. 데이터 클렌징** | - 불필요한 정보 제거 및 데이터 정규화 | Apache Spark, DataStage |
| **4. 저장소 적재** | - HDFS에 일단 적재 후, 목적별 마트로 분류 | HDFS, AWS S3 |
| **5. 분석 및 연계** | - Spark SQL을 통해 실시간 분석 후, BI 시스템에 전달 | Spark, Tableau, PowerBI |

### ****데이터 연계 표준 및 프로토콜 설계****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **프로토콜** | **설명** | **기술 스택** |
| **실시간 전송** | REST API, WebSocket | - 실시간 금융 거래 및 고객 정보 전송 - 즉시 처리 보장 | Spring Boot, Zuul |
| **대량 전송** | MFT, SFTP | - 금융 데이터, 고객 정보, 보험 거래 내역 등 대용량 데이터 전송 | EZGator MFT, FOS MFT |
| **메시지 전송** | Kafka, MQ | - 비동기 메시지 전달을 통한 이벤트 처리 | Kafka, WebSphere MQ |
| **보안 전송** | HTTPS, TLS 1.3 | - 모든 데이터 전송 시 암호화 적용 | OpenSSL, HAProxy |
| **인터페이스 표준** | ISO 8583, FIX Protocol | - 카드 결제 및 투자 거래 정보의 표준화된 데이터 전송 | ISO 8583, FIX |

### ****분석 플랫폼과의 연계 설계****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **BI 포털 연동** | - HDFS에 저장된 데이터를 Tableau 및 PowerBI로 시각화 | Tableau, PowerBI |
| **실시간 데이터 분석** | - Spark Streaming을 통해 실시간 데이터 분석 - 고객 행동 및 거래 추적 | Apache Spark, HDFS |
| **대시보드 연계** | - 실시간 거래 상태 및 금융 상품 현황을 시각화 | Grafana, Kibana |
| **모니터링 시스템 연동** | - Prometheus와 ELK Stack을 통해 실시간 모니터링 | Prometheus, ELK Stack |
| **예측 분석 연계** | - Spark MLlib을 활용한 금융 리스크 분석 및 예측 | Spark MLlib, HDFS |

## ****기대효과 및 전략적 가치****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼 구축 프로젝트는 **디지털 금융 혁신**을 목표로,  
**실시간 분석, 개인화된 금융 서비스, 마케팅 최적화**를 이루기 위한 아키텍처입니다.  
본 프로젝트의 기대효과는 금융 거래의 처리 속도를 높이고, 고객 중심의 금융 서비스를 강화하며,  
리스크 관리 및 마케팅 성과를 극대화하는 데 있습니다.

## ****기대효과 분석****

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **실시간 거래 처리** | - Kafka Streaming을 통해 1초 이내의 금융 거래 처리 - 실시간 승인 및 전송 | Spring Boot, Kafka, Spring Cloud |
| **개인화 마케팅 최적화** | - 고객 행동 분석을 통한 타겟 마케팅 - Spark Streaming을 통해 실시간 이벤트 발송 | Spark, Spring Boot, Redis |
| **고객 서비스 개선** | - 고객 요청에 대한 빠른 응답 처리 - 모바일 및 웹 채널에서의 원활한 서비스 제공 | Spring Boot, Zuul Gateway |
| **리스크 관리 최적화** | - 거래 내역 분석을 통한 실시간 리스크 평가 - Spark ML을 활용한 이상 거래 탐지 | Spark MLlib, HDFS |
| **데이터 분석 역량 강화** | - HDFS에 축적된 빅데이터 분석 - Tableau, PowerBI를 통한 시각화 | HDFS, Tableau, PowerBI |
| **운영 비용 절감** | - 클라우드 네이티브 인프라를 통한 하드웨어 비용 절감 - Kubernetes 오토스케일링 | Kubernetes, OpenShift |
| **확장성 확보** | - Microservices 기반으로 서비스 간 독립적 확장 가능 | Spring Boot, Docker, Kubernetes |

## ****전략적 가치 분석****

|  |  |  |
| --- | --- | --- |
| **구분** | **전략적 가치** | **세부 설명** |
| **디지털 금융 전환** | - 실시간 금융 서비스 제공 - 모바일, 웹, ATM 연계 통합 | 모든 금융 채널을 통합하여 고객이 어느 곳에서도 동일한 금융 서비스 경험 |
| **데이터 중심 의사결정** | - 고객 데이터 분석을 통한 맞춤형 금융 상품 제안 - 리스크 관리 최적화 | Spark 기반의 실시간 분석을 통해 금융 위험을 사전에 탐지 |
| **개인화 마케팅 실행** | - 고객 행동 분석을 기반으로 실시간 이벤트 발송 - 개인화된 금융 상품 추천 | 고객의 거래 패턴 분석을 통해 최적의 금융 상품을 제공 |
| **전사 통합 데이터 관리** | - Data Lake를 통한 전사 데이터 통합 - 통합된 분석 환경 제공 | HDFS와 AWS S3에 데이터를 통합하여 분석 및 시각화를 용이하게 함 |
| **확장성 및 유연성 확보** | - Microservice 기반의 독립적 서비스 확장 - 무중단 배포 지원 | Kubernetes 클러스터를 통해 Auto Scaling 및 Blue-Green 배포 적용 |
| **보안 및 규제 준수** | - 금융 데이터의 안전한 전송 및 저장 - 금융 규제 및 데이터 보호 규정 준수 | TLS 1.3 전송 암호화 및 AES-256 저장 암호화 적용 |

## ****설계 및 구현 방안****

### ****디지털 금융 전환을 위한 설계 방안****

1. **모바일 뱅킹, 인터넷 뱅킹 통합:**
   * Spring Cloud Gateway를 통해 모든 금융 채널을 통합
   * API Gateway가 모든 요청을 받아 인증 및 라우팅 처리
2. **API First Approach:**
   * 모든 금융 서비스는 REST API로 설계되어 빠른 연동 가능
   * ISO 20022 표준 메시지 형식을 적용하여 국제 금융 기관과의 호환성 강화
3. **실시간 트랜잭션 처리:**
   * Kafka Streaming을 통해 실시간 거래 내역 처리
   * Kafka Consumer가 거래 정보를 실시간으로 분석

### ****개인화 마케팅 실행을 위한 설계 방안****

1. **실시간 고객 분석:**
   * Spark Streaming을 통해 고객 행동을 실시간 분석
   * 고객의 클릭, 결제, 로그인 이력을 분석하여 즉각적인 캠페인 발송
2. **타겟 마케팅 최적화:**
   * Redis Cache에 실시간 세션 정보를 저장하여,  
     마케팅 대상자 선정 시 지연 없이 처리
   * Spring Cloud Stream을 통해 다중 채널로 메시지 전송
3. **Omni-Channel 전략:**
   * 모바일, 웹, ATM, 콜센터 등 모든 채널에서 동일한 마케팅 정보 제공
   * Spring Cloud Gateway로 트래픽을 제어하며,  
     메시지 전달의 일관성 유지

### ****리스크 관리 최적화 설계 방안****

1. **Spark ML 기반 리스크 모델링:**
   * 머신러닝 모델을 통해 거래 패턴 분석
   * 이상 거래 탐지 시 즉각적인 알림 및 프로세스 차단
2. **실시간 금융 리스크 분석:**
   * Apache Spark와 HDFS를 이용해 실시간으로 거래 정보를 분석
   * 특정 위험 신호가 감지되면 관리자에게 알림
3. **금융 규제 준수:**
   * KYC(고객 알기) 및 AML(자금 세탁 방지) 절차를 자동화
   * Spring Security와 Keycloak을 통해 접근 제어 강화

### ****데이터 분석 및 통합 관리 설계 방안****

1. **Data Lake 기반의 통합 관리:**
   * 모든 금융 데이터를 HDFS와 S3에 통합하여 관리
   * 실시간 분석을 위해 Spark와 연결
2. **ETL 처리 최적화:**
   * DataStage를 활용하여 실시간 ETL 수행
   * 클렌징된 데이터를 목적별 마트에 자동 적재
3. **BI 포털 연계:**
   * Tableau와 PowerBI를 통해 분석 결과를 실시간 시각화
   * Spark SQL을 통해 대시보드 업데이트

### ****보안 및 규제 대응 방안****

1. **전송 구간 암호화:**
   * TLS 1.3을 적용하여 모든 금융 트랜잭션 암호화
   * IPSec VPN을 통한 내부망 전송 보안
2. **저장 데이터 암호화:**
   * AES-256 방식으로 금융 데이터 암호화
   * AWS KMS(Key Management Service)를 통한 키 관리
3. **접근 통제 및 감사 로깅:**
   * Keycloak을 통한 Role-Based Access Control (RBAC)
   * ELK Stack을 통한 모든 접근 기록 저장 및 분석

# ****아키텍처 주요 정보****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼 아키텍처는 **전사 데이터를 일원화하고 실시간 처리와 분석이 가능한 구조**로 설계되었습니다.  
본 아키텍처는 **Private Cloud 기반의 Kubernetes 클러스터**, **Data Lake 통합 관리**,  
**Microservices 기반의 금융 서비스 처리**를 통해 고객 중심의 금융 서비스 제공과 디지털 금융 전환을 가속화합니다.  
또한, **API Gateway**, **Kafka Streaming**, **Spark Analytics**를 통해 고성능의 데이터 처리와  
무중단 서비스를 보장합니다.

## ****도입 시스템 정의****

### ****신규 도입 시스템 구성****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **시스템 명** | **기술 스택** | **주요 기능** |
| **Application** | API Gateway | Spring Cloud Gateway | - 금융 서비스에 대한 인증 및 라우팅 관리 |
|  | Campaign Service | Spring Boot, Redis | - 실시간 마케팅 캠페인 생성 및 실행 |
|  | Customer Service | Spring Boot, Oracle | - 고객 정보 조회 및 관리 |
|  | Transaction Service | Spring Boot, MySQL | - 금융 거래 처리 및 트랜잭션 관리 |
| **Data Storage** | Data Lake | HDFS, AWS S3 | - 전사 데이터 통합 관리 및 분석용 데이터 보관 |
|  | Data Mart | Oracle, MySQL | - 목적별 데이터 마트 구성 |
| **Integration** | Messaging Queue | Kafka, RabbitMQ | - 비동기 메시지 전송 및 이벤트 처리 |
|  | ETL Pipeline | DataStage, Spark | - 배치 처리 및 실시간 데이터 수집 |
| **Security** | Identity Management | Keycloak, OAuth2.0 | - 통합 인증 및 권한 관리 |
|  | Encryption Service | OpenSSL, AWS KMS | - 데이터 암호화 및 복호화 관리 |
| **Monitoring** | Monitoring System | Prometheus, Grafana | - 시스템 자원 모니터링 및 장애 탐지 |
|  | Logging System | ELK Stack | - 모든 서비스 로그 수집 및 분석 |

### ****기존 시스템 활용****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **시스템 명** | **기술 스택** | **주요 기능** |
| **Core Banking** | NH 농협은행 코어 뱅킹 | Mainframe, Oracle DB | - 계좌 관리, 거래 처리, 대출 관리 |
| **카드 시스템** | NH 농협카드 결제 시스템 | WebSphere, Oracle DB | - 카드 승인 및 결제 처리 |
| **보험 시스템** | NH 생명보험, NH 손해보험 | WebSphere, Oracle DB | - 보험 계약 관리 및 청구 처리 |
| **투자 시스템** | NH 투자증권 거래 시스템 | FIX Protocol, WebSphere MQ | - 주식 거래 및 투자 상품 관리 |
| **CRM 시스템** | 통합 고객 관리 시스템 | Siebel CRM, MySQL | - 고객 정보 관리 및 CRM 활동 |
| **전자 채널 시스템** | 인터넷뱅킹, 모바일뱅킹 | Spring Boot, React | - 온라인 금융 서비스 제공 |

## ****인터페이스 구성도****

### ****인터페이스 연계 구조****

[Channel Layer]

- Mobile Banking

- Internet Banking

- ATM Service

- API Gateway

|

[Integration Layer]

- FEP (Front End Processor)

- MCA (Message Communication Adapter)

- MFT (Managed File Transfer)

|

[Business Layer]

- Core Banking System

- Card System

- Insurance System

- Investment System

|

[Data Layer]

- Oracle DB

- MySQL DB

- Data Lake (HDFS, AWS S3)

|

[Analytics & BI Layer]

- Spark Streaming

- Tableau

- PowerBI

## ****데이터 주제 영역 정의****

|  |  |  |  |
| --- | --- | --- | --- |
| **주제 영역** | **설명** | **적재 위치** | **주요 기술 스택** |
| **고객 정보** | - 고객의 기본 정보 및 거래 이력 관리 - 신용 등급 및 마케팅 수신 동의 정보 | Oracle DB, MySQL | Spring Boot, Hibernate |
| **상품 정보** | - 금융 상품에 대한 기본 정보 및 조건 관리 | Oracle DB, MySQL | Spring Boot, MyBatis |
| **거래 정보** | - 예금, 대출, 카드 결제, 투자 내역 등 금융 거래 정보 관리 | Oracle DB, HDFS | Spring Boot, DataStage |
| **채널 정보** | - 인터넷 뱅킹, 모바일 뱅킹, 영업점 거래 이력 | Oracle DB, HDFS | Spring Cloud Gateway |
| **마케팅 정보** | - 캠페인 참여 이력 및 반응 정보 관리 | Redis, HDFS | Spring Boot, Kafka |
| **분석 정보** | - 금융 리스크, 마케팅 성과 분석 결과 | Spark, Tableau, PowerBI | Apache Spark, Tableau |
| **메타 데이터** | - 각 데이터의 스키마 및 변경 이력 관리 | Schema Registry | Confluent Schema Registry |

## ****도입 솔루션 목록****

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **솔루션 명** | **기술 스택** | **주요 기능** |
| **Web Server** | Nginx, HAProxy | Nginx, HAProxy | - 트래픽 로드 밸런싱 및 보안 강화 |
| **Application Server** | Spring Boot, WebSphere | Spring Framework | - 금융 트랜잭션 처리 및 서비스 로직 관리 |
| **Database** | Oracle, MySQL, HDFS | Oracle DB, MySQL, HDFS | - 금융 데이터 저장 및 분석 |
| **Messaging** | Kafka, RabbitMQ | Apache Kafka, RabbitMQ | - 비동기 메시지 처리 및 실시간 스트림 처리 |
| **ETL Pipeline** | DataStage, Spark | IBM DataStage, Spark | - 대량 데이터 클렌징 및 적재 |
| **Monitoring** | Prometheus, Grafana | Prometheus, Grafana | - 실시간 모니터링 및 알림 관리 |
| **Security** | Keycloak, OAuth2.0 | Keycloak, OAuth2.0 | - 인증 및 권한 관리 |
| **Logging** | ELK Stack | Elasticsearch, Logstash, Kibana | - 서비스 로그 수집 및 분석 |

# ****배포 및 운영 전략****

## ****9.1 도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **안정적인 금융 거래 처리와 실시간 데이터 분석**을 보장하기 위해  
**Kubernetes 기반의 클라우드 네이티브 아키텍처**로 설계되었습니다.  
본 전략에서는 **CI/CD 파이프라인 구축, 무중단 배포, 실시간 모니터링, 장애 복구**를 통해  
안정적이고 지속 가능한 금융 서비스를 운영할 수 있는 체계를 제공합니다.

## ****CI/CD 파이프라인 설계****

### ****CI/CD 구성 요소****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Source Control** | - 코드 버전 관리 및 협업 처리 | Git, GitLab |
| **Build Server** | - 코드 빌드 및 패키징 | Jenkins, GitLab CI |
| **Artifact Repository** | - 빌드 결과물 저장 및 관리 | Nexus, JFrog Artifactory |
| **Container Registry** | - Docker 이미지 관리 및 배포 | Docker Hub, ECR |
| **Deployment** | - Kubernetes 클러스터에 자동 배포 | ArgoCD, Helm |
| **Monitoring & Logging** | - 배포 상태와 로그 수집 | Prometheus, Grafana, ELK |

### ****CI/CD 파이프라인 단계****

1️⃣ **소스 관리 (Source Control):**

* GitLab을 통해 코드 버전 관리
* Branch 전략: main, develop, feature/\*

2️⃣ **빌드 (Build):**

* Jenkins Pipeline을 통해 Gradle 또는 Maven 빌드
* Unit Test 및 Code Quality Test (SonarQube 연동)

3️⃣ **패키징 및 저장 (Packaging & Store):**

* Docker 이미지로 패키징하여 Docker Hub 또는 ECR에 저장
* Nexus Repository에 JAR 및 WAR 파일 아카이빙

4️⃣ **배포 (Deploy):**

* ArgoCD를 통해 Kubernetes 클러스터에 무중단 배포
* Helm Chart를 활용한 버전 관리

5️⃣ **모니터링 및 로깅 (Monitoring & Logging):**

* Prometheus와 Grafana로 실시간 모니터링
* ELK Stack (Elasticsearch, Logstash, Kibana)으로 모든 로그 수집

## ****블루-그린 배포 및 롤링 업데이트****

### ****배포 전략 정의****

|  |  |  |
| --- | --- | --- |
| **배포 전략** | **설명** | **기술 스택** |
| **Blue-Green Deploy** | - 새로운 버전을 Blue 환경에 배포 후, 테스트 완료 시 Green 환경으로 전환 | ArgoCD, Kubernetes |
| **Canary Deploy** | - 일부 트래픽만 새로운 버전으로 전송하여 안정성을 확인한 후 점진적 확장 | Istio, Argo Rollouts |
| **Rolling Update** | - 기존 Pod를 순차적으로 대체하면서 무중단 배포 진행 | Kubernetes, Helm |
| **Hot Swap Deploy** | - 즉시 장애 복구가 가능하도록 예비 인스턴스를 미리 준비 | Kubernetes, Nginx |

### ****블루-그린 배포 프로세스****

[Current Environment: Green]

|

+---> 사용자 트래픽 처리

|

v

[New Version: Blue]

|

+---> 테스트 환경에서 모든 검증 완료

|

v

[Switch Traffic]

|

+---> Green → Blue 트래픽 전환

|

v

[Rollback 가능]

|

+---> 만약 오류 발생 시 Green으로 즉시 전환

## ****장애 복구 및 DR(Disaster Recovery) 전략****

### ****DR 구성 요소 정의****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Replication** | - DB 이중화 구성 (Active-Standby) | Oracle RAC, MySQL Replication |
| **Backup Storage** | - 일일 백업 및 스냅샷 생성 | AWS S3, HDFS |
| **Failover Strategy** | - Kubernetes 클러스터의 자동 장애 조치 설정 | Kubernetes, ArgoCD |
| **Cross-Region DR** | - AWS Multi-Region DR 전략 | AWS S3, Route 53 |
| **Message Queue 복구** | - Kafka 스트림 복구 및 로그 재처리 | Kafka, HDFS |

### ****DR 복구 프로세스****

1️⃣ **이중화된 DB 복구:**

* Oracle RAC와 MySQL Replication을 통해 장애 시 Standby로 즉시 전환

2️⃣ **스냅샷 및 백업 복구:**

* AWS S3에 저장된 스냅샷을 통해 15분 이내 복구 목표 설정

3️⃣ **Kubernetes Failover:**

* ArgoCD와 Kubernetes Auto Healing을 통해 Pod 복구

4️⃣ **Kafka 스트림 재처리:**

* 메시지 큐에 남아있는 로그를 HDFS에서 복원

## ****실시간 모니터링 및 로깅 관리****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Prometheus** | - 실시간 리소스 사용량, TPS, Latency 모니터링 | Prometheus, Grafana |
| **Grafana** | - 실시간 대시보드 제공 - 클러스터 상태 및 장애 탐지 시각화 | Grafana |
| **ELK Stack** | - Elasticsearch, Logstash, Kibana를 활용한 실시간 로그 분석 | ELK Stack |
| **Alert Manager** | - 장애 발생 시 Slack, 이메일, SMS 알림 설정 | Prometheus Alert Manager |
| **Sentry** | - 애플리케이션 레벨의 오류 추적 및 예외 분석 | Sentry |

## ****장애 대응 및 복구 시나리오****

|  |  |  |  |
| --- | --- | --- | --- |
| **장애 유형** | **발생 원인** | **대응 방안** | **복구 시간 목표 (RTO)** |
| **Database 장애** | - Master DB 장애 - 데이터 복제 지연 | Oracle RAC로 Standby 전환 MySQL은 Read Replica 활성화 | 10분 이내 |
| **Microservice 장애** | - Pod Crash - OutOfMemory Exception | Kubernetes Auto Healing ArgoCD 롤백 처리 | 5분 이내 |
| **네트워크 장애** | - API Gateway 다운 - DDoS 공격 발생 | Nginx 및 HAProxy 이중화 WAF 설정 강화 | 5분 이내 |
| **ETL Job 실패** | - DataStage Job 중단 - HDFS 접근 불가 | Control-M에서 Job 재실행 DataStage 복구 스크립트 실행 | 10분 이내 |
| **메시지 큐 장애** | - Kafka Broker 다운 - 메시지 처리 중단 | Kafka MirrorMaker로 복구 HDFS에서 메시지 재처리 | 3분 이내 |

# ****로그 및 모니터링 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **실시간 거래 처리, 금융 데이터 분석, 마케팅 캠페인 실행**을  
안정적으로 운영하기 위해 **모니터링 및 로그 관리 체계**를 도입하였습니다.  
본 전략은 **시스템 리소스 사용 현황, 애플리케이션 로그, 트랜잭션 처리 상태**를 실시간으로 모니터링하고,  
장애 발생 시 신속하게 탐지하여 복구할 수 있도록 설계되었습니다.

## ****로그 수집 및 분석 설계****

### ****로그 수집 구조****

|  |  |  |  |
| --- | --- | --- | --- |
| **로그 유형** | **설명** | **기술 스택** | **저장 위치** |
| **Application Log** | - Spring Boot 애플리케이션의 실행 로그 및 예외 처리 | Spring Boot, Logback | ELK Stack (Elasticsearch) |
| **Transaction Log** | - 금융 거래 및 캠페인 실행 시 발생하는 트랜잭션 로그 | Kafka, Spring Boot | HDFS, AWS S3 |
| **Security Log** | - OAuth2.0, JWT 인증 로그 - 권한 관리 및 접근 제어 기록 | Keycloak, Spring Security | Elasticsearch |
| **Database Log** | - Oracle, MySQL에서 발생하는 SQL 로그 및 쿼리 실행 시간 분석 | Oracle, MySQL | S3 Snapshot, LogStash |
| **Network Log** | - API Gateway 및 Nginx의 요청 및 응답 로그 | Nginx, Spring Cloud Gateway | Fluentd, Elasticsearch |
| **ETL Job Log** | - DataStage Job 실행 결과 및 에러 로그 | DataStage, Spark | HDFS, S3 Backup |
| **Infrastructure Log** | - Kubernetes, Docker 클러스터 상태 모니터링 | Prometheus, Grafana | Prometheus TSDB |

### ****로그 수집 및 분석 프로세스****

1️⃣ **로그 생성:**

* 각 마이크로서비스(Spring Boot), API Gateway, Kafka, DataStage 등에서 로그 생성

2️⃣ **로그 전송:**

* Fluentd와 Logstash를 통해 실시간 로그 수집
* Kafka에 전송 후, ELK Stack으로 전송

3️⃣ **로그 저장:**

* 모든 로그는 Elasticsearch에 인덱싱되어 실시간 검색 가능
* 장기 보관이 필요한 데이터는 AWS S3에 주기적으로 백업

4️⃣ **로그 분석 및 시각화:**

* Kibana와 Grafana를 통해 실시간 시각화
* 이상 탐지 시 알림 전송 (Slack, Email)

5️⃣ **알림 및 대응:**

* 장애 탐지 시 Prometheus AlertManager가 Slack, 이메일 알림 전송

## ****모니터링 아키텍처 설계****

### ****모니터링 구성 요소 정의****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Infrastructure Monitoring** | - 서버, 네트워크, Kubernetes 클러스터 상태 모니터링 | Prometheus, Grafana |
| **Application Monitoring** | - Spring Boot 애플리케이션의 메트릭 수집 및 분석 | Prometheus, Micrometer |
| **API Monitoring** | - API Gateway의 요청 처리 상태 및 응답 시간 모니터링 | Grafana, Prometheus |
| **Database Monitoring** | - Oracle, MySQL, Redis의 성능 및 트랜잭션 상태 모니터링 | Prometheus, Telegraf |
| **Messaging Queue Monitoring** | - Kafka, RabbitMQ의 메시지 처리량 및 에러 탐지 | Prometheus, Grafana |
| **ETL Job Monitoring** | - DataStage와 Spark의 ETL 작업 진행 상태 확인 | Control-M, Prometheus |

### ****실시간 모니터링 대시보드****

1️⃣ **Infrastructure Dashboard**

* CPU, Memory, Disk 사용량 시각화
* Kubernetes Pod 상태 및 노드 리소스 확인

2️⃣ **Application Dashboard**

* 마이크로서비스의 TPS, Latency, 오류 비율 시각화
* API Gateway의 요청 성공/실패 건수 표시

3️⃣ **Database Dashboard**

* Oracle 및 MySQL의 트랜잭션 처리 시간 및 Lock 발생 상태
* Replication Delay 및 Backup 상태 모니터링

4️⃣ **Kafka Monitoring Dashboard**

* Topic 생성 상태, Consumer Lag 확인
* 메시지 전송량과 처리 속도 분석

5️⃣ **ETL Monitoring Dashboard**

* DataStage Job 실행 현황, Spark Streaming 처리 속도
* 실패 시 알림 전송

## ****데이터 보존 및 관리 정책****

|  |  |  |  |
| --- | --- | --- | --- |
| **데이터 유형** | **보관 주기** | **파기 조건** | **파기 방식** |
| **Application Log** | 1년 | 보존 기한 만료 시 | Secure Delete (Shred) |
| **Transaction Log** | 7년 | 금융 거래 종료 시 | AES-256 암호화 후 파기 |
| **Security Log** | 5년 | 법적 규정에 따라 5년 보관 후 파기 | OpenSSL 기반 암호화 삭제 |
| **Database Log** | 3년 | DB 스냅샷 보관 후 삭제 | KMS를 통한 키 관리 및 삭제 |
| **Network Log** | 1년 | 분석 완료 시 정리 | Nginx Log Rotate |
| **ETL Job Log** | 3년 | Job 완료 후 백업 및 삭제 | AWS S3에 백업 후 삭제 |
| **Infrastructure Log** | 1년 | 클러스터 상태 기록 완료 후 정리 | Fluentd Log Rotation |

### ****로그 보관 정책****

* **단기 보관:** ELK Stack에 6개월 보관
* **장기 보관:** AWS S3에 5년간 백업
* **암호화:** AES-256으로 암호화 후 보관
* **접근 제어:** Role-Based Access Control (RBAC)으로 접근 통제

### ****로그 파기 정책****

* 보관 주기 만료 시 **Secure Delete**를 통해 완전 삭제
* 금융 거래와 관련된 데이터는 **암호화 후 물리적 삭제**
* AWS S3에 보관된 로그는 **S3 Object Lock**을 통해 일정 기간 보호

# ****API 관리 및 거버넌스 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **Microservices Architecture** 기반으로 설계되었으며,  
각 금융 서비스는 API를 통해 상호 연동하고 통합 관리됩니다.  
API 관리는 **표준화된 설계, 버전 관리, 보안 접근 제어**를 통해 신뢰성과 확장성을 보장하며,  
**Spring Cloud Gateway**, **Zuul**, **OAuth2.0**을 활용하여 안전한 데이터 전송을 수행합니다.

## ****API 설계 표준화****

### ****API 설계 원칙****

|  |  |  |
| --- | --- | --- |
| **원칙** | **설명** | **적용 기술** |
| **RESTful 설계** | - 모든 API는 RESTful 규약을 준수 - HTTP Method 기반 처리 | Spring Boot, Spring MVC |
| **JSON 기반 통신** | - 모든 요청 및 응답은 JSON 포맷으로 통일 | Jackson, GSON |
| **Stateless Design** | - API 호출 간 상태를 서버에 저장하지 않음 | Spring Boot, Redis |
| **URI Naming Rule** | - 명확하고 일관성 있는 URI 구조 설계 | REST API 규약 |
| **Pagination 적용** | - 대량 데이터 응답 시 페이지네이션 처리 | Spring Data JPA |
| **Rate Limiting** | - API 요청 제한 설정 (DDoS 대응) | Zuul, API Gateway |

### ****URI 설계 구조****

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **URI** | **설명** |
| **GET** | /api/v1/customers | 고객 목록 조회 |
| **GET** | /api/v1/customers/{id} | 고객 상세 정보 조회 |
| **POST** | /api/v1/customers | 고객 신규 등록 |
| **PUT** | /api/v1/customers/{id} | 고객 정보 수정 |
| **DELETE** | /api/v1/customers/{id} | 고객 정보 삭제 |
| **GET** | /api/v1/transactions | 거래 내역 조회 |
| **POST** | /api/v1/transactions | 거래 처리 |
| **GET** | /api/v1/marketing/campaigns | 마케팅 캠페인 목록 조회 |
| **POST** | /api/v1/marketing/campaigns | 마케팅 캠페인 생성 |
| **GET** | /api/v1/security/logs | 보안 로그 조회 |

## ****API 버전 관리 및 확장 전략****

### ****API 버전 관리 전략****

|  |  |  |
| --- | --- | --- |
| **버전 관리 방식** | **설명** | **예시** |
| **URI Versioning** | - URI에 버전을 명시하여 버전 간 호환성 유지 | /api/v1/customers |
| **Header Versioning** | - HTTP Header에 버전 정보를 포함하여 요청 처리 | Accept: application/vnd.nh.v1+json |
| **Parameter Versioning** | - Query Parameter에 버전 정보 전달 | /api/customers?version=1 |
| **Subdomain Versioning** | - 서브도메인에 버전 정보를 포함하여 요청 처리 | v1.api.nhbank.com/customers |

### ****API 확장 전략****

|  |  |  |
| --- | --- | --- |
| **구분** | **확장 전략** | **적용 기술** |
| **Backward Compatibility** | - 기존 API와의 호환성 보장 | Spring Boot, REST API |
| **Deprecation Notice** | - API 폐기 시 사전 공지 및 Grace Period 설정 | Spring Cloud Gateway |
| **Feature Toggle** | - 신규 기능 배포 시 플래그를 통해 단계적 전환 | Spring Boot Profiles |
| **Circuit Breaker** | - 장애 발생 시 API 호출을 중단하고 대체 처리 | Resilience4J, Hystrix |
| **Load Balancing** | - API 요청 부하를 자동 분산하여 처리 | Zuul, Nginx |

## ****API 보안 및 접근 제어****

### ****API 보안 정책****

|  |  |  |
| --- | --- | --- |
| **보안 항목** | **설명** | **적용 기술** |
| **인증(Authentication)** | - OAuth2.0를 활용한 인증 처리 - JWT 토큰 기반 접근 제어 | Spring Security, Keycloak |
| **인가(Authorization)** | - Role-Based Access Control (RBAC)로 권한 관리 | Apache Ranger |
| **전송구간 암호화** | - TLS 1.3을 통한 전송 구간 암호화 | OpenSSL, Nginx |
| **IP Restriction** | - 특정 IP 대역에서만 접근 허용 | Spring Cloud Gateway |
| **CORS 설정** | - Cross-Origin Resource Sharing을 통한 보안 정책 관리 | Spring MVC |

### ****API 보안 프로세스****

1️⃣ **인증 처리:**

* 모든 API 요청 시, OAuth2.0을 통해 인증
* 인증된 사용자는 JWT 토큰을 발급받아 API 호출 시 사용

2️⃣ **인가 처리:**

* JWT에 포함된 권한 정보를 기반으로 API 접근 제어
* Role-Based Access Control을 통해 관리자/사용자 권한 구분

3️⃣ **전송구간 암호화:**

* API Gateway와 모든 Microservice 간 통신 시 HTTPS 적용
* TLS 1.3으로 전송 구간을 암호화하여 데이터 유출 방지

4️⃣ **IP Restriction 설정:**

* 관리 페이지 접근은 사내 IP 또는 VPN을 통해서만 허용
* 외부 접속 시 OAuth2.0을 통해 2단계 인증

5️⃣ **CORS 정책 설정:**

* 신뢰할 수 있는 도메인만 API 호출 허용
* 불법 접근 시 API Gateway에서 차단

# ****DevOps 및 CI/CD 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 \*\*지속적 통합(Continuous Integration)\*\*과 \*\*지속적 배포(Continuous Deployment)\*\*를  
핵심으로 하는 **DevOps 전략**을 채택하였습니다.  
본 전략은 **자동화된 빌드, 테스트, 배포, 모니터링**을 통해 서비스 중단 없이 안정적인 금융 서비스를 제공하며,  
클라우드 네이티브 환경에서 유연하고 확장성 있는 배포를 가능하게 합니다.

## ****DevOps 전략 개요****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Source Control** | - Git 기반 코드 버전 관리 - Branch 전략으로 main, develop, feature 관리 | Git, GitLab |
| **Build Automation** | - Jenkins Pipeline을 통한 자동화 빌드 - Gradle/Maven을 사용한 패키징 | Jenkins, Gradle, Maven |
| **Containerization** | - Docker를 활용한 Microservice 패키징 및 배포 | Docker, Docker Compose |
| **Orchestration** | - Kubernetes 클러스터를 통해 자동 확장 및 롤링 업데이트 | Kubernetes, Helm |
| **Configuration Management** | - Ansible을 통한 환경 설정 및 배포 관리 | Ansible, Terraform |
| **Monitoring & Logging** | - Prometheus와 Grafana를 통한 실시간 모니터링 - ELK Stack으로 로그 분석 | Prometheus, ELK Stack |
| **Deployment Strategy** | - ArgoCD를 통한 GitOps 방식 배포 관리 | ArgoCD, Helm |

## ****CI/CD 파이프라인 설계****

### ****CI/CD 구성 요소****

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 코드 관리** | - GitLab에 코드 버전 관리 - 브랜치 전략: main, develop, feature/\* | Git, GitLab |
| **2. 빌드 및 테스트** | - Jenkins Pipeline에서 자동 빌드 및 테스트 실행 - SonarQube로 코드 품질 분석 | Jenkins, SonarQube |
| **3. 아티팩트 저장소** | - Nexus Repository에 빌드 결과물 저장 - Docker Hub에 Docker 이미지 등록 | Nexus, Docker Hub |
| **4. 컨테이너 생성** | - Dockerfile 기반으로 애플리케이션 컨테이너 생성 | Docker, Docker Compose |
| **5. 배포 자동화** | - ArgoCD를 통해 Kubernetes 클러스터에 자동 배포 | ArgoCD, Helm |
| **6. 모니터링 및 로깅** | - Prometheus와 Grafana로 실시간 모니터링 - ELK Stack으로 로그 수집 | Prometheus, ELK Stack |

### ****CI/CD 파이프라인 흐름도****

[Git Push]

|

+---> [Jenkins Build & Test]

|

+---> [SonarQube Code Analysis]

|

+---> [Docker Build]

|

+---> [Nexus Artifact Upload]

|

+---> [ArgoCD Deployment]

|

+---> [Kubernetes Cluster]

|

+---> [Monitoring & Logging]

## ****자동화 배포 및 모니터링****

### ****자동화 배포 전략****

|  |  |  |
| --- | --- | --- |
| **배포 전략** | **설명** | **기술 스택** |
| **Blue-Green Deployment** | - Blue와 Green 환경을 나누어 새로운 버전 테스트 후 전환 | ArgoCD, Kubernetes |
| **Canary Deployment** | - 일부 트래픽만 신규 버전으로 전달하여 오류 여부를 모니터링 | Istio, Argo Rollouts |
| **Rolling Update** | - 기존 Pod를 하나씩 업데이트하면서 무중단 배포 실행 | Kubernetes, Helm |
| **Hot Swap Deployment** | - 장애 발생 시 이전 버전으로 즉시 롤백 가능한 상태 유지 | Kubernetes, ArgoCD |

### ****실시간 모니터링 구성****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Application Monitoring** | - Spring Boot 애플리케이션의 메트릭 수집 - 메모리, CPU, 트랜잭션 모니터링 | Prometheus, Grafana |
| **Database Monitoring** | - Oracle, MySQL의 쿼리 처리 시간, 트랜잭션 처리량 분석 | Prometheus, Telegraf |
| **API Gateway Monitoring** | - Spring Cloud Gateway의 요청 수, 응답 시간 모니터링 | Grafana, Kibana |
| **Container Monitoring** | - Docker 컨테이너 상태 및 리소스 사용량 추적 | Prometheus, cAdvisor |
| **Cluster Monitoring** | - Kubernetes 노드 상태, Pod 상태 체크 | Prometheus, Grafana |

## ****장애 복구 및 롤백 전략****

### ****장애 복구 전략****

|  |  |  |  |
| --- | --- | --- | --- |
| **장애 유형** | **발생 원인** | **대응 방안** | **복구 시간 목표 (RTO)** |
| **Database 장애** | - Oracle, MySQL 장애 - Replication 지연 | - Standby DB로 즉시 전환 - 복구 후 동기화 실행 | 10분 이내 |
| **Microservice 장애** | - Pod Crash - OutOfMemory Exception | - Kubernetes Auto Healing - ArgoCD 롤백 처리 | 5분 이내 |
| **네트워크 장애** | - API Gateway 다운 - 트래픽 폭주로 인한 응답 불가 | - Nginx와 HAProxy 이중화 구성 - Circuit Breaker 설정 | 5분 이내 |
| **ETL Job 실패** | - DataStage Job 중단 - Spark Task 오류 | - Control-M에서 Job 재시작 - Spark Streaming 복구 | 10분 이내 |
| **Kafka 장애** | - Broker 다운 - 메시지 유실 발생 | - Kafka MirrorMaker를 통해 복구 - HDFS에 백업된 메시지 재처리 | 3분 이내 |

### ****롤백 전략****

|  |  |  |
| --- | --- | --- |
| **배포 전략** | **롤백 방식** | **기술 스택** |
| **Blue-Green Deployment** | - Green에서 문제가 발생하면 Blue로 즉시 트래픽 전환 | ArgoCD, Kubernetes |
| **Canary Deployment** | - Canary 테스트 중 문제가 발생하면 기존 버전으로 트래픽 복구 | Istio, Argo Rollouts |
| **Rolling Update** | - 이전 단계로 롤백 시 순차적으로 이전 Pod 복구 | Kubernetes, Helm |
| **Hot Swap Deployment** | - 예비 인스턴스로 즉시 전환하여 서비스 중단 없이 복구 | Kubernetes, Nginx |

# ****운영 및 유지보수 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **24시간 무중단 운영**을 목표로 설계되었습니다.  
**클라우드 네이티브 아키텍처**와 **Kubernetes 오케스트레이션**, **자동화된 CI/CD**를 통해  
운영 중 장애 발생 시 빠르게 복구할 수 있으며, **모니터링 및 자동화된 패치 관리**로 시스템 안정성을 보장합니다.  
본 전략은 **운영 관리, 장애 대응, 보안 패치, 데이터 백업**을 중심으로 체계적으로 구성되었습니다.

## ****운영 관리 개요****

### ****주요 운영 관리 항목****

|  |  |  |
| --- | --- | --- |
| **운영 항목** | **설명** | **적용 기술** |
| **Container Orchestration** | - Kubernetes 기반 클러스터 관리 - 서비스 오토스케일링 및 로드 밸런싱 | Kubernetes, Helm |
| **Service Discovery** | - Spring Cloud Service Discovery를 통한 마이크로서비스 탐색 | Spring Cloud Eureka |
| **Load Balancing** | - HAProxy 및 Nginx를 활용한 트래픽 분산 처리 | HAProxy, Nginx |
| **Configuration Management** | - Ansible을 활용한 설정 관리 및 자동화 배포 | Ansible, Terraform |
| **Resource Optimization** | - Prometheus와 Grafana를 통한 실시간 자원 모니터링 | Prometheus, Grafana |
| **Storage Management** | - Ceph, GlusterFS를 활용한 고가용성 스토리지 구성 | Ceph, GlusterFS |

### ****운영 관리 프로세스****

1️⃣ **Kubernetes 클러스터 관리:**

* Kubernetes 노드 상태 모니터링 및 Pod 오토스케일링
* Auto Healing 설정으로 Pod 장애 시 자동 복구

2️⃣ **Microservice Health Check:**

* Spring Boot Actuator를 활용한 실시간 Health Check
* Gateway에서 응답 시간 분석 후 비정상 서비스 격리

3️⃣ **로그 수집 및 분석:**

* Fluentd와 Logstash를 통해 실시간 로그 수집
* Kibana에서 분석 대시보드 제공

4️⃣ **알림 및 대응:**

* Prometheus AlertManager가 장애 발생 시 Slack, 이메일로 알림 전송
* 즉시 Control-M Job 재시작 및 복구 시나리오 실행

## ****장애 대응 및 복구 시나리오****

### ****장애 유형 정의 및 대응 방안****

|  |  |  |  |
| --- | --- | --- | --- |
| **장애 유형** | **원인** | **대응 방안** | **복구 시간 목표 (RTO)** |
| **애플리케이션 장애** | - Spring Boot 서비스 비정상 종료 - 메모리 누수 | - Kubernetes Auto Healing - ArgoCD 롤백 처리 | 5분 이내 |
| **DB 연결 실패** | - Oracle DB Connection Pool 초과 - MySQL 복제 실패 | - HikariCP 설정 조정 - Read Replica로 즉시 전환 | 10분 이내 |
| **네트워크 단절** | - API Gateway 장애 - 내부 네트워크 간 연결 끊김 | - Nginx 및 HAProxy를 통한 이중화 구성 - Fallback 처리 | 5분 이내 |
| **ETL Job 실패** | - DataStage Job 중단 - Spark Streaming 처리 중단 | - Control-M Job 재시작 - Spark에서 재처리 | 10분 이내 |
| **파일 전송 실패** | - MFT (Managed File Transfer) 시 파일 전송 중단 | - Fallback 경로 설정 - 실패 시 S3에 임시 저장 후 복구 | 5분 이내 |
| **메시지 큐 장애** | - Kafka Broker 다운 - 메시지 유실 발생 | - MirrorMaker로 실시간 복구 - HDFS 백업된 메시지 재처리 | 3분 이내 |

### ****장애 복구 시나리오 정의****

1️⃣ **Kubernetes 장애 발생 시:**

* Kubernetes Health Check를 통해 비정상 Pod 자동 복구
* Prometheus AlertManager가 Slack과 이메일로 알림

2️⃣ **Database 장애 발생 시:**

* Oracle RAC (Real Application Clusters)로 Standby 전환
* Read Replica를 활성화하여 서비스 중단 최소화

3️⃣ **Microservice 장애 발생 시:**

* ArgoCD를 통해 즉시 이전 배포 상태로 롤백
* ELK Stack에서 장애 원인 분석

4️⃣ **ETL Job 실패 시:**

* Control-M에서 중단된 Job을 재시작
* Spark Streaming은 체크포인트를 기준으로 재처리

## ****보안 업데이트 및 패치 관리****

### ****패치 관리 계획****

|  |  |  |  |
| --- | --- | --- | --- |
| **대상 시스템** | **패치 항목** | **주기** | **도구 및 기술** |
| **Kubernetes Cluster** | - K8s 버전 업데이트 - Helm Chart 업데이트 | 분기별 | Helm, ArgoCD |
| **Spring Boot Apps** | - Spring Framework 보안 패치 - Dependency 업데이트 | 월간 | Jenkins, SonarQube |
| **Database (Oracle, MySQL)** | - SQL Injection 방어 업데이트 - 버그 수정 패치 | 분기별 | Flyway, Liquibase |
| **Container Image** | - Docker 이미지 최신화 - 보안 취약점 점검 및 업데이트 | 주간 | Docker Hub, Clair |
| **Message Queue (Kafka)** | - Kafka Broker 및 Zookeeper 보안 패치 | 월간 | Ansible, Helm |
| **API Gateway** | - Spring Cloud Gateway 최신 버전 반영 - 보안 설정 강화 | 분기별 | Spring Cloud |

### ****패치 배포 프로세스****

1️⃣ **CI/CD 파이프라인에 패치 반영:**

* Git에 패치 업데이트 후 Jenkins Pipeline 실행
* SonarQube를 통한 코드 품질 검사

2️⃣ **Docker 이미지 생성:**

* Dockerfile에 패치 내용을 반영하여 이미지 빌드
* Docker Hub에 최신 이미지 Push

3️⃣ **Kubernetes 배포:**

* ArgoCD를 통해 Kubernetes 클러스터에 배포
* 무중단 Rolling Update 방식 적용

4️⃣ **모니터링 및 검증:**

* Prometheus와 Grafana로 실시간 모니터링
* 장애 발생 시 즉시 롤백 처리

## ****데이터 백업 및 복구 전략****

### ****데이터 백업 구성 요소****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **Database Backup** | - Oracle, MySQL DB 스냅샷 백업 - 주기적 Dump 파일 생성 | Oracle RMAN, mysqldump |
| **HDFS Backup** | - HDFS에 적재된 빅데이터 주기적 스냅샷 | HDFS Snapshots |
| **Object Storage Backup** | - AWS S3에 중요한 데이터 백업 - Cross-Region Replication | AWS S3, Glacier |
| **ETL Data Backup** | - DataStage Job 결과물 주기적 백업 | Control-M, S3 Backup |
| **Configuration Backup** | - Kubernetes 설정 파일, Helm Charts 백업 | GitOps (ArgoCD) |

### ****데이터 복구 프로세스****

1️⃣ **DB 복구:**

* Oracle RMAN과 MySQL Dump 파일을 통해 신속 복구
* Standby DB로 자동 전환

2️⃣ **HDFS 복구:**

* HDFS Snapshots을 활용하여 삭제된 데이터 복구
* 장애 발생 15분 이내 복구 목표 설정

3️⃣ **Object Storage 복구:**

* AWS S3에서 Cross-Region 복구
* 데이터 유실 시 Glacier에서 복구 가능

4️⃣ **ETL Job 복구:**

* Control-M에서 중단된 Job 재시작
* Spark 체크포인트 기준으로 재처리

# ****테스트 및 검증 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 금융 거래의 **안정성, 무결성, 성능 최적화**를 보장하기 위해  
다양한 테스트 및 검증 절차를 수행합니다.  
테스트는 **기능 테스트, 성능 테스트, 보안 테스트, 복구 시나리오 테스트**로 구분되며,  
각 단계별로 테스트 시나리오 정의, 테스트 자동화, 결과 분석을 통해 결함을 조기에 탐지하고,  
안정성을 극대화하기 위한 방안을 도출합니다.

## ****테스트 시나리오 정의****

### ****테스트 유형 정의****

|  |  |  |
| --- | --- | --- |
| **테스트 유형** | **설명** | **도구 및 기술 스택** |
| **기능 테스트** | - 각 애플리케이션의 기능이 올바르게 작동하는지 검증 - REST API 호출 및 상태 확인 | Postman, SoapUI, JUnit |
| **통합 테스트** | - Microservices 간의 통신 및 데이터 연계 상태 검증 | Spring Cloud Contract, WireMock |
| **성능 테스트** | - 대량 트랜잭션 처리 시 응답 시간, TPS(Transaction Per Second) 측정 | JMeter, Gatling |
| **부하 테스트** | - 동시 사용자 접속 시 시스템의 안정성 및 처리 성능 검증 | Locust, JMeter |
| **보안 테스트** | - SQL Injection, XSS, CSRF, 인증 및 권한 취약점 검증 | OWASP ZAP, Burp Suite |
| **장애 복구 테스트** | - 장애 발생 시 자동 복구 및 Failover 처리 확인 | Chaos Monkey, Gremlin |
| **배포 테스트** | - Blue-Green, Rolling Update, Canary 배포 시 무중단 테스트 | ArgoCD, Helm |

### ****주요 테스트 시나리오 정의****

|  |  |  |
| --- | --- | --- |
| **시나리오 명** | **설명** | **성공 기준** |
| **사용자 로그인 시 인증 처리** | - 사용자가 올바른 인증 정보를 입력했을 때 로그인 성공 여부 검증 | 200 OK 응답, JWT Token 발급 확인 |
| **API Gateway 라우팅 테스트** | - API Gateway를 통해 올바른 Microservice로 전달되는지 확인 | 정상적인 마이크로서비스 라우팅 확인 |
| **트랜잭션 처리 테스트** | - 금융 거래 처리 시 데이터베이스에 정상 반영 여부 확인 | 트랜잭션 완료 후 DB에 커밋된 데이터 확인 |
| **실시간 알림 서비스 테스트** | - 거래 발생 시 사용자에게 실시간 알림 전송 여부 확인 | 1초 이내 알림 수신 확인 |
| **캠페인 메시지 발송 테스트** | - 마케팅 캠페인 실행 시 대상자에게 메시지 정상 발송 확인 | 95% 이상 수신율 |
| **대량 데이터 적재 테스트** | - HDFS에 1TB 이상의 데이터를 적재 시 처리 시간 및 안정성 확인 | 5분 이내 적재 완료, 데이터 유실 없음 |
| **DR (Disaster Recovery) 테스트** | - 장애 발생 시 DR 센터로의 자동 Failover 처리 여부 확인 | 15분 이내 복구 완료 |
| **Blue-Green 배포 테스트** | - Blue 환경에서 Green으로 무중단 전환 시 정상 작동 여부 확인 | 전환 시 요청 실패율 0% 유지 |

## ****성능 테스트 및 결과 분석****

### ****성능 테스트 목표 정의****

|  |  |  |
| --- | --- | --- |
| **테스트 항목** | **목표 기준** | **테스트 도구** |
| **TPS (Transaction Per Second)** | - 초당 10,000 트랜잭션 처리 | JMeter, Gatling |
| **응답 시간 (Response Time)** | - 99%의 요청이 2초 이내 응답 | Locust, JMeter |
| **최대 동시 접속자 수** | - 최대 20,000명 동시 접속 시 안정성 유지 | JMeter, Blazemeter |
| **메모리 사용량 (Memory Usage)** | - 80% 이상의 메모리 사용 시 자동 확장 | Prometheus, Grafana |
| **CPU 사용량 (CPU Usage)** | - CPU 사용량 75% 이상 시 오토스케일링 트리거 | Prometheus, Grafana |
| **네트워크 처리량 (Network Throughput)** | - 초당 5Gbps 이상 처리 가능 | Wireshark, Prometheus |

### ****성능 테스트 시나리오 정의****

|  |  |  |
| --- | --- | --- |
| **테스트 명** | **설명** | **성공 기준** |
| **로그인 처리 테스트** | - 1,000명의 사용자가 동시에 로그인 요청 시 정상 처리 여부 | 1초 이내 응답, 99% 이상 성공율 |
| **대량 거래 처리 테스트** | - 1만 건의 거래를 동시에 처리했을 때 오류 없이 완료되는지 확인 | 5초 이내 완료, DB 커밋 상태 정상 |
| **API Gateway 부하 테스트** | - API Gateway를 통한 20,000 TPS 처리 시 응답 시간 검증 | 평균 응답 시간 2초 이내, 95% 이상 성공율 |
| **파일 전송 처리 테스트** | - MFT를 통해 1GB 파일 전송 시 정상 처리 여부 | 1분 이내 전송 완료, 데이터 유실 없음 |
| **Batch Job 테스트** | - DataStage Job이 대용량 데이터를 클렌징하여 HDFS에 적재 | 10분 이내 클렌징 및 적재 완료 |

## ****보안 테스트 및 취약점 분석****

### ****보안 테스트 항목 정의****

|  |  |  |
| --- | --- | --- |
| **보안 테스트 항목** | **설명** | **도구 및 기술 스택** |
| **SQL Injection 탐지** | - 쿼리 문자열에 악성 코드가 삽입될 때 시스템이 손상되지 않도록 검증 | OWASP ZAP, SQLMap |
| **XSS (Cross-Site Scripting)** | - 사용자의 브라우저에 악성 스크립트가 실행되지 않도록 방어 | Burp Suite, OWASP ZAP |
| **CSRF (Cross-Site Request Forgery)** | - 외부 사이트에서 사용자 인증을 도용하지 못하도록 방지 | OWASP ZAP, Burp Suite |
| **OAuth 인증 취약점 탐지** | - OAuth2.0 인증 처리 시 권한 탈취 및 세션 하이재킹 방지 | OWASP ZAP, Postman |
| **API Security** | - REST API 호출 시 인증 및 권한이 올바르게 처리되는지 확인 | Postman, SoapUI |

### ****14.4.2 보안 테스트 시나리오 정의****

|  |  |  |
| --- | --- | --- |
| **테스트 명** | **설명** | **성공 기준** |
| **SQL Injection 방어 테스트** | - 쿼리 파라미터에 악성 SQL 문이 삽입될 때 차단 여부 확인 | HTTP 403 Forbidden, DB Query 실행 안 됨 |
| **XSS 방어 테스트** | - 클라이언트 요청에 악성 스크립트가 포함될 때 실행 차단 여부 확인 | 브라우저 경고 메시지, 페이지 무반응 |
| **OAuth 인증 우회 방지** | - 인증 토큰을 위조하거나 세션 하이재킹이 발생하지 않는지 확인 | 401 Unauthorized 응답 |
| **API Rate Limiting 테스트** | - 동일 IP에서 과도한 API 호출 시 차단 여부 검증 | HTTP 429 Too Many Requests 응답 |
| **CSRF 보호 테스트** | - CSRF 공격 시 인증이 우회되지 않도록 방어 | 403 Forbidden 응답 |

# ****성능 최적화 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **실시간 거래 처리, 금융 데이터 분석, 마케팅 캠페인 실행**을  
최적의 성능으로 운영하기 위해 설계되었습니다.  
본 장에서는 **애플리케이션 최적화, 데이터베이스 최적화, 네트워크 최적화, 캐싱 전략**에 대한  
구체적인 방안을 제시하여, 처리 속도와 응답 시간을 극대화하고 자원 사용을 최적화하는 전략을 설명합니다.

## ****애플리케이션 성능 최적화****

### ****애플리케이션 최적화 방안****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **최적화 방안** | **적용 기술** |
| **Spring Boot Application** | - Lazy Initialization을 통해 불필요한 Bean 로딩 최소화 - WebFlux 비동기 처리 적용 | Spring Boot, Spring WebFlux |
| **API Gateway** | - Spring Cloud Gateway에서 Rate Limiting 적용 - Circuit Breaker 패턴 적용 | Spring Cloud Gateway, Resilience4J |
| **Session Management** | - Redis 기반의 Session Clustering으로 세션 공유 | Redis, Spring Session |
| **Thread Pool 최적화** | - Fixed Thread Pool을 도입하여 스레드 관리 최적화 | Spring Task Executor |
| **Non-blocking I/O** | - Netty 기반의 비동기 I/O 처리 | Spring WebFlux, Netty |
| **메모리 관리** | - G1 Garbage Collector 사용 - JVM Heap Size 최적화 | G1 GC, JVM Options |

### ****애플리케이션 성능 테스트 시나리오****

|  |  |  |
| --- | --- | --- |
| **테스트 명** | **설명** | **성공 기준** |
| **대량 거래 처리 테스트** | - 10,000 TPS를 동시에 처리 시 응답 시간 측정 | 평균 응답 시간 2초 이내, 95% 이상 성공 |
| **API Gateway 부하 테스트** | - 20,000명 동시 접속 시 API Gateway의 성능 검증 | 평균 응답 시간 1.5초 이내 유지 |
| **캠페인 메시지 발송 테스트** | - 100만 건의 캠페인 메시지 동시 발송 시 손실률 확인 | 메시지 손실률 0.01% 이하 |
| **세션 클러스터링 테스트** | - Redis 기반 세션 클러스터링의 장애 복구 및 재연결 시간 측정 | 장애 발생 시 2초 이내 재연결 |
| **비동기 처리 테스트** | - WebFlux 기반 비동기 API 처리 시 응답 시간 측정 | 1초 이내 응답 완료, 98% 이상 성공 |

## ****데이터베이스 성능 최적화****

### ****데이터베이스 최적화 방안****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **최적화 방안** | **적용 기술** |
| **쿼리 최적화** | - Index 최적화 - 쿼리 캐싱 적용 - 페이징 처리 최적화 | MySQL, Oracle |
| **DB 파티셔닝** | - 테이블 파티셔닝을 통해 대용량 데이터의 조회 성능 개선 | Oracle Partitioning |
| **Connection Pool** | - HikariCP 사용으로 DB 커넥션 풀 관리 최적화 | HikariCP, Spring Data |
| **읽기/쓰기 분리** | - Master-Slave 구조로 트랜잭션 처리 성능 향상 | MySQL Replication |
| **데이터 압축** | - 대용량 데이터 압축으로 I/O 성능 최적화 | Oracle Advanced Compression |
| **Batch Processing** | - 대량 데이터 처리 시 Batch Insert 적용 | Spring Batch |

### ****데이터베이스 성능 테스트 시나리오****

|  |  |  |
| --- | --- | --- |
| **테스트 명** | **설명** | **성공 기준** |
| **대량 조회 테스트** | - 1천만 건의 데이터 조회 시 응답 시간 측정 | 2초 이내 응답 완료 |
| **Batch Insert 테스트** | - 100만 건의 데이터 일괄 Insert 시 처리 시간 확인 | 10초 이내 완료 |
| **읽기/쓰기 분리 테스트** | - Master-Slave 구조에서 읽기/쓰기 성능 검증 | Master: 10ms, Slave: 5ms 이내 |
| **Index 활용 테스트** | - Index를 통한 검색 속도 향상 측정 | 쿼리 응답 시간 50% 이상 단축 |
| **Connection Pool 테스트** | - 1만 개의 동시 연결 시 Connection Pool 안정성 확인 | Connection Timeout 없음 |

## ****네트워크 성능 최적화****

### ****네트워크 최적화 방안****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **최적화 방안** | **적용 기술** |
| **로드 밸런싱** | - HAProxy와 Nginx를 활용한 트래픽 분산 | HAProxy, Nginx |
| **압축 전송** | - Gzip Compression을 통한 데이터 전송 속도 향상 | Nginx, Spring Boot |
| **HTTP/2 사용** | - 멀티플렉싱을 통해 하나의 연결로 다수 요청 처리 | Nginx, Spring MVC |
| **Connection Pooling** | - HTTP Keep-Alive 설정으로 연결 재사용 | Spring RestTemplate |
| **Content Delivery Network** | - 정적 리소스 캐싱 및 전송 속도 최적화 | CloudFront, Akamai |
| **멀티 리전 구성** | - 주요 데이터는 다중 리전에 분산 저장하여 지연 시간 최소화 | AWS Multi-Region |

### ****네트워크 성능 테스트 시나리오****

|  |  |  |
| --- | --- | --- |
| **테스트 명** | **설명** | **성공 기준** |
| **로드 밸런싱 테스트** | - 10만 TPS의 트래픽을 HAProxy로 분산 처리 | 서버 과부하 없이 99% 이상 응답 성공 |
| **압축 전송 테스트** | - 1GB 파일 전송 시 Gzip Compression 적용 후 전송 시간 측정 | 50% 이상 전송 시간 단축 |
| **HTTP/2 멀티플렉싱 테스트** | - 다중 API 요청 시 단일 커넥션에서 응답 시간 비교 | 응답 시간 30% 이상 단축 |
| **CDN 캐싱 테스트** | - 정적 리소스의 CDN 캐싱 여부 확인 | 90% 이상 캐시 Hit Rate |
| **멀티 리전 전송 테스트** | - AWS Multi-Region에서 데이터 복제 및 전송 시간 측정 | 평균 전송 시간 2초 이내 유지 |

## ****캐싱 전략 및 개선 방안****

|  |  |  |
| --- | --- | --- |
| **캐싱 항목** | **캐싱 기술** | **설명** |
| **API Response Cache** | - Redis, Spring Cache | API 호출 시 자주 조회되는 데이터를 캐싱하여 응답 시간 단축 |
| **Session Cache** | - Redis, Spring Session | 세션 정보를 In-Memory에 저장하여 빠른 인증 처리 |
| **DB Query Cache** | - MyBatis Second Level Cache | 반복 쿼리에 대한 캐싱을 통해 DB 부하 최소화 |
| **Static Resource Cache** | - CDN, CloudFront | 정적 파일(CSS, JS, Image)을 캐싱하여 로딩 시간 단축 |
| **Data Lake Query Cache** | - AWS Athena, HDFS | 대용량 분석 쿼리의 결과를 캐싱하여 반복 조회 시 빠른 응답 제공 |

# ****아키텍처 개선 및 확장 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **Microservices Architecture**를 중심으로  
금융 서비스의 확장성과 유연성을 극대화하기 위해 설계되었습니다.  
본 장에서는 **아키텍처 개선 방안, 글로벌 확장 전략, 멀티 리전 구성**에 대해 설명하며,  
이를 통해 금융 데이터의 안정적 처리와 글로벌 서비스 확장을 목표로 합니다.

## ****아키텍처 개선 방안****

### ****개선 목표 정의****

* **확장성 확보:** 금융 서비스 증가에 따른 트래픽 대응
* **유연한 배포:** Blue-Green Deployment 및 Canary Release 도입
* **지속적 확장:** Kubernetes 기반의 오토스케일링 설정
* **API 안정성:** Circuit Breaker 패턴과 Rate Limiting 적용
* **보안 강화:** Zero Trust Architecture 도입

### ****주요 개선 방안****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **개선 사항** | **적용 기술** |
| **API Gateway** | - Spring Cloud Gateway에서 Rate Limiting 적용 - Circuit Breaker 도입 | Spring Cloud Gateway, Resilience4J |
| **Service Discovery** | - Spring Cloud Eureka를 활용한 마이크로서비스 동적 탐색 | Spring Cloud Eureka |
| **Service Mesh** | - Istio를 도입하여 마이크로서비스 간 보안 트래픽 관리 및 트래픽 라우팅 | Istio, Envoy Proxy |
| **데이터 레이크 확장** | - AWS S3와 HDFS를 연동하여 대용량 데이터 통합 관리 | HDFS, AWS S3 |
| **ETL 최적화** | - Spark Streaming으로 실시간 데이터 수집 및 전처리 최적화 | Apache Spark, DataStage |
| **오토스케일링 설정** | - Kubernetes Horizontal Pod Autoscaler 설정 | Kubernetes, Helm |
| **모니터링 개선** | - Prometheus와 Grafana를 통한 실시간 리소스 모니터링 강화 | Prometheus, Grafana |

### ****개선 아키텍처 구성도****

[Channel Layer]

- Mobile Banking

- Internet Banking

- ATM Service

- API Gateway (Spring Cloud Gateway)

|

[Service Mesh Layer]

- Istio (Service-to-Service Security, Traffic Management)

|

[Microservices Layer]

- Customer Service

- Transaction Service

- Marketing Service

- Analytics Service

|

[Data Layer]

- HDFS (Big Data Storage)

- Oracle DB, MySQL (Transactional Data)

- AWS S3 (Object Storage)

|

[Integration Layer]

- Kafka (Streaming)

- DataStage (ETL)

|

[Monitoring & Logging Layer]

- Prometheus, Grafana

- ELK Stack (Elasticsearch, Logstash, Kibana)

## ****확장 전략 및 로드맵****

### ****확장 전략****

|  |  |  |
| --- | --- | --- |
| **확장 대상** | **확장 전략** | **적용 기술** |
| **서비스 확장** | - Microservices 구조로 신규 금융 서비스 유연하게 추가 - API Gateway에 라우팅 설정 | Spring Boot, Spring Cloud Gateway |
| **데이터 레이크 확장** | - AWS S3 버킷 추가를 통해 실시간 데이터 적재 - HDFS 클러스터 확장 | HDFS, AWS S3 |
| **글로벌 확장** | - Multi-Region 구성으로 글로벌 사용자 서비스 제공 | AWS Route 53, CloudFront |
| **오토스케일링 설정** | - Horizontal Pod Autoscaler를 통해 트래픽 급증 시 자동 확장 | Kubernetes, Helm |
| **멀티 클러스터 지원** | - Kubernetes Federation을 통해 다중 클러스터 관리 | Kubernetes Federation |

### ****확장 로드맵****

|  |  |  |  |
| --- | --- | --- | --- |
| **단계** | **기간** | **주요 내용** | **완료 조건** |
| **1단계:** 서비스 분리 및 마이크로서비스 전환 | 1~3개월 | - 기존 Monolithic 구조를 Microservices로 전환 | - 모든 서비스가 독립적으로 배포되고 관리됨 |
| **2단계:** 데이터 레이크 확장 및 분석 최적화 | 4~6개월 | - HDFS 및 AWS S3에 대용량 데이터 통합 - Spark Streaming 최적화 | - 분석 쿼리 시간 30% 이상 단축 |
| **3단계:** 글로벌 확장 준비 | 6~9개월 | - AWS Multi-Region 설정 - API Gateway의 Global Routing | - 글로벌 사용자 200ms 이내 응답 시간 확보 |
| **4단계:** 오토스케일링 및 멀티 클러스터 확장 | 9~12개월 | - Kubernetes Auto Scaling 및 Federation 설정 | - 10만 TPS 대응 테스트 완료 |

## ****글로벌 확장 및 멀티 리전 전략****

### ****글로벌 확장 전략 정의****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **적용 기술** |
| **Global Load Balancer** | - AWS Route 53을 통해 글로벌 사용자 트래픽 분산 | AWS Route 53 |
| **Multi-Region Deployment** | - AWS Multi-Region 구성으로 글로벌 장애 대응 | AWS S3, RDS, EC2 |
| **Data Replication** | - Cross-Region Replication을 통해 데이터 동기화 | AWS S3, HDFS |
| **Geo-Distributed Caching** | - CloudFront를 활용한 전세계 콘텐츠 캐싱 | AWS CloudFront |
| **Latency Optimization** | - 사용자 위치 기반 라우팅 설정으로 최적의 응답 시간 확보 | AWS Global Accelerator |

### ****글로벌 확장 아키텍처 구성도****

[Global Load Balancer]

- AWS Route 53

|

[Multi-Region Deployment]

- US-East (Primary Region)

- EU-West (Secondary Region)

- AP-Northeast (Backup Region)

|

[Data Synchronization Layer]

- AWS S3 Cross-Region Replication

- HDFS Geo-Replication

|

[Application Layer]

- Spring Boot Microservices

|

[Monitoring & Disaster Recovery]

- Prometheus, Grafana

- AWS CloudWatch, ELK Stack

### ****글로벌 확장 시나리오 정의****

|  |  |  |
| --- | --- | --- |
| **시나리오 명** | **설명** | **성공 기준** |
| **Multi-Region Failover Test** | - Primary Region 장애 발생 시 Secondary Region으로 트래픽 전환 | 15초 이내 트래픽 정상 전환 |
| **Geo-Distributed Caching Test** | - CloudFront에 캐싱된 정적 리소스의 글로벌 응답 시간 테스트 | 200ms 이하 응답 시간 유지 |
| **Cross-Region Data Sync Test** | - AWS S3 버킷 간 데이터 동기화 지연 시간 측정 | 3초 이내 데이터 동기화 완료 |
| **Global Load Balancer Test** | - 글로벌 사용자 접속 시 최적의 리전으로 라우팅 처리 | 평균 응답 시간 2초 이하 |

# ****보안 및 규제 준수 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 금융권의 **보안 규정 및 규제 준수**를 철저히 반영하여 설계되었습니다.  
**금융보안원(FSI) 권고 사항**, **전자금융거래법**, **개인정보 보호법**, **신용정보의 이용 및 보호에 관한 법률**을  
기반으로 금융 데이터의 기밀성, 무결성, 가용성을 보장합니다.  
본 장에서는 **금융 규제 준수 전략, 데이터 보안 및 암호화, 접근 제어 및 인증 체계, 감사 및 로깅 전략**에 대해 설명합니다.

## ****금융권 규제 준수 전략****

### ****금융권 주요 규제 사항 정의****

|  |  |  |
| --- | --- | --- |
| **규제 항목** | **설명** | **적용 기술** |
| **전자금융거래법** | - 금융 데이터의 안전한 전송 및 저장 - 전자 금융거래의 인증 및 보안 강화 | Spring Security, TLS 1.3 |
| **개인정보 보호법** | - 민감 정보 암호화 및 최소 권한 접근 원칙 준수 | AES-256 Encryption, RBAC |
| **신용정보의 이용 및 보호에 관한 법률** | - 고객의 신용 정보와 거래 정보의 무결성 및 보호 | Oracle TDE, Vault |
| **정보통신망 이용촉진 및 정보보호법** | - 데이터 유출 방지 및 네트워크 보안 강화 | IPS, WAF, IDS |
| **금융보안원(FSI) 권고 사항** | - 금융 거래의 무결성 보장 및 장애 대응 전략 | HSM (Hardware Security Module) |

### ****규제 준수 아키텍처 구성****

[Channel Layer]

- Mobile Banking

- Internet Banking

- API Gateway

|

[Security Layer]

- TLS 1.3 (전송 구간 암호화)

- OAuth2.0, JWT (인증 및 접근 제어)

- AES-256 (저장 데이터 암호화)

|

[Service Layer]

- Customer Information Service

- Transaction Management Service

- Marketing Campaign Service

|

[Data Layer]

- Oracle TDE (Transparent Data Encryption)

- AWS KMS (Key Management Service)

- HSM (Hardware Security Module)

|

[Monitoring & Audit Layer]

- Prometheus, Grafana

- ELK Stack (Elasticsearch, Logstash, Kibana)

## ****데이터 보안 및 암호화 전략****

### ****전송 구간 암호화 (Encryption in Transit)****

|  |  |  |
| --- | --- | --- |
| **암호화 대상** | **암호화 기술** | **설명** |
| **API 요청/응답** | TLS 1.3 | - 모든 API 요청과 응답은 TLS 1.3을 통해 암호화 전송 |
| **파일 전송** | SFTP, MFT | - 대용량 파일 전송 시 SFTP 및 MFT(Multi File Transfer) 사용 |
| **메시지 큐 통신** | SSL/TLS | - Kafka 및 RabbitMQ 통신 시 SSL 적용 |
| **Batch Job 데이터 전송** | HTTPS, IPSec VPN | - 배치 처리 시 HTTPS 및 IPSec VPN으로 보안 전송 |

### ****저장 데이터 암호화 (Encryption at Rest)****

|  |  |  |
| --- | --- | --- |
| **데이터 유형** | **암호화 기술** | **적용 대상** |
| **고객 정보** | AES-256 | Oracle DB, MySQL, HDFS |
| **금융 거래 정보** | Transparent Data Encryption (TDE) | Oracle Advanced Security Option |
| **로그 파일** | AES-128 + Gzip Compression | ELK Stack (Elasticsearch) |
| **마케팅 데이터** | AES-256 | Redis, HDFS |
| **ETL Job 결과물** | PGP Encryption | DataStage, AWS S3 |

## ****접근 제어 및 인증 체계****

### ****인증 체계 설계****

|  |  |  |
| --- | --- | --- |
| **인증 방식** | **설명** | **적용 기술** |
| **OAuth2.0** | - 금융권 표준 인증 방식으로, 토큰 기반의 인증을 제공 - JWT 발급을 통해 접근 권한 관리 | Spring Security, Keycloak |
| **SSO (Single Sign-On)** | - 통합 로그인 시스템을 통해 하나의 인증으로 여러 서비스 접근 - 금융 계열사 간 연동 | SAML, OpenID Connect |
| **MFA (Multi-Factor Authentication)** | - 2단계 인증을 통해 보안 강화 (OTP, SMS 인증 등) | Google Authenticator, Duo |
| **RBAC (Role-Based Access Control)** | - 역할 기반의 세분화된 권한 관리 | Apache Ranger, Keycloak |
| **IAM (Identity Access Management)** | - 클라우드 리소스에 대한 접근 제어 및 정책 설정 | AWS IAM, Azure AD |

### ****접근 제어 설계****

|  |  |  |
| --- | --- | --- |
| **접근 대상** | **접근 권한** | **보안 적용** |
| **Database** | - DBA, Data Analyst, Application User | Oracle TDE, RBAC |
| **API Gateway** | - Admin, Service User | OAuth2.0, JWT |
| **ETL Pipeline** | - Data Engineer, Data Analyst | Role-Based Access Control |
| **Monitoring System** | - DevOps, IT Admin | Prometheus, Grafana |
| **HDFS Cluster** | - Data Engineer, Data Scientist | Kerberos, HDFS ACLs |

## ****감사 및 로깅 전략****

### ****감사 및 로깅 구성 요소****

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **적용 기술** |
| **Audit Log** | - 금융 거래 기록, API 요청/응답 내역, 인증 로그 기록 - 7년간 보관 | Elasticsearch, Logstash |
| **Security Log** | - 보안 이벤트 발생 시 자동 로깅 (인증 실패, 권한 탈취 시도 등) | Kibana, Prometheus |
| **Database Log** | - SQL Injection 시도, 데이터 변경 이력 추적 | Oracle Audit Vault |
| **File Access Log** | - SFTP 및 MFT를 통한 파일 접근 이력 기록 | Fluentd, Filebeat |
| **System Event Log** | - Spring Boot 애플리케이션 상태 및 에러 로그 | Spring Actuator, Logback |

### ****감사 로깅 시나리오 정의****

|  |  |  |
| --- | --- | --- |
| **시나리오 명** | **설명** | **성공 기준** |
| **로그인 시도 기록** | - 사용자의 로그인 시도 기록 및 실패 이력 확인 | 모든 시도에 대해 100% 기록 |
| **API 요청 로깅** | - 각 서비스로 전달된 API 요청 내역 로깅 | 요청 시간, 사용자 정보, 요청 결과 |
| **DB 변경 이력 기록** | - 금융 데이터 변경 시 이력 추적 (CREATE, UPDATE, DELETE) | 변경 내역과 변경자 정보 기록 |
| **파일 접근 이력 로깅** | - SFTP 및 MFT 파일 접근 기록 | IP 주소, 접근 시간, 파일 경로 |
| **보안 이벤트 탐지** | - 이상 거래 탐지 시 보안 이벤트 기록 | ELK Stack에 실시간 로깅 |

# 마이그레이션 및 연계 전략

## **마이그레이션 전략**

* + **단계적 마이그레이션 방안**
    - 기존 계열사 데이터베이스의 데이터 정합성 검토 후 단계적 마이그레이션 진행
    - OLAP, RDW, ADW의 데이터를 순차적으로 통합 데이터 플랫폼으로 이관
    - **Zero Downtime Migration** 기법을 적용하여 서비스 중단 없이 이전
    - CDC(Change Data Capture)를 활용한 실시간 동기화 유지
  + **데이터 정제 및 표준화**
    - 이관 전 Data Cleansing 작업 수행
    - 표준화되지 않은 데이터 포맷을 ISO 표준에 맞춰 통합
    - 마스터 데이터 관리(MDM) 전략을 기반으로 중복 및 오류 제거
  + **검증 및 테스트**
    - 마이그레이션 후 데이터 정합성 테스트 (Hash Check, Row Count Validation)
    - 분석 환경 및 OLAP 연동 테스트
    - 마이그레이션 후 1개월간 운영 중 데이터 무결성 모니터링

## **연계 전략**

* + **내부 시스템 간 연계**
    - API Gateway(Spring Cloud Gateway) 기반의 통합 인터페이스 관리
    - Kafka를 활용한 비동기 메시지 전달 및 이벤트 기반 처리
    - EAI(Enterprise Application Integration)를 통한 실시간 데이터 교환
  + **외부 금융기관 및 공공기관 연계**
    - \*\*FEP (Front-End Processor)\*\*를 통한 금융기관 데이터 연동
    - 표준 금융 API(ISO 20022) 및 SOAP API를 통한 연계 설계
    - 공공기관 연계 시 오픈 API 방식과 OAuth2.0 기반 인증 체계 적용
  + **파일 전송 연계**
    - EZGATOR, FOS Storage를 통한 대용량 파일 전송 관리
    - \*\*MFT (Managed File Transfer)\*\*를 통해 보안 전송 및 로그 추적 관리
    - SFTP, FTPS 방식의 전송 암호화 적용

## **전송 구간 보안 및 데이터 무결성 보장**

* + **TLS 1.3**과 **IPSec**을 통한 데이터 전송 구간 암호화
  + 전송 중 데이터 변조 및 유실 방지를 위한 Hash Checksum 적용
  + 데이터 수신 시 **Kafka Consumer Group**을 활용하여 중복 수신 방지

## **오류 대응 및 복구 전략**

* + **Retry Logic**을 통한 재시도 처리
    - 네트워크 장애 시 3회까지 자동 재전송 시도
  + \*\*Dead Letter Queue (DLQ)\*\*를 활용하여 실패한 메시지 추적 및 재처리
  + 실시간 모니터링을 통한 장애 발생 시 알림 전송 및 대응

## **성능 최적화 및 확장성 고려**

* + **Streaming 방식**을 통한 실시간 데이터 처리 최적화
  + OLAP 연계 시 멀티스레드 기반 데이터 처리
  + Kubernetes 기반의 오토스케일링 설정으로 유연한 확장성 보장
  + 데이터 마이그레이션 중 발생하는 부하를 최소화하기 위한 리소스 분리

# ****비용 최적화 및 분석 전략****

## ****도입 전 안내말****

NH 금융지주의 통합 데이터 플랫폼은 **클라우드 네이티브 아키텍처**를 기반으로 설계되었으며,  
**운영 비용 절감, 리소스 최적화, 데이터 관리 비용 절감**을 목표로 합니다.  
본 장에서는 **클라우드 비용 최적화, 데이터 저장소 비용 최적화, 네트워크 비용 최적화**를 위한 전략을 수립하고,  
효율적인 운영을 위한 **비용 분석 및 최적화 방안**을 설명합니다.

## ****클라우드 비용 최적화****

### ****최적화 대상 정의****

|  |  |  |
| --- | --- | --- |
| **최적화 대상** | **설명** | **적용 기술** |
| **Compute 비용 최적화** | - Kubernetes Auto Scaling을 통해 필요 시에만 Pod 확장 | Kubernetes, Helm |
| **Storage 비용 최적화** | - 사용하지 않는 EBS Volume 탐색 및 제거 - S3 Intelligent Tiering 적용 | AWS EBS, AWS S3 |
| **Container 비용 최적화** | - Unused Docker Containers 및 Images 정리 | Docker, Kubernetes |
| **Serverless 최적화** | - Lambda 함수의 실행 시간을 최적화 - Cold Start 방지 설정 | AWS Lambda, DynamoDB |
| **Reserved Instance 활용** | - EC2와 RDS의 Reserved Instance 구매를 통해 비용 절감 | AWS EC2, RDS |
| **Spot Instance 활용** | - 비핵심 서비스에 Spot Instance 적용하여 비용 절감 | AWS EC2 Spot Instances |

### ****최적화 방안****

|  |  |  |
| --- | --- | --- |
| **최적화 항목** | **설명** | **예상 절감 비용** |
| **Auto Scaling** | - Kubernetes Horizontal Pod Autoscaler 설정을 통해 트래픽 변화에 따른 자동 확장 및 축소 | 30% 비용 절감 |
| **S3 Tiering** | - AWS S3 Intelligent Tiering을 통해 장기 미사용 파일 비용 감소 | 20% 비용 절감 |
| **EBS Volume 최적화** | - 사용하지 않는 EBS Volume 삭제 및 스냅샷 정리 | 15% 비용 절감 |
| **Lambda 최적화** | - 불필요한 Lambda 함수 종료 및 실행 시간 최적화 | 10% 비용 절감 |
| **Reserved Instance 구매** | - 3년 약정의 Reserved Instance 구매 시 60% 할인 적용 | 40% 비용 절감 |
| **Spot Instance 사용** | - Batch Job 및 비핵심 서비스에 Spot Instance 도입 | 70% 비용 절감 |

## ****데이터 저장소 비용 최적화****

### ****최적화 대상 정의****

|  |  |  |
| --- | --- | --- |
| **최적화 대상** | **설명** | **적용 기술** |
| **Data Lake 최적화** | - HDFS 및 AWS S3에 중복 데이터 제거 - 데이터 압축을 통한 저장소 절감 | HDFS, AWS S3 |
| **RDS 비용 최적화** | - Aurora Serverless를 도입하여 트래픽에 따라 자동 확장 및 축소 | AWS Aurora |
| **Data Archive 전략** | - 자주 접근하지 않는 데이터는 Glacier Deep Archive로 이동 | AWS Glacier |
| **Backup 비용 절감** | - Full Backup 대신 Incremental Backup으로 전환 | AWS Backup, Oracle RMAN |
| **Caching 사용** | - Redis 및 Memcached를 통한 DB 접근 최소화 | Redis, Memcached |
| **Partitioning 최적화** | - Oracle 및 MySQL Partitioning으로 읽기 성능 및 저장 공간 최적화 | Oracle Partitioning |

### ****최적화 방안****

|  |  |  |
| --- | --- | --- |
| **최적화 항목** | **설명** | **예상 절감 비용** |
| **S3 Intelligent Tiering** | - 자주 접근하지 않는 객체를 자동으로 비용 효율적인 스토리지 클래스로 이동 | 20% 비용 절감 |
| **Glacier Archive** | - 1년 이상 미사용 데이터는 Glacier로 이동 | 30% 비용 절감 |
| **Database Partitioning** | - 대규모 테이블을 파티셔닝하여 검색 성능 최적화 및 스토리지 절감 | 15% 비용 절감 |
| **HDFS Compression** | - Hadoop 데이터 압축(LZO, Snappy) 적용 | 25% 비용 절감 |
| **Incremental Backup** | - 매일 전체 백업에서 Incremental Backup으로 변경 | 40% 비용 절감 |

## ****네트워크 비용 최적화****

### ****최적화 대상 정의****

|  |  |  |
| --- | --- | --- |
| **최적화 대상** | **설명** | **적용 기술** |
| **API Gateway 최적화** | - 사용량 기반의 과금 구조 최적화 - 중복 요청 캐싱을 통한 비용 절감 | Spring Cloud Gateway, Zuul |
| **Content Delivery Network** | - CloudFront를 통한 정적 리소스 캐싱 | AWS CloudFront |
| **VPC 최적화** | - Private Subnet으로 전환하여 트래픽 비용 절감 | AWS VPC |
| **Data Transfer 최적화** | - VPC Peering 및 Direct Connect를 통한 전송 비용 절감 | AWS Direct Connect, VPC Peering |
| **인터페이스 최적화** | - 내부 통신을 Public IP 대신 Private IP로 변경 | VPC Private IP |

### ****최적화 방안****

|  |  |  |
| --- | --- | --- |
| **최적화 항목** | **설명** | **예상 절감 비용** |
| **CloudFront 사용** | - 정적 파일(CSS, JS, Image)을 CDN으로 캐싱하여 대역폭 절감 | 40% 비용 절감 |
| **VPC Peering 설정** | - 데이터 센터 간 전송 시 Private IP 사용하여 공공망 비용 절감 | 20% 비용 절감 |
| **Direct Connect 활용** | - AWS와 On-premise 데이터 전송 시 전용 네트워크를 사용하여 비용 절감 | 30% 비용 절감 |
| **API Caching** | - API Gateway에 Redis Cache를 활용하여 반복 요청 최소화 | 15% 비용 절감 |

## ****운영 비용 분석 및 최적화 방안****

|  |  |  |
| --- | --- | --- |
| **비용 항목** | **분석 항목** | **최적화 방안** |
| **Compute 비용** | - EC2, Kubernetes 클러스터 리소스 사용량 분석 - Auto Scaling 효과 | - 리소스 사용량 모니터링 - 오토스케일링 임계치 최적화 설정 |
| **Storage 비용** | - EBS 사용량 및 미사용 Volume 탐색 | - 미사용 Volume 삭제 - 필요 시 Archive로 이동 |
| **Network 비용** | - Data Transfer, NAT Gateway 사용량 분석 | - VPC Peering 전환 - CloudFront를 통한 트래픽 최적화 |
| **Database 비용** | - RDS, DynamoDB 사용량 분석 | - Aurora Serverless 적용 - Read Replica 활용 |
| **Backup 및 Archive 비용** | - S3, Glacier Backup 비용 분석 | - Full Backup 주기 조정 - Incremental Backup 전환 |

# ****결론 및 요약****

## ****프로젝트 주요 성과 요약****

NH 금융지주의 통합 데이터 플랫폼 구축 프로젝트는 **디지털 금융 혁신**을 실현하기 위해  
클라우드 네이티브 아키텍처를 기반으로 설계되었습니다.  
본 프로젝트의 주요 성과는 다음과 같습니다:

| **구분** | **주요 성과** |
| --- | --- |
| **디지털 금융 전환 실현** | - Microservices Architecture 도입으로 금융 서비스 확장성 극대화 |
| **고성능 데이터 처리** | - Kafka Streaming을 통해 초당 10,000 TPS 이상 처리 가능 |
| **안정적 서비스 운영** | - Kubernetes 오토스케일링을 통한 무중단 서비스 제공 |
| **보안 및 규제 준수 강화** | - 금융보안원(FSI) 권고 사항, 전자금융거래법, 개인정보 보호법 완벽 준수 |
| **글로벌 확장성 확보** | - AWS Multi-Region 구성으로 글로벌 금융 서비스 지원 |
| **운영 비용 최적화** | - 클라우드 리소스 최적화 및 S3 Intelligent Tiering 도입으로 30% 비용 절감 |
| **신속한 장애 대응 및 복구** | - DR(Disaster Recovery) 전략 수립으로 15분 이내 복구 가능 |

## ****기술적 개선 사항 요약****

### ****아키텍처 개선 사항****

* **Microservices Architecture 도입:**
  + 기존 Monolithic 구조에서 독립적 서비스 구조로 전환
  + Spring Cloud Gateway 및 Spring Boot를 활용한 확장성 확보
* **Data Lake 통합 관리:**
  + HDFS 및 AWS S3를 통해 금융 데이터 일원화
  + ETL Pipeline 최적화로 실시간 분석 지원
* **Service Mesh 도입:**
  + Istio를 활용한 서비스 간 통신 보안 강화
  + 사이드카 패턴 적용으로 트래픽 제어 및 보안 정책 일관성 유지
* **오토스케일링 설정:**
  + Kubernetes Horizontal Pod Autoscaler 적용
  + 부하 발생 시 자동으로 리소스 확장

### ****보안 및 규제 준수 개선 사항****

* **TLS 1.3 전송 암호화:**
  + 모든 API 통신 구간을 TLS 1.3으로 암호화하여 금융 거래의 기밀성 확보
* **AES-256 저장 암호화:**
  + 고객 정보, 거래 정보, 금융 데이터는 AES-256 방식으로 암호화 저장
* **OAuth2.0 인증 체계 도입:**
  + OAuth2.0 기반 JWT 토큰 발급을 통해 접근 제어 강화
* **Role-Based Access Control (RBAC):**
  + Spring Security와 Keycloak을 통해 역할 기반 권한 관리
* **실시간 보안 모니터링:**
  + Prometheus와 ELK Stack을 통한 실시간 보안 이벤트 모니터링

### ****성능 최적화 개선 사항****

* **API Gateway 최적화:**
  + Spring Cloud Gateway에서 Rate Limiting 및 Circuit Breaker 패턴 적용
* **Database Partitioning:**
  + 대용량 데이터 처리를 위한 파티셔닝 설정으로 쿼리 성능 최적화
* **Caching 전략 강화:**
  + Redis Cache를 통한 API 응답 시간 단축
  + MyBatis Second Level Cache를 활용한 DB 조회 최적화
* **네트워크 최적화:**
  + Nginx 및 HAProxy를 활용한 트래픽 로드밸런싱
  + HTTP/2를 도입하여 멀티플렉싱 지원

## ****비즈니스 가치 창출 요약****

|  |  |  |
| --- | --- | --- |
| **가치 창출 항목** | **설명** | **비즈니스 효과** |
| **금융 거래 처리 속도 개선** | - Kafka Streaming을 통한 실시간 거래 처리 - 초당 10,000건 이상 처리 가능 | 금융 거래 시 응답 시간 단축 및 고객 만족도 향상 |
| **디지털 금융 혁신 실현** | - 모바일 뱅킹, 인터넷 뱅킹, ATM 통합 관리 | 사용자 경험 향상 및 금융 접근성 확대 |
| **글로벌 서비스 확장 가능** | - AWS Multi-Region 구성을 통해 글로벌 금융 서비스 지원 | 해외 고객 유치 및 국제 금융 거래 처리 가능 |
| **비용 최적화 및 효율성 증대** | - 클라우드 리소스 최적화와 S3 Intelligent Tiering 도입으로 운영 비용 절감 | IT 인프라 비용 30% 절감 |
| **보안 강화 및 규제 준수** | - 금융보안원(FSI) 권고 사항, 개인정보 보호법, 전자금융거래법 준수 | 금융 사고 예방 및 법적 리스크 감소 |
| **데이터 분석 역량 강화** | - Spark Analytics와 HDFS 통합으로 실시간 데이터 분석 지원 | 마케팅 타겟팅 최적화 및 금융 리스크 분석 효율화 |

## ****향후 개선 사항 및 로드맵****

|  |  |  |  |
| --- | --- | --- | --- |
| **단계** | **기간** | **주요 개선 사항** | **완료 목표** |
| **1단계:** 완전한 MSA 전환 | 1~3개월 | - 모든 서비스의 Microservices Architecture 전환 - API Gateway 최적화 | - 서비스 간 독립적 배포 완성 |
| **2단계:** 빅데이터 분석 강화 | 4~6개월 | - Spark Streaming 최적화 - 실시간 금융 리스크 분석 도입 | - 분석 속도 30% 향상 |
| **3단계:** 글로벌 확장 준비 | 7~9개월 | - Multi-Region 확장 및 Geo-Distributed Caching 도입 | - 글로벌 서비스 레이턴시 200ms 이하 유지 |
| **4단계:** 완전한 DevOps 도입 | 10~12개월 | - GitOps 기반의 CI/CD 완성 - Kubernetes Federation 도입 | - 무중단 배포 및 다중 클러스터 관리 |
| **5단계:** AI 금융 분석 도입 | 13~18개월 | - AI 기반 금융 분석 모델 개발 - 개인화 금융 상품 추천 엔진 도입 | - 금융 상품 추천 정확도 20% 향상 |

# ****부록 및 참고자료****

## ****용어 정의****

|  |  |
| --- | --- |
| **용어** | **정의** |
| **MSA (Microservices Architecture)** | - 단일 애플리케이션을 작은 독립적 서비스들로 나누어 관리하는 아키텍처 |
| **API Gateway** | - 클라이언트 요청을 적절한 마이크로서비스로 라우팅하고, 인증 및 부하 분산을 관리하는 시스템 |
| **HDFS (Hadoop Distributed File System)** | - 대용량 데이터를 분산 처리할 수 있는 파일 시스템 |
| **S3 (Simple Storage Service)** | - AWS의 객체 스토리지 서비스로 대용량 데이터를 안전하게 보관하고 관리하는 서비스 |
| **Kubernetes** | - 컨테이너화된 애플리케이션의 배포, 확장 및 관리 자동화를 지원하는 오픈소스 플랫폼 |
| **ETL (Extract, Transform, Load)** | - 데이터를 추출하고 변환한 후 적재하는 데이터 처리 프로세스 |
| **TLS (Transport Layer Security)** | - 데이터 전송 시 암호화하여 보안을 강화하는 통신 프로토콜 |
| **IAM (Identity Access Management)** | - 사용자와 권한을 관리하여 클라우드 리소스에 대한 접근을 제어하는 시스템 |
| **RBAC (Role-Based Access Control)** | - 사용자에게 역할(Role)을 부여하고 역할에 따른 권한을 제어하는 접근 관리 방식 |
| **OAuth2.0** | - API 인증 및 권한 부여를 위한 오픈 표준 프로토콜 |
| **Istio** | - 마이크로서비스 간의 네트워크 보안, 트래픽 관리 및 모니터링을 제공하는 Service Mesh 기술 |
| **Data Lake** | - 구조화, 반구조화, 비구조화된 데이터를 대량으로 저장하고 분석할 수 있는 데이터 저장소 |
| **Prometheus** | - 애플리케이션 메트릭을 수집하고 실시간 모니터링하는 오픈소스 솔루션 |
| **Grafana** | - Prometheus 등의 메트릭을 시각화하여 대시보드 형태로 제공하는 도구 |

## ****참고 자료****

|  |  |  |
| --- | --- | --- |
| **자료명** | **출처** | **설명** |
| **Spring Boot Documentation** | <https://spring.io/projects/spring-boot> | Spring Boot 프레임워크의 공식 문서 |
| **Kubernetes Best Practices** | https://kubernetes.io/docs | Kubernetes 배포 및 오케스트레이션 관련 베스트 프랙티스 문서 |
| **AWS Well-Architected Framework** | <https://aws.amazon.com/architecture> | AWS 클라우드 아키텍처 설계 가이드 |
| **Hadoop & HDFS Guide** | <https://hadoop.apache.org> | Hadoop 및 HDFS 분산 처리 시스템에 대한 공식 문서 |
| **Kafka Streams Documentation** | <https://kafka.apache.org/documentation/streams> | 실시간 스트리밍 처리를 위한 Kafka Streams의 공식 문서 |
| **Istio Service Mesh Guide** | https://istio.io/docs | Istio를 활용한 마이크로서비스 네트워크 관리 가이드 |
| **Prometheus Monitoring Guide** | https://prometheus.io/docs | Prometheus 메트릭 수집 및 모니터링 설정 가이드 |
| **OWASP Security Guide** | <https://owasp.org> | 웹 애플리케이션 보안 및 취약점 방지에 대한 권고 사항 |

## ****아키텍처 다이어그램****

### ****전체 아키텍처 다이어그램****

* **설명:** NH 금융지주 통합 데이터 플랫폼의 전체 아키텍처 구성
* **구성 요소:**
  + Channel Layer: Mobile Banking, Internet Banking, API Gateway
  + Service Layer: Customer Service, Transaction Service, Marketing Service
  + Data Layer: HDFS, AWS S3, Oracle DB, MySQL
  + Integration Layer: Kafka, DataStage
  + Monitoring & Logging: Prometheus, Grafana, ELK Stack

[Channel Layer]

- Mobile Banking

- Internet Banking

- API Gateway

|

[Service Layer]

- Customer Service

- Transaction Service

- Marketing Service

|

[Data Layer]

- HDFS (Big Data Storage)

- AWS S3 (Object Storage)

- Oracle DB, MySQL (Transactional Data)

|

[Integration Layer]

- Kafka (Streaming)

- DataStage (ETL)

|

[Monitoring & Logging Layer]

- Prometheus, Grafana

- ELK Stack (Elasticsearch, Logstash, Kibana)

## ****API 명세서****

|  |  |  |  |
| --- | --- | --- | --- |
| **API 이름** | **URI** | **HTTP Method** | **설명** |
| **고객 정보 조회** | /api/v1/customers/{id} | GET | 고객 ID를 기반으로 고객 상세 정보를 조회합니다. |
| **거래 내역 조회** | /api/v1/transactions | GET | 고객의 거래 내역을 조회합니다. |
| **신규 고객 등록** | /api/v1/customers | POST | 신규 고객 정보를 등록합니다. |
| **거래 실행 요청** | /api/v1/transactions | POST | 금융 거래를 실행합니다. |
| **마케팅 캠페인 조회** | /api/v1/marketing/campaigns | GET | 진행 중인 마케팅 캠페인을 조회합니다. |
| **마케팅 캠페인 생성** | /api/v1/marketing/campaigns | POST | 신규 마케팅 캠페인을 생성합니다. |
| **로그인 요청** | /api/v1/auth/login | POST | 사용자 로그인 요청을 처리합니다. |
| **로그아웃 요청** | /api/v1/auth/logout | POST | 사용자 로그아웃 요청을 처리합니다. |
| **보안 로그 조회** | /api/v1/security/logs | GET | 시스템 보안 이벤트 로그를 조회합니다. |

## ****데이터 모델링 정의****

|  |  |  |  |
| --- | --- | --- | --- |
| **테이블명** | **컬럼명** | **데이터 타입** | **설명** |
| **CUSTOMERS** | customer\_id | VARCHAR(50) | 고객 ID (Primary Key) |
|  | name | VARCHAR(100) | 고객 이름 |
|  | email | VARCHAR(100) | 이메일 주소 |
|  | phone\_number | VARCHAR(20) | 전화번호 |
|  | created\_at | TIMESTAMP | 생성일자 |
|  | updated\_at | TIMESTAMP | 수정일자 |
| **TRANSACTIONS** | transaction\_id | VARCHAR(50) | 거래 ID (Primary Key) |
|  | customer\_id | VARCHAR(50) | 고객 ID (Foreign Key) |
|  | amount | DECIMAL(15, 2) | 거래 금액 |
|  | transaction\_date | TIMESTAMP | 거래 일시 |
|  | status | VARCHAR(20) | 거래 상태 (성공, 실패 등) |
| **MARKETING\_CAMPAIGN** | campaign\_id | VARCHAR(50) | 캠페인 ID (Primary Key) |
|  | campaign\_name | VARCHAR(100) | 캠페인명 |
|  | target\_audience | VARCHAR(200) | 타겟 고객 |
|  | execution\_date | TIMESTAMP | 실행일자 |
|  | status | VARCHAR(20) | 상태 (진행 중, 완료 등) |

# ****프로젝트 마무리 보고서****

## ****프로젝트 주요 성과****

NH 금융지주의 통합 데이터 플랫폼 구축 프로젝트는 **디지털 금융 혁신**을 목표로 설계되었습니다.  
본 프로젝트는 **Microservices Architecture**, **Kubernetes 기반 클라우드 네이티브 설계**,  
**데이터 레이크 통합 관리**, **글로벌 확장성 확보**, **보안 및 규제 준수**를 중심으로 성공적으로 완료되었습니다.

### ****주요 성과 정리****

|  |  |
| --- | --- |
| **구분** | **주요 성과** |
| **Microservices 전환** | - Spring Boot를 활용한 MSA 전환으로 서비스 확장성 극대화 |
| **데이터 레이크 통합 관리** | - HDFS와 AWS S3를 통합하여 데이터 분석 및 저장소 일원화 |
| **API Gateway 최적화** | - Spring Cloud Gateway를 통해 트래픽 관리 및 API 보안 강화 |
| **실시간 데이터 분석** | - Kafka Streaming과 Spark Analytics를 통해 실시간 분석 가능 |
| **보안 및 규제 준수 강화** | - 금융보안원(FSI), 전자금융거래법, 개인정보 보호법을 철저히 준수 |
| **클라우드 비용 최적화** | - Kubernetes Auto Scaling 및 S3 Intelligent Tiering 도입으로 비용 절감 |
| **글로벌 서비스 확장성 확보** | - AWS Multi-Region 구성으로 글로벌 금융 서비스 지원 |
| **무중단 배포 및 장애 복구** | - ArgoCD를 활용한 Blue-Green, Canary 배포 전략으로 무중단 서비스 운영 |

## ****프로젝트 목표 달성 분석****

|  |  |  |
| --- | --- | --- |
| **프로젝트 목표** | **달성 여부** | **설명** |
| **디지털 금융 혁신 실현** | ✅ 달성 | - MSA 도입으로 고객 중심의 금융 서비스 실현 - 신속한 거래 처리와 마케팅 캠페인 지원 |
| **실시간 금융 데이터 분석** | ✅ 달성 | - Kafka와 Spark를 활용한 실시간 금융 거래 분석 및 대시보드 제공 |
| **안정적 서비스 운영** | ✅ 달성 | - Kubernetes 기반의 오토스케일링과 무중단 배포 전략을 통해 안정성 확보 |
| **보안 및 규제 준수** | ✅ 달성 | - 금융보안원(FSI) 권고 사항 및 금융 관련 법규 준수로 데이터 기밀성 및 무결성 보장 |
| **글로벌 서비스 확장** | ✅ 달성 | - AWS Multi-Region 아키텍처 구축으로 글로벌 사용자 대응 |
| **운영 비용 최적화** | ✅ 달성 | - 클라우드 리소스 최적화 및 미사용 리소스 제거로 약 30% 비용 절감 |
| **자동화된 배포 및 모니터링** | ✅ 달성 | - CI/CD 파이프라인 구축으로 빠르고 안전한 배포 실현 - Prometheus 모니터링 도입 |

## ****향후 개선 및 확장 계획****

### ****개선 계획****

* **서버리스(Serverless) 도입 검토:**
  + 비핵심 서비스 및 데이터 분석용 단위 작업에 Lambda 도입 검토
* **AI 분석 모델 통합:**
  + Spark MLlib을 활용하여 고객 행동 분석 및 금융 리스크 예측
* **지능형 API Gateway 도입:**
  + AWS API Gateway와 Spring Cloud Gateway의 연동을 통해 라우팅 최적화
* **Zero Trust Architecture 확립:**
  + 내부 네트워크뿐만 아니라 모든 API 요청에 대한 Zero Trust 모델 도입
* **컨테이너 보안 강화:**
  + Docker Bench 및 Trivy를 통한 보안 스캔 주기적 수행

### ****확장 계획****

|  |  |  |
| --- | --- | --- |
| **확장 항목** | **계획 설명** | **완료 목표** |
| **글로벌 서비스 확장** | - AWS Multi-Region을 활용한 글로벌 마케팅 및 금융 거래 지원 | 6개월 내 글로벌 서비스 개시 |
| **멀티 클러스터 오케스트레이션** | - Kubernetes Federation을 통해 다중 클러스터 관리 | 3개월 내 오케스트레이션 완료 |
| **AI 기반 마케팅 분석** | - AI 모델을 통한 고객 세분화 및 마케팅 타겟팅 최적화 | 6개월 내 모델 도입 및 테스트 |
| **모니터링 자동화** | - Grafana 및 Prometheus 연동으로 자동 알림 설정 | 2개월 내 실시간 모니터링 구축 |
| **지능형 보안 탐지 시스템** | - SIEM(Security Information and Event Management) 도입 검토 | 3개월 내 PoC(Proof of Concept) |

## ****마무리 및 결론****

NH 금융지주의 통합 데이터 플랫폼 구축 프로젝트는 **디지털 금융 혁신**과 **고성능 금융 서비스 제공**을 목표로  
성공적으로 수행되었습니다.  
**Microservices Architecture**를 통한 서비스 확장성과 **Kubernetes 기반 클라우드 네이티브 설계**를 통해  
안정적이고 유연한 금융 IT 인프라를 마련하였으며, **데이터 레이크 통합**과 **실시간 분석 플랫폼** 구축으로  
데이터 활용의 극대화를 이루었습니다.

또한, **글로벌 확장성을 확보**하고 **보안 및 규제 준수**를 철저히 지키면서, **비용 최적화**를 통해  
운영 비용을 30% 이상 절감하는 성과를 이뤄냈습니다.

앞으로 AI 분석 모델의 도입과 글로벌 금융 서비스 확장을 통해 NH 금융지주는  
디지털 금융의 선두 주자로 자리 잡을 수 있을 것입니다.