## 📌 0. 도입 전 안내말

## 0.1 프로젝트 개요

NH 금융지주의 통합 데이터 플랫폼 프로젝트는 금융지주 내 NH농협은행, NH농협카드, NH투자증권, NH생명보험, NH손해보험의 **계열사 데이터를 통합**하고,  
실시간 분석 및 개인화 마케팅을 가능하게 하는 **데이터 허브 구축**을 목표로 합니다.

이 프로젝트는 **대용량 데이터의 실시간 수집, 분석, 마케팅 실행**을 최적화하여 금융 서비스의 경쟁력을 강화하고,  
효율적인 데이터 분석 및 비즈니스 전략 수립을 가능하게 합니다.

## 0.2 프로젝트 목적 및 목표

|  |  |
| --- | --- |
| 구분 | 설명 |
| 프로젝트 목적 | - NH 금융지주의 모든 계열사 데이터를 통합하여 분석 및 마케팅에 활용 - 실시간 분석 기반의 캠페인 실행 및 고객 맞춤형 서비스 제공 |
| 프로젝트 목표 | - 1만 TPS 처리 가능한 확장성 확보 - 실시간 분석 환경 구축 (Kafka + Spark Streaming) - AI 기반 예측 분석 및 개인화 마케팅 실행 |
| 핵심 성과 지표(KPI) | - 평균 응답 시간: 3초 이내 - 데이터 수집 및 분석 주기: 1초 이내 실시간 반영 - 캠페인 타겟팅 정확도: 95% 이상 |

## 0.3 주요 시스템 정의

본 프로젝트의 주요 시스템은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 시스템명 | 설명 | 기술 스택 |
| API Gateway | - 외부 트래픽 및 내부 서비스 요청을 관리 - 인증 및 라우팅 처리 | Spring Cloud Gateway, Zuul |
| Microservices | - 고객 정보, 거래 내역, 마케팅 시스템, 캠페인 관리 등 독립적 서비스 | Spring Boot, Kubernetes |
| Data Lake | - 수집된 데이터를 HDFS 및 S3에 중앙 관리 | Hadoop, AWS S3 |
| ETL Processing | - 계열사로부터 배치 데이터 수집 및 클렌징 처리 | IBM DataStage, Control-M |
| Analytics & BI | - Tableau, PowerBI를 통한 실시간 대시보드 및 분석 | Tableau, PowerBI, Spark |
| Security & Access Control | - OAuth2.0 및 JWT 인증 - 전송 구간 암호화 | Spring Security, TLS 1.3 |
| Monitoring & Logging | - 실시간 모니터링 및 로그 분석 | Prometheus, Grafana, ELK Stack |
| Deployment & CI/CD | - 블루-그린 배포 및 무중단 롤링 업데이트 | GitLab CI, ArgoCD, Docker |

## 0.4 아키텍처 설계의 주요 고려 사항

* **확장성 (Scalability):**
  + 금융거래량의 변동에도 대응할 수 있는 **자동 확장(Auto Scaling)** 구조
  + Kubernetes 기반의 Pod 스케일링, HPA(Horizontal Pod Autoscaler) 적용
* **안정성 (Stability):**
  + Active-Standby 구조를 통한 무중단 서비스 운영
  + 장애 발생 시 Prometheus 기반의 Auto-Recovery 구조
* **보안성 (Security):**
  + 전송 구간 암호화(TLS 1.3)
  + OAuth2.0, JWT 인증 기반의 API 보안
* **실시간 분석 (Real-time Analytics):**
  + Kafka Streaming과 Spark Streaming을 통한 실시간 데이터 분석
  + OLAP 및 BI 대시보드에서 1초 단위로 갱신
* **데이터 처리 최적화 (Data Optimization):**
  + IBM DataStage를 통한 배치 처리 최적화
  + Redis를 활용한 캐싱 전략

# 📌 **1. 기술 아키텍처 (TA)**

## **1.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼을 구축하기 위한 기술 아키텍처 설계는 **확장성, 안정성, 보안성, 실시간 분석**을 중점으로 구성됩니다.  
본 아키텍처는 **PaaS 및 IaaS 환경**을 기반으로 클라우드 네이티브 기술을 최대한 활용하며, 무중단 서비스와 빠른 복구를 목표로 설계되었습니다.

## **1.2 인프라 환경 정의**

통합 데이터 플랫폼은 NH농협은행을 중심으로 각 계열사의 금융 데이터를 안정적으로 수집, 저장, 처리, 분석할 수 있는 인프라 구조를 정의합니다.

|  |  |  |
| --- | --- | --- |
| **구성요소** | **설명** | **기술 스택** |
| **Application Layer** | - Spring Boot 기반의 Microservice 구조 - Kubernetes 클러스터 관리 | Spring Boot, Kubernetes |
| **Data Layer** | - Oracle DB, MySQL, HDFS 및 S3에 데이터 저장 - 실시간 조회를 위한 Redis Cache | Oracle DB, MySQL, HDFS, Redis |
| **Integration Layer** | - 외부 서비스와의 통신을 위한 API Gateway와 EAI 구성 | Spring Cloud Gateway, WebSphere MQ |
| **Storage Layer** | - HDFS 및 S3를 통한 대용량 데이터 저장 - IBM DataStage를 활용한 ETL 처리 | HDFS, S3, DataStage |
| **Security Layer** | - OAuth2.0, JWT를 통한 인증 및 인가 - 전송 구간 암호화 (TLS 1.3) | OAuth2.0, JWT, TLS 1.3 |
| **Monitoring Layer** | - Prometheus와 Grafana를 활용한 실시간 모니터링 | Prometheus, Grafana |
| **Logging Layer** | - ELK Stack을 활용한 통합 로그 관리 | Elasticsearch, Logstash, Kibana |

## **1.3 HW, SW, 인력 계획**

|  |  |  |
| --- | --- | --- |
| **항목** | **내용** | **비고** |
| **서버 (VM)** | - Application Server: 10대 - Database Server: 5대 - ETL Server: 3대 | PaaS 기반 클러스터 구성 |
| **스토리지 (Storage)** | - HDFS: 50TB - AWS S3: 100TB | 이중화 구성 |
| **네트워크 (Network)** | - 10Gbps 전용선 사용 - DR센터와 이중화 네트워크 구성 | MPLS 망 활용 |
| **보안 솔루션** | - WAF, IPS, IDS, 방화벽 설정 | 3중 보안 계층 적용 |
| **인력 계획** | - 개발자: 10명 - 아키텍트: 5명 - 인프라 엔지니어: 5명 | 전문 분야별 역할 분리 |
| **개발 및 운영 툴** | - GitLab CI, Jenkins, Docker, Kubernetes | DevOps 자동화 관리 |

## **1.4 예산 계획**

**예산 산정 기준:**

* **인프라 구축 비용:** VM 및 스토리지 구축, 네트워크 장비 구매
* **소프트웨어 라이선스:** Oracle DB, DataStage, WebSphere MQ
* **클라우드 사용료:** AWS S3, RDS, VPC 등

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **항목** | **예산 (억원)** | **비고** |
| **인프라 구축 비용** | 서버, 네트워크, 보안 장비 | 15 | 이중화 구성 기준 |
| **SW 라이선스 비용** | Oracle, DataStage | 8 | 연간 라이선스 비용 |
| **클라우드 사용료** | AWS S3, RDS, VPC | 10 | 5년 기준 예상 |
| **인건비** | 개발자, 아키텍트, 엔지니어 | 12 | 프로젝트 2년간 |
| **기타** | 교육 및 유지보수 | 2 |  |
| **총 예산** |  | **47** |  |

## **1.5 스토리지 및 임시 보관 영역 설계**

NH 금융지주의 통합 데이터 플랫폼은 실시간 수집 및 분석을 위한 **스토리지 레이어**를 설계하였습니다.

* **HDFS (Hadoop Distributed File System):**
  + 일괄 배치 수집된 데이터를 분산 처리
  + DataStage를 통해 클렌징 후 저장
* **AWS S3:**
  + 장기 보관 및 백업 목적으로 사용
  + 실시간 분석 시 데이터를 Spark와 연계
* **Redis Cache:**
  + 조회 빈도가 높은 데이터는 Redis에 캐싱하여 조회 성능 개선
  + 예시: 고객 정보, 금융 상품 정보
* **임시 보관 영역:**
  + 배치 수집 중인 데이터가 실시간 분석 전 저장되는 공간
  + 데이터 전처리 후 HDFS로 전송

## **1.6 설계 및 구현 방안**

### 🔍 ****1. 무중단 운영 구조****

* Kubernetes를 통한 **Auto Healing** 및 **Auto Scaling** 지원
* 장애 발생 시 Active-Standby 전환

### 🔍 ****2. 네트워크 최적화****

* **10Gbps 전용선**을 통한 빠른 데이터 전송
* MPLS 망을 통해 데이터센터 간 통신 최적화

### 🔍 ****3. 데이터 중복 제거 및 최적화****

* HDFS에 저장 시 중복 데이터 필터링
* Redis Cache를 통해 반복 조회 최적화

### 🔍 ****4. DR (Disaster Recovery) 전략****

* AWS S3에 매일 증분 백업
* 매주 전체 백업을 통해 DR 센터에 이중화

# 📌 **2. 애플리케이션 아키텍처 (AA)**

## **2.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼의 애플리케이션 아키텍처는 **Microservice 기반**으로 설계되어, 각 계열사의 데이터를 실시간으로 수집하고,  
분석하며, 개인화된 마케팅 캠페인을 실행할 수 있는 구조를 지원합니다.  
주요 프로세스는 **데이터 수집 → 처리 → 저장 → 분석 → 활용**으로 구성되며, 각각 독립적인 서비스로 설계되어 확장성과 유지보수성이 뛰어납니다.

## **2.2 시스템 연계 및 수집 프로세스**

NH 금융지주의 계열사 및 외부 시스템과의 데이터 수집 및 연계 프로세스는 다음과 같이 정의됩니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **계열사 데이터 수집** | - NH농협은행, NH농협카드, NH투자증권, NH생명보험, NH손해보험의 배치 및 실시간 수집 | IBM DataStage, Control-M |
| **올원뱅크 행동 데이터** | - 올원뱅크 사용자 행동 데이터를 실시간으로 수집 | Kafka, Spark Streaming |
| **외부 연계 인터페이스** | - 공공기관 및 외부 금융사와의 실시간 데이터 연계 | FEP, EZGATOR, MCA |
| **암호화 전송 및 복호화** | - 전송 구간에 대해 TLS 1.3을 적용하여 암호화된 전송 수행 | TLS 1.3, WebSphere MQ |
| **파일 전송 및 처리** | - MFT(Managed File Transfer) 방식으로 파일을 전송 | EZGATOR, FOS MFT |
| **로그 관리 및 감사 기록** | - ELK Stack을 통해 전송 내역 및 처리 이력 실시간 모니터링 | Elasticsearch, Kibana |

## **2.3 설계 및 구현 방안**

### 🔍 ****1. 계열사 데이터 수집 설계****

* **Batch 수집 프로세스:**
  + IBM DataStage를 통해 일괄 배치 처리
  + Control-M을 활용하여 스케줄링 및 오류 관리
  + 배치 완료 후 HDFS에 적재
* **실시간 수집 프로세스:**
  + Kafka를 활용하여 실시간 메시지 큐 전송
  + Spark Streaming이 데이터를 분석하고, 실시간 분석 결과를 HDFS에 저장

### 🔍 ****2. 올원뱅크 행동 데이터 수집 설계****

* **Kafka Topic 기반 실시간 전송:**
  + 사용자 로그인, 거래, 화면 전환 시 이벤트 발생
  + 이벤트는 Kafka Topic으로 스트림 처리
* **Spark Streaming 분석:**
  + 1초 단위의 마이크로 배치를 통해 실시간 분석
  + 고객의 금융 패턴 분석, 이벤트 기반 마케팅 실행

### 🔍 ****3. 외부 연계 인터페이스 설계****

* **FEP 및 MCA 표준 인터페이스 적용:**
  + 금융권 표준 프로토콜을 준수하여 전송
  + WebSphere MQ를 통해 메시지 보관 및 복원 가능
* **EZGATOR 연계:**
  + 외부 금융기관 및 공공기관과의 데이터 전송 시 MFT 방식 사용
  + 전송 구간 암호화 및 복호화 처리

### 🔍 ****4. 데이터 전송 및 암호화 설계****

* **TLS 1.3 전송 보안:**
  + 모든 HTTP 요청은 HTTPS를 통해 암호화 처리
  + API Gateway에서 OAuth2.0 및 JWT를 통해 인증
* **전송 시 암호화 처리:**
  + 전송 전 AES-256으로 파일 암호화
  + 전송 중 TLS 1.3 채널을 통해 안전한 전송
  + 수신 후 RSA 공개키를 통해 복호화

### 🔍 ****5. 통합고객번호 채번 및 관리 설계****

* **GUID (Global Unique Identifier) 방식 채번:**
  + 모든 고객 정보는 통합된 GUID로 관리
  + 중복 방지를 위한 Redis Lock 활용
* **Redis Cache 적용:**
  + 고객 정보 조회 시 Redis Cache에 적재하여 조회 속도 최적화
  + TTL(Time To Live) 설정을 통해 만료 관리

### 🔍 ****6. 배치 로깅 및 실시간 연계 서비스****

* **배치 로그 관리:**
  + Control-M에서 배치 완료 시 ELK Stack에 로그 전송
  + 성공/실패 여부 및 처리 시간 모니터링
* **실시간 연계 서비스:**
  + FEP와 MCA를 통해 실시간 데이터 송수신
  + Prometheus와 Grafana를 통해 실시간 상태 모니터링

### 🔍 ****7. 금융지주 OLAP 환경 설계****

* **OLAP 플랫폼 구성:**
  + Tableau와 PowerBI를 활용하여 다차원 분석
  + 실시간 대시보드 제공: 예적금 현황, 카드 거래 분석, 고객 분류
* **대시보드 예시:**
  + **실시간 거래 분석:** 특정 시간대 거래량 분석
  + **고객 세분화 분석:** 연령별, 지역별 금융 상품 선호도 분석
  + **마케팅 캠페인 성과 분석:** 시나리오별 고객 반응 확인

### 🔍 ****8. 설계 다이어그램 예시****

* **연계 및 수집 아키텍처 다이어그램:**
  + 계열사 → IBM DataStage → ETL 처리 → HDFS
  + 올원뱅크 → Kafka → Spark Streaming → Data Lake
  + 외부 금융사 → EZGATOR → FEP → 통합 데이터 플랫폼

### 🔍 ****9. 테스트 및 검증 방안****

* **배치 처리 테스트:**
  + Control-M의 스케줄링과 DataStage 처리 시간 검증
  + 처리 완료 시 ELK Stack에 로그 저장
* **실시간 처리 테스트:**
  + Kafka 스트림 전송 후 1초 내 분석 결과 생성 확인
  + Spark Streaming의 분석 정확도 테스트
* **암호화 전송 검증:**
  + TLS 1.3 채널을 통한 안전한 전송 여부 체크
  + AES-256 암호화된 파일이 정상 복호화되는지 테스트

# 📌 **3. 비즈니스 아키텍처 (BA)**

## **3.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서 \*\*비즈니스 아키텍처(BA)\*\*는 개인화 마케팅, 데이터 분석, 고객 행동 분석 등  
실질적인 비즈니스 가치를 창출하는 핵심 영역입니다.  
개인화 마케팅을 통해 고객에게 맞춤형 금융 상품을 추천하고, 실시간 분석을 통해 고객의 행동 패턴을 예측하여  
최적의 마케팅 시나리오를 실행합니다.

## **3.2 개인화 마케팅 서비스 설계**

개인화 마케팅 서비스는 고객의 행동 데이터를 실시간으로 수집하고 분석하여,  
**타겟 마케팅, 우선순위 설정, 피로도 정책 관리, 실시간 캠페인 실행**을 지원합니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **타겟 마케팅** | - 고객의 거래 내역, 금융 상품 이용 정보를 기반으로 맞춤형 상품 추천 | Spark ML, Kafka Streaming |
| **우선순위 설정** | - 고객의 반응 정보를 분석하여 마케팅 우선순위 설정 | Redis, Spring Boot |
| **피로도 정책 관리** | - 반복 노출 방지를 위해 피로도 정책 관리 | Spring Scheduler, Redis Cache |
| **실시간 캠페인 실행** | - 금융 거래 시 실시간 분석 후 캠페인 실행 | Kafka, Spring Cloud Stream |
| **성과 분석 및 리포팅** | - 모든 마케팅 캠페인의 성과를 실시간으로 분석하고 리포팅 | Tableau, PowerBI |

## **3.3 설계 및 구현 방안**

### 🔍 ****1. 타겟 마케팅 설계****

* **Spark ML 기반 고객 분석:**
  + 고객의 금융 거래 내역을 분석하여 **고위험군, 중위험군, 저위험군**으로 분류
  + 예적금 가입 패턴, 카드 사용 내역, 대출 상환 이력 등을 학습
* **Kafka를 통한 실시간 이벤트 분석:**
  + 고객의 거래 발생 시 Kafka Topic으로 실시간 전송
  + Spark Streaming이 이벤트를 분석하고 결과를 Redis Cache에 반영
* **실시간 추천 예시:**
  + 대출 연체 고객 → "대출 만기 알림" 메시지 전송
  + 예적금 만기 고객 → "재가입 혜택 캠페인" 메시지 실행

## 🔍 **2. 우선순위 설정 설계**

* **Redis를 통한 실시간 우선순위 관리:**
  + 고객의 최근 거래 내역 및 행동 패턴을 Redis에 캐싱
  + 실시간 분석을 통해 마케팅 노출의 우선순위를 설정
* **Spring Scheduler를 통한 관리:**
  + 하루 2회 우선순위 업데이트 (00:00, 12:00)
  + 캠페인 실행 시 우선순위가 높은 고객에게 먼저 전송

## 🔍 **3. 피로도 정책 관리 설계**

* **피로도 정책 정의:**
  + 동일한 마케팅 메시지가 특정 기간 내 반복 노출되지 않도록 설정
  + 고객 반응에 따라 **쿨타임 (Cool Time)** 적용
* **Redis TTL 설정:**
  + 특정 고객에게 메시지 전송 시 TTL(Time To Live)을 설정
  + 예를 들어, '신용대출 추천' 메시지는 7일 이내에 동일 고객에게 중복 노출되지 않음

## 🔍 **4. 실시간 캠페인 실행 설계**

* **Kafka 기반 실시간 캠페인 트리거:**
  + 고객의 금융 거래가 발생하면 Kafka Topic에 이벤트 생성
  + Spring Cloud Stream을 통해 실시간 캠페인 실행
* **캠페인 실행 예시:**
  + 카드 결제 시 → "결제 완료 알림 및 포인트 적립 안내"
  + 대출 신청 시 → "신용평가 완료 및 금리 조건 안내"

## 🔍 **5. 성과 분석 및 리포팅 설계**

* **Tableau와 PowerBI 활용:**
  + 실시간 캠페인 결과를 분석하고 대시보드에 시각화
  + 고객 반응 분석 (클릭율, 전환율) 및 ROI 분석
* **주요 리포트 예시:**
  + **마케팅 캠페인 리포트:** 실행 횟수, 전환율, 고객 응답률 분석
  + **세그먼트 분석:** 연령별, 지역별, 상품별 분석
  + **이탈율 분석:** 특정 고객군의 서비스 이탈율

## 🔍 **6. 설계 다이어그램 예시**

* **개인화 마케팅 아키텍처 다이어그램:**
  + 고객 이벤트 → Kafka → Spark Streaming → Redis Cache
  + Redis → Spring Boot → 캠페인 실행
  + 캠페인 결과 → PowerBI/Tableau → 실시간 대시보드

## 🔍 **7. 테스트 및 검증 방안**

* **타겟 마케팅 테스트:**
  + 고객 정보 변경 시 실시간 반영되는지 검증
  + Kafka 메시지 전송 후 Spark에서 1초 이내 처리 확인
* **우선순위 설정 테스트:**
  + Redis에 적재된 우선순위가 올바르게 반영되는지 확인
  + TTL 설정에 따라 중복 노출이 제한되는지 검증
* **피로도 정책 검증:**
  + 동일 고객에게 7일 이내 동일 메시지가 발송되지 않는지 확인
* **실시간 캠페인 실행 검증:**
  + 금융 거래 이벤트 발생 시 실시간으로 실행되는지 확인
  + Slack 및 Kibana를 통해 로그 확인

# 📌 **4. 데이터 아키텍처 (DA)**

## **4.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서 데이터 아키텍처는 **데이터의 수집, 저장, 처리, 분석**을 효과적으로 관리하기 위한 핵심 구조입니다.  
계열사 데이터, 고객 거래 데이터, 금융 상품 데이터 등을 체계적으로 관리하고, **OLAP 분석, 머신러닝 학습, 실시간 마케팅**에 필요한 데이터를 최적화합니다.

## **4.2 데이터 주제 영역 정의**

통합된 데이터 플랫폼에서 관리할 주요 주제영역을 정의합니다.

|  |  |  |
| --- | --- | --- |
| **주제영역** | **세부 항목** | **설명** |
| **고객정보** | 고객기본정보, 고객실적정보, 고객관리정보 | 개인 및 법인 고객의 금융 거래 및 관리 정보 |
| **상품정보** | 상품기본정보, 상품분류정보 | 금융 상품의 세부 정보 (대출, 예적금, 카드, 보험 등) |
| **거래정보** | 수신거래, 여신거래, 카드거래, 외환거래 | 금융 거래 내역 및 결제 정보 |
| **채널정보** | 가입채널정보, 채널이용정보, 상담정보 | 고객이 접근한 금융 채널 (모바일, 인터넷뱅킹, ATM) |
| **공통정보** | 공통코드, 사무소/가맹점정보, 기타 공통정보 | 모든 계열사에서 공통적으로 사용하는 참조 정보 |
| **분석정보** | OLAP 분석, 마케팅 성과 분석 | BI 및 Big Square를 통한 통계 및 예측 분석 |

## **4.3 원천 데이터 적재 및 목적별 마트 구성**

### 🔍 ****1. 원천 데이터 적재 구조****

* **Data Lake**
  + 모든 금융 거래 및 고객 정보는 실시간으로 Data Lake에 적재
  + HDFS를 통해 분산 저장 및 처리
  + 비정형 데이터와 정형 데이터를 동시에 처리
* **ETL 처리:**
  + IBM DataStage를 활용하여 배치 처리된 데이터를 정제 후 적재
  + Spark Streaming을 통해 실시간 스트림 처리

### 🔍 ****2. 목적별 데이터 마트 구성****

* **정제/통합 레이어 (Refined & Integrated Layer):**
  + 원본 데이터를 통합하고 중복 제거
  + 통합 고객번호 기준으로 조회 최적화
* **요약/분석 레이어 (Summary & Analytics Layer):**
  + 마케팅 분석, 상품 분석, 금융 리스크 평가를 위한 요약 정보 제공
  + Tableau와 PowerBI 대시보드와 실시간 연동

## **4.4 메타 데이터 관리 및 조회**

NH 금융지주는 금융 데이터의 표준화를 위해 **메타 데이터 관리 체계**를 도입합니다.  
모든 데이터는 일관된 정의를 가지고 저장되며, 메타 데이터는 중앙에서 관리되고 조회 가능합니다.

|  |  |  |
| --- | --- | --- |
| **항목** | **설명** | **기술 스택** |
| **메타 데이터 관리** | - 데이터의 정의, 유형, 출처, 관계 정보를 관리 | Apache Atlas |
| **데이터 사전** | - 금융 데이터의 속성 및 구조를 문서화하고 조회 가능 | Data Dictionary |
| **버전 관리** | - 데이터 변경 시 버전 관리 및 추적 가능 | GitLab, Nexus |
| **접근 제어** | - 메타 데이터 조회 권한 관리 및 Role-Based Access Control | Apache Ranger |

## **4.5 분석용 목적 DB 구성**

* **OLAP DB (Online Analytical Processing):**
  + Tableau와 PowerBI 대시보드의 실시간 분석을 위한 다차원 큐브 구성
  + 주요 분석 항목: 고객 세분화, 상품 매출 분석, 금융 리스크 평가
* **Spark SQL & DataFrame:**
  + 실시간으로 수집된 스트림 데이터를 분석
  + 빅데이터 분석 시 빠른 조회와 병렬 처리 지원
* **ML/DL 모델 데이터 구성:**
  + 분석용 목적 DB에 모델 학습 데이터를 저장
  + TensorFlow 및 PyTorch 모델 학습 시 활용

## **4.6 데이터 보관 주기 및 파기 정책**

NH 금융지주는 개인정보 보호법 및 금융 관련 법규에 따라 데이터의 보관 및 파기 정책을 준수합니다.

|  |  |  |
| --- | --- | --- |
| **항목** | **설명** | **기술 스택** |
| **데이터 보관 주기** | - 고객 정보: 5년 보관 - 금융 거래 정보: 10년 보관 | DataStage, Oracle DB |
| **파기 대상 데이터 정의** | - 금융 상품 만기, 거래 종료 후 5년이 경과한 데이터 | Control-M Scheduler |
| **파기 절차 정의** | - 데이터 파기 전 사전 검토 및 암호화 해제 - 파기 기록 로그 관리 | ELK Stack |
| **데이터 백업 정책** | - AWS S3에 주기적 증분 백업 및 주간 전체 백업 실행 | AWS S3, DataStage |

#### 🔍 ****7. 설계 다이어그램 예시****

* **데이터 아키텍처 다이어그램:**
  + 계열사 → ETL (DataStage) → Data Lake (HDFS, S3) → 분석 마트
  + 실시간 이벤트 → Kafka → Spark Streaming → 분석 DB

#### 🔍 ****8. 테스트 및 검증 방안****

* **Data Lake 적재 테스트:**
  + 계열사로부터 수집된 데이터가 HDFS에 올바르게 적재되는지 확인
  + DataStage의 배치 처리 시 오류 로그 탐색
* **데이터 마트 조회 테스트:**
  + Tableau와 PowerBI의 실시간 조회 속도 측정
  + SQL 조회 시간 1초 이내 유지 여부 검증
* **파기 정책 테스트:**
  + 만료된 데이터가 암호화 해제 후 올바르게 삭제되는지 확인
  + ELK Stack을 통해 로그 추적

# 📌 **5. 보안 아키텍처 (SA)**

## **5.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서는 **금융 데이터의 보안성**을 최우선으로 고려하여 설계되었습니다.  
모든 금융 거래 및 고객 정보는 **전송 구간 암호화, 저장 시 암호화, 접근 제어**를 통해 보호되며,  
내부 관리자는 **Role-Based Access Control (RBAC)** 체계를 통해 접근 권한을 관리합니다.  
또한, **제3자 동의 기반 통합 인증 체계**를 통해 개인정보의 안전한 활용을 보장합니다.

## **5.2 제3자 동의 기반 통합 데이터 인증**

NH 금융지주의 각 계열사에서 수집되는 고객 데이터는 **제3자 동의**를 기반으로 통합 관리됩니다.  
모든 개인정보는 고객의 동의 하에 수집되며, 수집된 데이터는 암호화되어 전송 및 저장됩니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **통합 인증 체계** | - OAuth2.0 및 OpenID Connect 기반 인증 | Keycloak, Spring Security |
| **SSO (Single Sign-On)** | - 통합 인증을 통해 각 계열사의 금융 서비스에 단일 로그인 제공 | Keycloak, OpenAM |
| **동의서 관리** | - 고객 동의서 버전 관리 및 동의 내역 로그 기록 | ELK Stack, GitOps |
| **접근 통제** | - Role-Based Access Control (RBAC) 적용 | Spring Security, Apache Ranger |

## **5.3 전송구간 암호화 설계**

NH 금융지주의 모든 금융 데이터는 전송 시 **TLS 1.3** 암호화 프로토콜을 적용하여 보호됩니다.  
외부 기관 및 계열사 간 데이터 전송 시에도 **FEP 및 MCA** 표준 인터페이스를 통해 안전하게 처리됩니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **TLS 1.3 암호화** | - 모든 전송 구간 암호화 처리 - API Gateway를 통한 HTTPS 통신 적용 | Spring Cloud Gateway, Zuul |
| **FEP (Front-End Processor)** | - 외부 금융 기관과의 표준 인터페이스 처리 | WebSphere MQ, MCA |
| **전송파일 암호화** | - 민감 정보 포함 파일은 전송 시 AES-256 암호화 | OpenSSL, DataStage |

## **5.4 전송파일 암호화 설계**

* **AES-256 알고리즘을 활용한 파일 암호화:**
  + 계열사에서 전송된 금융 데이터를 수신하기 전 AES-256으로 암호화
  + 송신 시점에 인증된 키를 통해 암호화 처리
  + 수신 시점에 TLS 1.3으로 보호된 환경에서 복호화
* **PGP 암호화 지원:**
  + 외부 기관과의 데이터 전송 시 PGP (Pretty Good Privacy) 암호화 적용
  + 전자서명 검증 후 파일 전송

## **5.5 개인정보 처리 시 암호화/마스킹/파기 설계**

* **암호화 처리:**
  + 모든 개인정보는 저장 시 AES-256으로 암호화하여 DB에 저장
  + 예: 주민등록번호, 계좌번호, 카드번호 등
* **마스킹 처리:**
  + 화면 조회 시 중요 정보는 마스킹 처리 (●●●-●●●●-●●●●)
  + 조회 권한에 따라 마스킹 여부가 결정됨
* **파기 프로세스:**
  + 개인정보의 보관 기한이 만료되면 **DataStage**를 통해 자동 삭제
  + 파기 시 로그 기록이 남으며, ELK Stack을 통해 이력 관리

## **5.6 계정 및 권한 관리 체계**

* **RBAC (Role-Based Access Control) 적용:**
  + 각 사용자 및 관리자는 역할(Role) 기반으로 접근이 제한됩니다.
  + 예를 들어, 금융 상품 조회 권한과 고객 정보 수정 권한은 별도로 분리됨.
* **정책 관리:**
  + Apache Ranger를 활용하여 정책 관리
  + 실시간 로그 기록 및 접근 내역 추적

| **구성요소** | **설명** | **기술 스택** |
| --- | --- | --- |
| **인증 서버** | - Keycloak을 통한 OAuth2.0 인증 | Keycloak, Spring Security |
| **접근 정책 관리** | - Apache Ranger를 활용한 역할 기반 접근 제어 | Apache Ranger |
| **로그 관리** | - 모든 접근 기록은 ELK Stack에 저장 | Elasticsearch, Kibana |
| **정책 실행 엔진** | - Policy Engine을 통한 실시간 정책 적용 | Apache Ranger |

#### 🔍 ****7. 설계 다이어그램 예시****

* **보안 아키텍처 다이어그램:**
  + 외부 → API Gateway → OAuth2.0 인증 → FEP → AES-256 암호화
  + 내부 → Spring Boot Microservices → 데이터 접근 시 RBAC → Apache Ranger 정책 검증

#### 🔍 ****8. 테스트 및 검증 방안****

* **암호화 전송 테스트:**
  + TLS 1.3 암호화 전송 시 Wireshark로 패킷 분석하여 복호화 불가능 여부 확인
  + AES-256 암호화된 파일의 복호화 테스트
* **마스킹 처리 검증:**
  + 조회 권한이 없는 사용자는 마스킹된 상태로 조회되는지 확인
  + 관리자 권한 사용자는 복호화된 상태로 조회되는지 확인
* **접근 제어 검증:**
  + Apache Ranger의 정책이 올바르게 실행되는지 검증
  + 접근 로그가 ELK Stack에 정확히 적재되는지 확인
* **파기 테스트:**
  + 만료된 개인정보가 DataStage에 의해 정상 파기되는지 검증
  + ELK Stack을 통해 파기 이력 조회

# 📌 **6. 인터페이스 아키텍처 (IA)**

## **6.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서 \*\*인터페이스 아키텍처(IA)\*\*는 다양한 계열사 및 외부 금융 기관과의 **데이터 연계 및 통신 표준화**를 목표로 합니다.  
모든 인터페이스는 \*\*표준화된 프로토콜(MQ, REST, gRPC)\*\*을 따르며, 실시간 데이터 처리와 대량 데이터 전송이 가능한 구조로 설계되었습니다.  
또한, **전송 구간 암호화** 및 \*\*표준 인터페이스(FEP, MCA)\*\*를 통해 안전한 데이터 전송을 보장합니다.

## **6.2 계열사 데이터 수집 인터페이스 설계**

각 계열사의 데이터 수집은 **배치(Batch)와 실시간(Real-time)** 두 가지 방식으로 수행됩니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **배치 수집** | - 매일 정해진 시간에 각 계열사에서 배치 데이터를 수집하여 ETL 처리 후 적재 | IBM DataStage, Control-M |
| **실시간 수집** | - 고객의 금융 거래 발생 시 실시간으로 수집하여 HDFS에 저장 | Kafka, Spark Streaming |
| **표준 인터페이스 적용** | - NH 금융지주 내 표준 프로토콜인 FEP 및 MCA를 활용한 통신 처리 | WebSphere MQ, EZGATOR |
| **암호화 및 전송 관리** | - TLS 1.3을 통한 암호화 전송 - MFT를 통한 파일 전송 보안 강화 | TLS 1.3, FOS MFT |

### ****6.3 올원뱅크 행동 데이터 수집 설계****

* **Kafka 기반 실시간 스트림 처리:**
  + 올원뱅크에서 발생하는 고객 행동 데이터를 실시간으로 Kafka에 전송
  + 주요 데이터: 로그인, 결제, 조회, 송금 이벤트
* **Spark Streaming 분석:**
  + Kafka로부터 수신된 데이터를 \*\*1초 단위 마이크로 배치(Micro Batch)\*\*로 분석
  + 고객의 행동 패턴 분석을 통한 실시간 마케팅 트리거

|  |  |  |  |
| --- | --- | --- | --- |
| **행동 이벤트** | **수집 주기** | **전송 방식** | **적재 위치** |
| 로그인 기록 | 실시간 | Kafka Topic | HDFS, Data Lake |
| 결제 및 송금 내역 | 실시간 | Kafka Topic | HDFS, Data Lake |
| 금융 상품 조회 | 실시간 | Kafka Topic | Redis Cache |
| 마케팅 응답 | 실시간 | Kafka Topic | HDFS, Analytics DB |

## **6.4 데이터 연계 표준 및 프로토콜 설계**

NH 금융지주는 **표준화된 인터페이스 프로토콜**을 통해 외부 금융 기관 및 내부 시스템 간 연계합니다.

|  |  |  |
| --- | --- | --- |
| **프로토콜** | **설명** | **기술 스택** |
| **FEP (Front-End Processor)** | - 외부 금융 기관과의 실시간 데이터 연계 처리 | WebSphere MQ, EZGATOR |
| **MCA (Message Control Adapter)** | - 계열사 간 메시지 통신 표준화 및 관리 | MCA Adapter |
| **REST API** | - 외부 채널(모바일, 인터넷뱅킹)과의 RESTful 인터페이스 | Spring Cloud Gateway |
| **gRPC** | - 내부 마이크로서비스 간의 초고속 통신 처리 | gRPC, Protobuf |
| **MFT (Managed File Transfer)** | - 대량의 파일을 안전하게 전송하며 보안 암호화 처리 | FOS MFT, SFTP |

#### 🔍 ****1. FEP 및 MCA 설계****

* **FEP (Front-End Processor):**
  + 외부 금융기관(예: 증권사, 보험사)과의 실시간 메시지 전송
  + 금융 거래 시 MQ를 통해 전송 및 응답 처리
* **MCA (Message Control Adapter):**
  + 내부 시스템 간 대량 메시지 처리
  + 비동기 메시지 전송을 통해 트랜잭션 무결성 유지

#### 🔍 ****2. REST API 및 gRPC 설계****

* **REST API 구조:**
  + 외부 채널에서 고객 정보를 조회하거나 금융 거래를 수행할 때 사용
  + 예:
    - /api/v1/accounts/{accountId} → 계좌 조회
    - /api/v1/transactions/{transactionId} → 거래 내역 조회
* **gRPC 구조:**
  + 내부 마이크로서비스 간 초고속 통신
  + Protobuf로 직렬화된 데이터 전송 → 네트워크 지연 최소화

#### 🔍 ****3. MFT 설계****

* **FOS MFT (Managed File Transfer):**
  + 대량 금융 데이터 및 배치 파일 전송 시 사용
  + AES-256 암호화 후 전송 → 전송 중 데이터 유출 방지
* **SFTP 보안 전송:**
  + 외부 기관과의 데이터 연동 시 사용
  + 전송 시 TLS 1.3을 통한 암호화 적용

## **6.5 분석 플랫폼과의 연계 설계**

* **Big Square 및 BI 포털 연계:**
  + 실시간 분석 데이터는 Spark Streaming을 통해 Big Square로 전달
  + 분석 결과는 Tableau, PowerBI에서 시각화
* **연계 방식:**
  + Kafka → Spark Streaming → Data Lake (HDFS, S3) → BI 포털

#### 🔍 ****7. 설계 다이어그램 예시****

* **인터페이스 아키텍처 다이어그램:**
  + 계열사 → DataStage (ETL) → Data Lake (HDFS)
  + 올원뱅크 → Kafka → Spark Streaming → Data Lake
  + 외부 금융기관 → FEP → MFT → 통합 데이터 플랫폼

#### 🔍 ****8. 테스트 및 검증 방안****

* **FEP 및 MCA 테스트:**
  + 금융 거래 시 메시지 손실 없이 전달되는지 검증
  + 비정상 메시지 처리 시 오류 로그 생성 여부 확인
* **REST API 및 gRPC 테스트:**
  + HTTP 요청 시 2초 이내 응답 확인
  + gRPC 요청 시 평균 50ms 이하 응답 시간 유지
* **MFT 전송 테스트:**
  + 대용량 파일 전송 시 데이터 손실 없는지 확인
  + 전송 완료 시 SFTP 로그 검토

# 📌 **7. 기대효과 및 전략적 가치**

## **7.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼 구축을 통해 데이터 중심의 경영을 실현하고,  
**실시간 분석, 예측 모델, 개인화 마케팅**을 기반으로 금융 서비스의 경쟁력을 극대화합니다.  
통합된 금융 데이터를 기반으로 고객의 금융 행동을 분석하고, 이를 통해 **맞춤형 금융 서비스**를 제공할 수 있습니다.  
또한, **운영 비용 절감, 의사결정 최적화, 마케팅 효과 극대화**를 기대할 수 있습니다.

## **7.2 기대효과 분석**

| **구분** | **설명** | **주요 기술 스택** |
| --- | --- | --- |
| **비용 절감 효과** | - IT 인프라 통합 및 최적화를 통해 운영 비용 절감 - PaaS/IaaS 클라우드 기반 비용 최적화 | Kubernetes, AWS S3, HDFS |
| **데이터 분석 최적화** | - 실시간 데이터 분석 및 AI 모델 학습 최적화 - OLAP 분석을 통한 비즈니스 인사이트 도출 | Spark Streaming, Tableau |
| **의사결정 지원** | - 통합된 데이터 분석 결과를 통해 경영진의 신속한 의사결정 지원 - 실시간 대시보드로 KPI 관리 | PowerBI, Grafana |
| **마케팅 효과 극대화** | - 개인화 마케팅을 통한 전환율 상승 - 타겟 마케팅을 통해 고객의 재방문율 증가 | Redis Cache, Spring Boot |
| **보안 및 규제 준수** | - 전송 구간 및 저장 시 데이터 암호화 - GDPR 및 개인정보 보호법 준수 | TLS 1.3, AES-256, Apache Ranger |

## **7.3 전략적 가치 분석**

#### 🔍 ****1. 금융 데이터의 통합 관리****

* 모든 계열사 데이터를 통합하여 관리하므로, **이중 관리 비용 절감**
* 중앙 집중식 관리로 인해 데이터 조회 및 분석 속도 최적화

#### 🔍 ****2. 실시간 분석 및 예측 능력 향상****

* Kafka와 Spark Streaming을 통해 실시간으로 금융 데이터를 분석
* 고객 행동 패턴을 예측하여 최적의 마케팅 시점에 캠페인 실행

#### 🔍 ****3. 고객 맞춤형 금융 서비스 제공****

* **마케팅 캠페인 최적화:** 고객의 금융 패턴을 분석하여 맞춤형 금융 상품 제공
* **리스크 관리 최적화:** 실시간 분석을 통해 금융 리스크 최소화

#### 🔍 ****4. 무중단 서비스와 장애 대응력 강화****

* Kubernetes 기반의 Auto Healing 및 Auto Scaling 적용
* 장애 발생 시 자동 복구 및 무중단 서비스 유지

#### 🔍 ****5. 규제 준수와 보안 강화****

* GDPR, ISMS, 개인정보 보호법에 맞춘 암호화 및 접근 제어
* TLS 1.3, AES-256을 통한 보안 강화

## **7.4 설계 및 구현 방안**

#### 🔍 ****1. PaaS/IaaS 기반의 비용 절감 전략****

* Kubernetes와 AWS S3를 통한 인프라 최적화
* IaaS를 통한 서버 확장 및 축소가 자동화
* 중복 저장소를 제거하고 HDFS 및 S3에 통합

#### 🔍 ****2. 실시간 분석 및 예측 모델 설계****

* **Kafka Streaming:**
  + 실시간 이벤트 처리, 거래 내역 분석, 고객 반응 분석
* **Spark ML 모델 학습:**
  + 고객 행동 데이터를 기반으로 모델 학습
  + 마케팅 캠페인에 최적화된 예측 모델 생성

#### 🔍 ****3. 무중단 서비스 및 DR 전략****

* **Auto Healing:**
  + Pod 장애 시 자동 재시작 및 복구
* **Disaster Recovery (DR):**
  + AWS S3에 실시간 복제 및 백업
  + 장애 발생 시 DR 센터로 15분 이내 전환

#### 🔍 ****4. 보안 및 규제 준수 설계****

* **전송 구간 암호화:**
  + TLS 1.3을 통한 데이터 전송 보안
  + FEP 및 MCA 표준 인터페이스를 활용
* **데이터 접근 제어:**
  + Apache Ranger를 통한 Role-Based Access Control (RBAC)
  + ELK Stack을 통한 모든 접근 기록 로그 관리

#### 🔍 ****5. 데이터 관리 최적화****

* **Data Lake 최적화:**
  + 중복 데이터 제거, 압축 처리로 저장 공간 최적화
  + DataStage를 통한 정제 및 통합
* **Analytics DB 최적화:**
  + OLAP 분석 시 실시간 조회 최적화
  + BI 포털을 통해 실시간 시각화

#### 🔍 ****6. 설계 다이어그램 예시****

* **전략적 가치 아키텍처 다이어그램:**
  + 계열사 → DataStage (ETL) → Data Lake (HDFS, S3)
  + Data Lake → Kafka → Spark Streaming → BI 포털
  + BI 포털 → Tableau, PowerBI → 실시간 분석 및 리포팅

#### 🔍 ****7. 테스트 및 검증 방안****

* **실시간 처리 테스트:**
  + Kafka 스트림 처리 후 1초 이내 분석 결과 생성
  + BI 포털에서 실시간 대시보드 업데이트 확인
* **DR 시나리오 테스트:**
  + 장애 발생 시 DR 센터로 15분 이내 복구 확인
  + S3에 백업된 데이터가 정상적으로 복원되는지 검증
* **보안 검증:**
  + TLS 1.3 암호화가 정상적으로 이루어지는지 Wireshark 패킷 분석
  + RBAC 정책이 Apache Ranger에서 올바르게 적용되는지 확인

# 📌 8. 아키텍처 주요 정보

## 8.1 도입 전 안내말

* NH 금융지주의 통합 데이터 플랫폼 구축은 기존 시스템을 최대한 활용하면서,  
  신규 도입이 필요한 시스템을 최적화하여 구성되었습니다.  
  주요 목표는 **확장성, 실시간 분석, 무중단 서비스, 데이터 통합**이며,  
  이를 위해 최신 기술 스택을 적용하고 표준화된 인터페이스를 도입하였습니다.

## 8.2 도입 시스템 정의

* 통합 데이터 플랫폼을 구성하는 시스템은 **신규 도입 시스템**과 **기존 시스템 활용**으로 나뉩니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 시스템명 | 설명 | 기술 스택 |
| 신규 도입 | 통합 데이터 DB | 금융 거래 및 분석 데이터를 통합하여 저장 | Oracle HEX |
|  | 통합 데이터 운영 DB Storage | 장기 보관 및 분석 목적의 스토리지 구성 | 3TB 용량, RAID 10 |
|  | 통합 데이터 ETL 서버 | 배치 처리 및 실시간 데이터 클렌징 수행 | IBM DataStage |
|  | BI 포털 DB | 분석 결과 시각화를 위한 OLAP DB | MySQL, Tableau |
|  | API Gateway | RESTful API 관리 및 인증 처리 | Spring Cloud Gateway, Zuul |
| 기존 시스템 활용 | WEB 서버 | 외부 채널과의 연동을 위한 웹 서비스 관리 | Apache, xFRAME UI 솔루션 |
|  | PT WAS 서버 | 실시간 거래 처리 및 애플리케이션 로직 관리 | Tomcat, Spring Boot |
|  | BT WAS 서버 | 배치 처리 및 마케팅 캠페인 실행 | Tomcat, Spring Boot |
|  | 마케팅 HUB 서버 | 실시간 캠페인 관리 및 고객 세분화 분석 | Tomcat, Spring Boot |
|  | 은행 빅스퀘어 서버 | 금융 거래 분석 및 시각화를 위한 분석 엔진 | Tomcat, Spring Boot |
|  | e-마케팅 시스템 서버 | 고객 맞춤형 마케팅 자동화 시스템 | Tomcat, Spring Boot |
|  | UMS 서버 | 메일, 문자, 푸시 알림 등의 대량 발송 관리 | WebLogic |
|  | 올원뱅크 채널 서버 | 올원뱅크 사용자 행위 분석 및 거래 처리 | Apache, Tomcat, Spring Boot |

## 8.3 인터페이스 구성도

* 통합 데이터 플랫폼의 주요 인터페이스는 **계열사, 외부 기관, 분석 플랫폼** 간의 데이터 연계를 담당합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 연계 시스템 | 설명 | 기술 스택 |
| 계열사 인터페이스 | NH농협은행, NH카드, NH투자증권 | 배치 및 실시간 금융 거래 데이터 수집 | WebSphere MQ, IBM DataStage |
| 외부 기관 인터페이스 | 금융감독원, 공공기관 | 실시간 금융 정보 및 보고서 연계 | FEP, MCA, SFTP |
| BI 포털 연계 | Tableau, PowerBI | 금융 분석 결과 시각화, 캠페인 성과 분석 | Tableau, PowerBI |
| 실시간 마케팅 인터페이스 | 마케팅 허브, UMS | 고객 맞춤형 마케팅 실행 및 알림 발송 | Kafka, Spring Cloud Stream |
| 분석 플랫폼 연계 | Spark Streaming, Big Square | 실시간 데이터 분석 및 머신러닝 모델 학습 | Spark, HDFS, Kafka |

## 8.4 데이터 주제 영역 정의

* 통합 데이터 플랫폼에서 관리하는 주요 데이터 주제 영역은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 주제영역 | 세부 항목 | 설명 |
| 고객정보 | 고객기본정보, 고객실적정보, 고객관리정보 | 개인 및 법인 고객의 금융 거래 및 관리 정보 |
| 상품정보 | 상품기본정보, 상품분류정보 | 금융 상품의 세부 정보 (대출, 예적금, 카드, 보험 등) |
| 거래정보 | 수신거래, 여신거래, 카드거래, 외환거래 | 금융 거래 내역 및 결제 정보 |
| 채널정보 | 가입채널정보, 채널이용정보, 상담정보 | 고객이 접근한 금융 채널 (모바일, 인터넷뱅킹, ATM) |
| 공통정보 | 공통코드, 사무소/가맹점정보, 기타 공통정보 | 모든 계열사에서 공통적으로 사용하는 참조 정보 |
| 분석정보 | OLAP 분석, 마케팅 성과 분석 | BI 및 Big Square를 통한 통계 및 예측 분석 |

## 8.5 도입 솔루션 목록

* NH 금융지주 통합 데이터 플랫폼 구축 시 사용된 주요 솔루션은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 솔루션명 | 기능 설명 | 적용 영역 |
| Spring Boot | 마이크로서비스 개발 및 REST API 제공 | Application Layer |
| Zuul | API Gateway를 통한 라우팅 및 보안 관리 | Interface Layer |
| Kubernetes | 마이크로서비스 클러스터 관리 및 Auto Scaling | Deployment & Scaling |
| Kafka | 실시간 스트림 처리 및 이벤트 전송 | Streaming Layer |
| DataStage | ETL(Extract, Transform, Load) 처리 | ETL Processing |
| HDFS | 대량 데이터 저장 및 분산 처리 | Data Lake |
| Prometheus | 실시간 모니터링 및 장애 탐지 | Monitoring Layer |
| Grafana | 시각화 대시보드 및 실시간 상태 표시 | Monitoring & Analytics |
| Tableau | BI 분석 및 시각화 | BI Portal |
| PowerBI | 실시간 보고서 생성 및 분석 | BI Portal |
| Apache Ranger | RBAC 기반의 접근 통제 관리 | Security Layer |

# 📌 **9. 배포 및 운영 전략**

## **9.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼의 배포 및 운영 전략은 **무중단 서비스, 실시간 장애 복구, 확장성 최적화**를 중심으로 설계되었습니다.  
Microservice 기반의 애플리케이션은 Kubernetes 클러스터에 배포되며, **ArgoCD, Jenkins, GitLab CI/CD**를 통해  
자동화된 배포 및 모니터링이 진행됩니다.

## **9.2 CI/CD 파이프라인 설계**

모든 애플리케이션 배포는 GitLab CI/CD 파이프라인을 통해 자동화됩니다.  
코드가 커밋되면 GitLab Runner가 빌드 및 테스트를 수행하고, Docker 이미지를 생성하여  
Kubernetes 클러스터에 ArgoCD를 통해 배포합니다.

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **코드 커밋 (Commit)** | - GitLab에 소스 코드가 커밋되면 CI 파이프라인이 트리거됨 | GitLab, GitOps |
| **빌드 (Build)** | - Maven을 통해 빌드가 진행되며, Docker 이미지를 생성 | Maven, Docker |
| **테스트 (Test)** | - JUnit을 통한 단위 테스트 및 SonarQube를 통한 정적 분석 수행 | JUnit, SonarQube |
| **배포 (Deploy)** | - ArgoCD를 통해 Kubernetes 클러스터에 무중단 배포 | ArgoCD, Kubernetes |
| **모니터링 (Monitor)** | - Prometheus와 Grafana를 통해 실시간 모니터링 및 장애 탐지 | Prometheus, Grafana |
| **롤백 (Rollback)** | - 배포 실패 시 이전 안정 버전으로 자동 롤백 처리 | ArgoCD Rollback |

### ****9.3 블루-그린 배포, 롤링 업데이트****

NH 금융지주는 **블루-그린 배포**와 **롤링 업데이트**를 통해 무중단 서비스를 실현합니다.

|  |  |  |
| --- | --- | --- |
| **배포 방식** | **설명** | **적용 기술** |
| **블루-그린 배포** | - 기존 서비스(Blue)와 신규 버전(Green)을 동시에 배포 - 테스트 완료 후 트래픽을 Green으로 전환 | Kubernetes, ArgoCD |
| **롤링 업데이트** | - Pod 단위로 순차적으로 업데이트하여 트래픽 손실 없이 배포 | Kubernetes HPA |
| **Canary 배포** | - 일정 비율의 트래픽만 신규 버전으로 보내 점진적 검증 | Istio, Argo Rollouts |
| **Zero-Downtime** | - 배포 중에도 모든 서비스가 정상 작동 | Nginx, HAProxy |

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

**DR(Disaster Recovery)** 전략은 주요 장애 발생 시 15분 이내로 복구되도록 설계되었습니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **Auto Healing** | - Kubernetes의 Auto Healing 기능으로 Pod 장애 시 자동 재시작 | Kubernetes, Prometheus |
| **Failover 구성** | - Active-Standby 구조로 실시간 Failover 처리 | Oracle RAC, MySQL Cluster |
| **DR 센터 전환** | - 장애 발생 시 AWS S3에 백업된 데이터를 DR 센터로 전환 | AWS S3, Route 53 |
| **Snapshot Backup** | - 매일 증분 백업 및 매주 전체 백업을 S3에 저장 | AWS Backup, DataStage |
| **RPO (Recovery Point Objective)** | - 15분 이내 데이터 복구 목표 | S3 Snapshot, HDFS Sync |
| **RTO (Recovery Time Objective)** | - 30분 이내 서비스 복구 목표 | Kubernetes, ArgoCD |

## **9.5 실시간 모니터링 및 로깅 관리**

통합 데이터 플랫폼은 실시간 상태 모니터링과 로그 수집을 통해 장애를 빠르게 탐지하고 대응합니다.

|  |  |  |
| --- | --- | --- |
| **구성요소** | **설명** | **기술 스택** |
| **실시간 모니터링** | - Prometheus를 통한 실시간 리소스 모니터링 - Grafana 대시보드를 통한 시각화 | Prometheus, Grafana |
| **로그 수집 및 분석** | - ELK Stack (Elasticsearch, Logstash, Kibana) 활용 - 모든 서비스 로그 중앙 관리 | ELK Stack |
| **알림 및 대응** | - AlertManager를 통해 장애 발생 시 Slack 및 이메일 알림 전송 | Slack, PagerDuty |
| **대시보드 구성** | - 애플리케이션 성능, DB 사용량, 네트워크 트래픽, 에러율을 시각화 | Grafana, Kibana |

## **9.6 장애 대응 및 복구 시나리오**

모든 장애 상황에 대한 대응 시나리오를 미리 정의하고, 자동화된 복구가 가능하도록 설계하였습니다.

|  |  |  |
| --- | --- | --- |
| **장애 유형** | **대응 시나리오** | **복구 시간 목표 (RTO)** |
| **애플리케이션 장애** | - Kubernetes Auto Healing으로 Pod 자동 재시작 | 5분 이내 |
| **DB 장애** | - Oracle RAC와 MySQL Cluster의 Active-Standby 전환 | 10분 이내 |
| **네트워크 장애** | - Nginx, HAProxy를 통한 트래픽 재조정 및 복구 | 3분 이내 |
| **API Gateway 장애** | - Spring Cloud Gateway를 Failover 구성으로 전환 | 2분 이내 |
| **대규모 장애 (DR)** | - AWS S3에 백업된 스냅샷을 DR 센터로 복원 | 15분 이내 |

#### 🔍 ****7. 설계 다이어그램 예시****

* **배포 및 운영 아키텍처 다이어그램:**
  + 개발자 → GitLab CI/CD → Docker Build
  + Docker Image → ArgoCD → Kubernetes Cluster
  + Prometheus → Grafana → 실시간 모니터링
  + ArgoCD → Blue-Green, Rolling Update

#### 🔍 ****8. 테스트 및 검증 방안****

* **CI/CD 배포 테스트:**
  + GitLab에서 커밋 시 Docker 이미지 생성 및 Kubernetes 배포 확인
  + ArgoCD에서 Blue-Green 전환이 정상 동작하는지 검증
* **DR 복구 테스트:**
  + DR 센터로의 스냅샷 복원 시간 측정
  + AWS S3에서 백업된 데이터의 복구 여부 확인
* **모니터링 및 알림 테스트:**
  + Prometheus AlertManager가 장애 발생 시 Slack 알림을 정상 전송하는지 확인
  + Kibana에서 오류 로그 분석

# 📌 **10. 로그 및 모니터링 전략**

## **10.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서 **로그 수집 및 모니터링 전략**은 시스템 장애를 빠르게 탐지하고 대응하기 위해 설계되었습니다.  
모든 애플리케이션의 실행 로그, 데이터베이스 트랜잭션, 네트워크 통신 로그를 실시간으로 수집하며,  
\*\*ELK Stack (Elasticsearch, Logstash, Kibana)\*\*를 통해 중앙 집중 관리합니다.  
또한, **Prometheus, Grafana**를 통해 실시간 리소스 사용량과 트래픽 상태를 모니터링합니다.

## **10.2 로그 수집 및 분석 설계**

모든 로그는 **ELK Stack**을 통해 실시간으로 수집되며, 장애 발생 시 빠르게 탐지하고 대응할 수 있습니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **Application Logs** | - Spring Boot 및 Tomcat에서 발생하는 애플리케이션 로그 - 거래 내역, 사용자 액션 기록 | Logstash, Elasticsearch |
| **Database Logs** | - Oracle DB 및 MySQL의 트랜잭션 로그 - 쿼리 성능 및 장애 탐지 | Filebeat, Elasticsearch |
| **API Gateway Logs** | - Spring Cloud Gateway의 트래픽 로그 - 인증 실패, 권한 오류 기록 | Fluentd, Elasticsearch |
| **Security Logs** | - OAuth2.0 인증 및 접근 기록 - JWT 토큰 검증 실패 로그 | Apache Ranger, ELK Stack |
| **ETL Processing Logs** | - DataStage의 배치 처리 결과 및 오류 로그 | Logstash, HDFS |
| **Network Logs** | - FEP, MCA, MFT 전송 상태 기록 | Prometheus, Grafana |

## **10.3 모니터링 아키텍처 설계**

통합 데이터 플랫폼은 실시간 상태 모니터링을 통해 시스템 가용성을 유지하고 성능 저하를 방지합니다.

|  |  |  |
| --- | --- | --- |
| **모니터링 항목** | **설명** | **기술 스택** |
| **CPU 사용량** | - 모든 애플리케이션의 CPU 사용 현황을 실시간 모니터링 | Prometheus, Grafana |
| **메모리 사용량** | - JVM Heap Memory 및 GC 상태 관리 | Prometheus, Grafana |
| **네트워크 트래픽** | - Inbound/Outbound 트래픽 모니터링 | Prometheus, Grafana |
| **TPS (Transactions Per Second)** | - 초당 거래 처리량 실시간 체크 | Prometheus, Grafana |
| **에러 발생 현황** | - Spring Boot 및 API Gateway에서 발생하는 예외 상황 탐지 | Kibana, Grafana |
| **DB 쿼리 응답 시간** | - Oracle DB 및 MySQL의 쿼리 응답 시간 측정 | Prometheus, Grafana |

#### 🔍 ****1. Prometheus & Grafana 구성****

* Prometheus는 애플리케이션과 Kubernetes 클러스터의 실시간 메트릭을 수집합니다.
* Grafana는 수집된 데이터를 시각화하여 **대시보드**에 표시합니다.

**주요 대시보드 항목:**

* 애플리케이션별 CPU 및 메모리 사용량
* DB Connection Pool 상태
* 네트워크 I/O 및 트래픽 분석
* TPS 모니터링 및 SLA 준수 여부

#### 🔍 ****2. ELK Stack 구성****

* **Logstash:**
  + 애플리케이션 로그, DB 로그, API 로그를 수집하여 Elasticsearch로 전달
  + JSON 포맷으로 파싱하여 검색 최적화
* **Elasticsearch:**
  + 중앙 집중형 로그 저장소
  + Kibana에서 실시간 조회 가능
* **Kibana:**
  + 로그 데이터를 시각화하여 장애 분석 및 탐지
  + 검색 키워드 예시:
    - "transactionId:12345" → 특정 거래 내역 조회
    - "ERROR" → 애플리케이션 오류 로그 검색

#### 🔍 ****3. 알림 및 대응 체계****

* **AlertManager:**
  + Prometheus에서 장애 발생 시 AlertManager에 알림 전달
  + Slack, Email, PagerDuty로 실시간 알림 전송
* **Slack Notification:**
  + 주요 서비스 장애 시 DevOps 팀에 즉시 알림
  + 예: "API Gateway CPU 사용량 90% 초과", "DB Connection Pool 소진"
* **자동 대응 시나리오:**
  + 특정 트리거 발생 시 자동 롤백 실행 (ArgoCD)
  + 네트워크 장애 시 재시도 정책 적용

#### 🔍 ****4. 데이터 보존 및 관리 정책****

* **보존 주기:**
  + 모든 애플리케이션 로그는 **90일간 보관**
  + 보안 관련 로그는 **180일간 보관** 후 암호화하여 S3에 백업
* **암호화 저장:**
  + Elasticsearch에 저장된 모든 민감 정보는 **AES-256** 암호화 적용
* **스냅샷 생성:**
  + 주 단위로 스냅샷 생성 후 AWS S3에 백업
  + 장애 발생 시 복구 포인트로 활용

#### 🔍 ****5. 설계 다이어그램 예시****

* **모니터링 및 로깅 아키텍처 다이어그램:**
  + Application → Fluentd → Logstash → Elasticsearch
  + DB → Filebeat → Elasticsearch
  + Prometheus → Grafana → 실시간 대시보드
  + AlertManager → Slack, PagerDuty

#### 🔍 ****6. 테스트 및 검증 방안****

* **로그 수집 테스트:**
  + Spring Boot 애플리케이션에서 발생한 에러 로그가 Kibana에 실시간 반영되는지 확인
  + DataStage 배치 처리 중 발생한 오류가 Logstash를 통해 적재되는지 검증
* **모니터링 대시보드 테스트:**
  + Prometheus에서 수집한 CPU 및 메모리 사용량이 Grafana에 정상 표시되는지 확인
  + TPS 수치가 SLA 기준 이내로 유지되는지 체크
* **알림 시스템 테스트:**
  + 네트워크 장애 시 Slack에 실시간 알림이 전송되는지 확인
  + DB Connection Pool 고갈 시 PagerDuty에 자동 티켓 생성

# 📌 **11. API 관리 및 거버넌스 전략**

## **11.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼에서 **API 관리 및 거버넌스 전략**은 내부 서비스 간 통신과 외부 금융 서비스와의 연계를  
효율적으로 관리하기 위한 핵심 요소입니다.  
API Gateway를 통해 모든 트래픽을 중앙 집중적으로 관리하며, OAuth2.0 및 JWT 기반의 인증 체계를 통해 보안을 강화합니다.  
또한, API 버전 관리를 통해 하위 호환성을 유지하고, 거버넌스 체계를 통해 모든 API의 접근 권한과 사용 현황을 체계적으로 관리합니다.

## **11.2 API 설계 표준화**

모든 API는 **RESTful API 표준**에 맞춰 설계되며, **HTTP Methods**를 명확히 구분합니다.

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **설명** | **예시 엔드포인트** |
| **GET** | - 리소스 조회 | /api/v1/accounts/{accountId} |
| **POST** | - 리소스 생성 | /api/v1/customers |
| **PUT** | - 리소스 전체 업데이트 | /api/v1/customers/{customerId} |
| **PATCH** | - 리소스 부분 업데이트 | /api/v1/accounts/{accountId} |
| **DELETE** | - 리소스 삭제 | /api/v1/customers/{customerId} |

* **URI Naming Rule:**
  + 모든 엔드포인트는 복수형 명칭 사용 (e.g., /customers, /accounts)
  + 서브 리소스 접근 시 계층 구조 사용 (e.g., /accounts/{accountId}/transactions)
* **HTTP Status Code 표준화:**
  + 200: 성공 (OK)
  + 201: 생성됨 (Created)
  + 204: 삭제됨 (No Content)
  + 400: 잘못된 요청 (Bad Request)
  + 401: 인증 실패 (Unauthorized)
  + 403: 권한 없음 (Forbidden)
  + 404: 리소스 없음 (Not Found)
  + 500: 서버 오류 (Internal Server Error)

## **11.3 API 버전 관리 및 확장 전략**

NH 금융지주는 API의 안정성과 호환성을 보장하기 위해 **버전 관리**를 체계적으로 수행합니다.

|  |  |  |
| --- | --- | --- |
| **버전 관리 방식** | **설명** | **예시 엔드포인트** |
| **URI Versioning** | - URI에 버전 정보 명시 (v1, v2) | /api/v1/accounts |
| **Header Versioning** | - HTTP Header에 버전 정보 추가 (X-API-Version) | X-API-Version: v1 |
| **Content Negotiation** | - Accept Header를 통한 버전 구분 | Accept: application/vnd.nhbank.v1+json |
| **Deprecation Policy** | - 기존 API는 Deprecated 상태로 유지하고 6개월 후 종료 알림 | Kibana Dashboard 모니터링 |

* **버전 관리 정책:**
  + 메이저 버전 (v1, v2)은 구조 변경 시 적용
  + 마이너 버전 (v1.1, v1.2)은 기능 추가 시 적용
  + 패치 버전 (v1.0.1)은 버그 수정 시 적용

## **11.4 API 보안 및 접근 제어**

모든 API는 **OAuth2.0**과 \*\*JWT (JSON Web Token)\*\*를 통해 인증 및 인가가 이루어집니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술 스택** |
| **OAuth2.0 인증** | - 사용자 인증을 통해 Access Token 발급 - Refresh Token을 통해 재인증 | Keycloak, Spring Security |
| **JWT 토큰 인증** | - API 호출 시 JWT를 통해 사용자 인증 처리 | JWT, Spring Boot |
| **SSO (Single Sign-On)** | - 하나의 인증으로 여러 서비스 접근 | Keycloak, OpenAM |
| **RBAC (Role-Based Access Control)** | - Role에 따른 API 접근 권한 관리 | Apache Ranger |
| **Rate Limiting** | - 사용자별 호출 횟수 제한 (Rate Limiting) | Spring Cloud Gateway |
| **CORS 설정** | - 외부 도메인에서 API 호출 시 허용 도메인 제한 | Spring Boot, Nginx |

#### 🔍 ****1. OAuth2.0 인증 및 JWT 발급 프로세스****

1. 클라이언트가 OAuth2.0 인증 요청
2. Keycloak이 사용자 인증 후 Access Token, Refresh Token 발급
3. API 호출 시 Header에 JWT 포함 (Authorization: Bearer {token})
4. Spring Boot에서 JWT 검증 후 서비스 처리

#### 🔍 ****2. RBAC 기반 접근 제어****

* **Admin Role:** 모든 서비스 접근 권한
* **User Role:** 자신의 거래 내역 및 금융 정보 접근 권한
* **Viewer Role:** 분석 결과 조회만 가능

|  |  |  |
| --- | --- | --- |
| **Role** | **허용 API** | **설명** |
| **Admin** | /api/v1/admin/\* | 모든 시스템 관리 및 설정 접근 |
| **User** | /api/v1/accounts, /api/v1/customers | 자신의 금융 정보 접근 |
| **Viewer** | /api/v1/reports/\* | 분석 보고서 조회 |

#### 🔍 ****3. Rate Limiting 및 CORS 정책****

* **Rate Limiting:**
  + 사용자 IP 기준으로 초당 최대 100회 요청 제한
  + Spring Cloud Gateway에서 설정
* **CORS 정책:**
  + https://nhbank.com, https://nhcard.com 만 허용
  + 외부 도메인 접근 시 403 Forbidden 응답

#### 🔍 ****4. 설계 다이어그램 예시****

* **API Gateway 아키텍처 다이어그램:**
  + 외부 요청 → Spring Cloud Gateway → OAuth2.0 인증
  + 인증 후 → Microservice → DB 또는 외부 서비스 연동

#### 🔍 ****5. 테스트 및 검증 방안****

* **OAuth2.0 인증 테스트:**
  + Access Token 만료 시 Refresh Token을 통한 갱신 여부 확인
  + 만료된 토큰으로 접근 시 401 Unauthorized 응답 확인
* **JWT 발급 및 검증 테스트:**
  + 발급된 JWT가 만료 시간 이내에 정상 동작하는지 확인
  + 잘못된 JWT가 전달될 경우 403 Forbidden 응답
* **Rate Limiting 테스트:**
  + 100회 초과 요청 시 429 Too Many Requests 응답 확인
* **CORS 정책 테스트:**
  + 허용되지 않은 도메인에서 접근 시 403 Forbidden 발생 여부

# 📌 **12. DevOps 및 CI/CD 전략**

## **12.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼의 DevOps 및 CI/CD 전략은 **무중단 배포, 빠른 회복, 확장성**을 목표로 설계되었습니다.  
모든 서비스는 Kubernetes 클러스터 상에서 동작하며, **GitOps 기반의 CI/CD 파이프라인**을 통해 지속적인 통합과 배포가 자동화됩니다.  
이를 통해 개발에서 운영까지의 전 과정을 신속하게 관리하고, 서비스 중단 없이 업데이트가 가능하도록 최적화하였습니다.

## **12.2 DevOps 전략 개요**

DevOps 전략은 \*\*Infrastructure as Code (IaC)\*\*와 **GitOps**를 중심으로 구성되며,  
모든 인프라와 애플리케이션의 배포는 코드로 관리되어 **자동화**와 **버전 관리**가 용이합니다.

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **설명** | **기술 스택** |
| **소스 관리** | - 모든 애플리케이션 및 인프라 코드는 GitLab에 버전 관리 | GitLab |
| **빌드 자동화** | - Maven과 Docker를 통해 애플리케이션 빌드 및 이미지 생성 | Maven, Docker |
| **CI/CD 파이프라인** | - GitLab CI/CD를 통해 자동화된 빌드, 테스트, 배포 실행 | GitLab CI, ArgoCD |
| **배포 자동화** | - ArgoCD를 통한 Kubernetes 클러스터에 자동 배포 | ArgoCD, Kubernetes |
| **모니터링 및 로깅** | - Prometheus와 Grafana로 실시간 모니터링, ELK Stack으로 로그 분석 | Prometheus, Grafana, ELK Stack |
| **Rollback 관리** | - ArgoCD를 통한 무중단 롤백 지원 | Argo Rollouts |

## **12.3 CI/CD 파이프라인 설계**

NH 금융지주의 CI/CD 파이프라인은 **코드 커밋 → 빌드 → 테스트 → 배포**의 순서로 자동화되며,  
GitLab의 Merge Request 시 모든 과정이 실행됩니다.

|  |  |  |
| --- | --- | --- |
| **단계** | **설명** | **기술 스택** |
| **1. 소스 코드 커밋** | - GitLab에 소스 코드가 커밋되면 CI 파이프라인이 트리거됨 | GitLab, GitOps |
| **2. 빌드 및 Docker 이미지 생성** | - Maven 빌드 후 Docker 이미지를 생성하여 레지스트리에 저장 | Maven, Docker |
| **3. 정적 분석 및 테스트** | - SonarQube를 활용한 정적 분석 및 JUnit 단위 테스트 실행 | SonarQube, JUnit |
| **4. 컨테이너 배포 준비** | - 생성된 Docker 이미지를 ArgoCD가 Kubernetes에 배포 준비 | ArgoCD, Kubernetes |
| **5. Canary 배포 및 검증** | - 일부 트래픽을 새로운 버전으로 전달하여 안정성 테스트 | Argo Rollouts, Istio |
| **6. 블루-그린 전환** | - Canary 배포 검증 후 트래픽 전환, 실패 시 이전 버전으로 롤백 | Nginx, HAProxy |
| **7. 모니터링 및 검증** | - Prometheus와 Grafana를 통해 리소스 사용량 및 에러 검증 | Prometheus, Grafana |
| **8. 로그 수집 및 분석** | - ELK Stack을 활용하여 모든 로그 중앙 집중 관리 | ELK Stack |

## **12.4 자동화 배포 및 모니터링**

**ArgoCD**를 활용하여 모든 애플리케이션 배포가 Kubernetes 클러스터에 무중단으로 진행됩니다.

|  |  |  |
| --- | --- | --- |
| **배포 전략** | **설명** | **적용 기술** |
| **Blue-Green 배포** | - 기존 서비스(Blue)와 새로운 버전(Green)을 동시에 배포 - 테스트 완료 후 트래픽 전환 | ArgoCD, Kubernetes |
| **Canary 배포** | - 10% → 25% → 50% → 100% 순서로 점진적 트래픽 전환 | Istio, Argo Rollouts |
| **Rolling Update** | - Pod 단위로 순차적으로 업데이트하여 무중단 배포 진행 | Kubernetes HPA |
| **Zero-Downtime** | - 모든 배포 시 Nginx와 HAProxy를 통해 무중단 서비스 유지 | Nginx, HAProxy |

## **12.5 장애 복구 및 롤백 전략**

**ArgoCD**와 **Kubernetes의 Auto Healing** 기능을 활용하여, 장애 발생 시 빠르게 복구하고 이전 버전으로 롤백합니다.

|  |  |  |
| --- | --- | --- |
| **장애 유형** | **대응 시나리오** | **복구 시간 목표 (RTO)** |
| **애플리케이션 장애** | - Kubernetes Auto Healing으로 Pod 자동 재시작 | 5분 이내 |
| **DB 장애** | - Oracle RAC와 MySQL Cluster의 Active-Standby 전환 | 10분 이내 |
| **네트워크 장애** | - Nginx, HAProxy를 통한 트래픽 재조정 및 복구 | 3분 이내 |
| **API Gateway 장애** | - Spring Cloud Gateway를 Failover 구성으로 전환 | 2분 이내 |
| **대규모 장애 (DR)** | - AWS S3에 백업된 스냅샷을 DR 센터로 복원 | 15분 이내 |

#### 🔍 ****1. 롤백 시나리오 예시****

1. Canary 배포 중 장애 탐지 → 트래픽 100% 롤백
2. Prometheus AlertManager가 오류 탐지 → ArgoCD에 롤백 요청
3. 이전 버전 Docker Image로 배포 → 서비스 정상화

#### 🔍 ****2. 모니터링 및 알림 체계****

* **Prometheus & Grafana:**
  + 실시간 모니터링 대시보드 생성
  + CPU, Memory, Network 상태 시각화
* **AlertManager:**
  + 주요 서비스 장애 시 Slack, Email, PagerDuty로 실시간 알림
  + 예: "API Gateway Down", "DB Connection Timeout"

#### 🔍 ****3. 설계 다이어그램 예시****

* **CI/CD 아키텍처 다이어그램:**
  + GitLab → Docker Build → ArgoCD → Kubernetes
  + Prometheus → Grafana → 실시간 모니터링
  + ArgoCD → Blue-Green, Rolling Update

#### 🔍 ****4. 테스트 및 검증 방안****

* **CI/CD 배포 테스트:**
  + GitLab에서 커밋 시 Docker 이미지 생성 및 Kubernetes 배포 확인
  + ArgoCD에서 Blue-Green 전환이 정상 동작하는지 검증
* **DR 복구 테스트:**
  + DR 센터로의 스냅샷 복원 시간 측정
  + AWS S3에서 백업된 데이터의 복구 여부 확인
* **모니터링 및 알림 테스트:**
  + Prometheus AlertManager가 장애 발생 시 Slack 알림을 정상 전송하는지 확인
  + Kibana에서 오류 로그 분석

# 📌 **13. 운영 및 유지보수 전략**

## **13.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼은 **24/7 무중단 서비스 운영**을 목표로 하며,  
애플리케이션 및 데이터베이스의 안정성을 보장하기 위한 운영 관리 전략이 필요합니다.  
이를 위해 **자동화된 모니터링, 빠른 장애 복구, 주기적 백업, 보안 업데이트**를 체계적으로 관리합니다.

## **13.2 운영 관리 개요**

운영 관리는 **애플리케이션 관리, 인프라 관리, 데이터 관리** 세 가지 영역으로 구분됩니다.

|  |  |  |
| --- | --- | --- |
| **관리 항목** | **설명** | **기술 스택** |
| **애플리케이션 관리** | - Spring Boot 및 Microservices 관리 - API Gateway 모니터링 | Spring Boot, Zuul |
| **인프라 관리** | - Kubernetes 클러스터 관리 - Auto Healing 및 Auto Scaling | Kubernetes, Prometheus |
| **데이터 관리** | - Data Lake 및 RDBMS 관리 - HDFS 및 S3 백업 관리 | HDFS, AWS S3 |
| **보안 관리** | - OAuth2.0 인증 및 RBAC 접근 통제 | Keycloak, Apache Ranger |
| **로그 및 모니터링** | - ELK Stack을 통한 실시간 로그 수집 - Grafana 대시보드 시각화 | ELK Stack, Grafana |

## **13.3 장애 대응 및 복구 시나리오**

통합 데이터 플랫폼은 예상 가능한 장애에 대해 **대응 시나리오**를 미리 정의하고,  
자동화된 복구 메커니즘을 통해 신속하게 정상화됩니다.

|  |  |  |
| --- | --- | --- |
| **장애 유형** | **대응 시나리오** | **복구 시간 목표 (RTO)** |
| **애플리케이션 장애** | - Kubernetes Auto Healing으로 Pod 자동 재시작 | 5분 이내 |
| **DB 장애** | - Oracle RAC와 MySQL Cluster의 Active-Standby 전환 | 10분 이내 |
| **네트워크 장애** | - Nginx, HAProxy를 통한 트래픽 재조정 및 복구 | 3분 이내 |
| **API Gateway 장애** | - Spring Cloud Gateway를 Failover 구성으로 전환 | 2분 이내 |
| **대규모 장애 (DR)** | - AWS S3에 백업된 스냅샷을 DR 센터로 복원 | 15분 이내 |

## **13.4 보안 업데이트 및 패치 관리**

보안 업데이트와 패치 관리는 **주기적 업데이트**와 **긴급 보안 패치** 두 가지 방식으로 진행됩니다.

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **적용 기술** |
| **주기적 업데이트** | - 매월 정기적인 보안 업데이트 및 최신 패치 적용 | Ansible, Jenkins |
| **긴급 보안 패치** | - Zero-day 취약점 발견 시 즉시 핫픽스 배포 | GitOps, ArgoCD |
| **OWASP 검사** | - OWASP Top 10에 대한 주기적 보안 검사 | ZAP, SonarQube |
| **API 보안 강화** | - OAuth2.0 및 JWT 토큰 만료 시간 관리 | Keycloak, Spring Security |
| **SSL/TLS 관리** | - 만료된 인증서 자동 갱신 및 TLS 1.3 암호화 적용 | Let's Encrypt, Cert Manager |

## **13.5 데이터 백업 및 복구 전략**

모든 금융 데이터는 **실시간 백업 및 주기적 스냅샷**을 통해 보관되며, 장애 발생 시 신속히 복구할 수 있는 구조로 설계되었습니다.

|  |  |  |  |
| --- | --- | --- | --- |
| **백업 유형** | **설명** | **주기** | **기술 스택** |
| **실시간 백업** | - 모든 트랜잭션 로그를 실시간으로 HDFS에 적재 | 실시간 | HDFS, DataStage |
| **일일 증분 백업** | - 하루 동안 변경된 데이터만 AWS S3에 증분 백업 | 매일 자정 | AWS S3, DataStage |
| **주간 전체 백업** | - 매주 주말에 모든 데이터를 풀 백업 | 매주 일요일 자정 | AWS S3, DataStage |
| **월간 스냅샷 백업** | - 주요 DB의 스냅샷을 생성하고 DR 센터에 복제 | 매월 첫째 주 월요일 | AWS RDS, DataStage |
| **DR 복구 테스트** | - DR 센터로 전환 시 15분 이내 복구 여부 확인 | 분기별 1회 | AWS S3, RDS |

## **13.6 스케줄링 및 점검 주기**

* **일일 점검:**
  + Prometheus와 Grafana를 통해 리소스 사용량 및 오류 로그 분석
  + DataStage에서 ETL 작업 완료 여부 확인
* **주간 점검:**
  + Kibana에서 로그 분석을 통해 에러 발생률 및 처리 지연 분석
  + Kafka 스트림 정상 여부 확인
* **월간 점검:**
  + 주요 API Gateway와 Spring Boot 서비스의 성능 점검
  + SSL 인증서 만료일 확인 및 갱신
* **분기별 점검:**
  + DR 센터로의 백업 복구 테스트
  + OWASP Top 10 보안 점검 및 취약점 분석

#### 🔍 ****1. 운영 다이어그램 예시****

* **운영 및 유지보수 아키텍처 다이어그램:**
  + Application → ELK Stack → Kibana 대시보드
  + Prometheus → Grafana → 실시간 모니터링
  + Kubernetes → Auto Healing & Scaling
  + AWS S3 → 주간 백업 → DR 센터 복구

#### 🔍 ****2. 테스트 및 검증 방안****

* **백업 및 복구 테스트:**
  + 일일 증분 백업에서 데이터가 정상 복원되는지 확인
  + DR 센터에서 15분 이내 복구가 가능한지 검증
* **보안 업데이트 테스트:**
  + OWASP 검사를 통한 보안 취약점 탐지
  + TLS 1.3 암호화 통신이 정상적으로 유지되는지 확인
* **모니터링 및 대응 테스트:**
  + 장애 발생 시 AlertManager가 Slack에 실시간 알림 전송 여부 확인
  + Grafana 대시보드에서 CPU, Memory 사용량 정상 표시 여부

# 📌 **14. 최적화 및 성능 개선 전략**

## **14.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼의 성능 최적화는 **대량 트랜잭션 처리, 실시간 분석, 무중단 서비스**를 목표로 설계되었습니다.  
대용량 데이터 처리 시 병목 현상을 최소화하고, **Kubernetes Auto Scaling, DB 최적화, 애플리케이션 튜닝**을 통해 성능을 극대화합니다.  
또한, 트랜잭션 처리 시간 단축과 데이터 접근 최적화를 통해 **고객 응답 시간 3초 이내**를 보장합니다.

## **14.2 최적화 전략 개요**

|  |  |  |
| --- | --- | --- |
| **최적화 영역** | **설명** | **적용 기술** |
| **애플리케이션 최적화** | - Spring Boot 서비스의 비동기 처리, Redis 캐시 적용 | Spring Boot, Redis |
| **데이터베이스 최적화** | - 인덱스 최적화, 파티셔닝, Query 튜닝 | Oracle HEX, MySQL |
| **네트워크 최적화** | - CDN 및 로드 밸런싱 적용, Nginx 최적화 | Nginx, HAProxy |
| **캐싱 전략** | - Redis를 활용한 데이터 캐싱 - API Gateway에서 요청 캐싱 처리 | Redis, Spring Cloud Gateway |
| **병렬 처리 최적화** | - Spark를 활용한 병렬 분석 처리 | Spark, HDFS |
| **Auto Scaling** | - Kubernetes HPA를 통한 동적 확장 | Kubernetes, Prometheus |

## **14.3 성능 최적화 설계**

#### 🔍 ****1. 애플리케이션 최적화****

* **Spring Boot Async 처리:**
  + 비동기 처리를 위한 @Async 사용
  + HTTP 요청 처리 시 Thread Pool을 활용하여 응답 시간 단축
* **Connection Pool 최적화:**
  + HikariCP를 사용하여 DB 연결 관리 최적화
  + 기본 설정:
    - Maximum Pool Size: 50
    - Connection Timeout: 5초
    - Idle Timeout: 10분
* **Circuit Breaker 적용:**
  + 외부 API 요청 시 장애 전파 방지
  + Resilience4J를 활용한 자동 장애 격리

#### 🔍 ****2. 데이터베이스 최적화****

* **인덱스 최적화:**
  + 자주 조회되는 컬럼에 B-Tree Index 및 Bitmap Index 생성
  + 예: customer\_id, transaction\_date
* **파티셔닝:**
  + 시간 기반 파티셔닝을 통해 테이블 크기 최적화
  + 예: 월별 파티셔닝 (PARTITION BY RANGE (transaction\_date))
* **Query 튜닝:**
  + 서브쿼리 최소화 및 JOIN 최적화
  + N+1 문제 방지를 위한 Fetch Join 사용
* **데이터 압축:**
  + Oracle HEX의 압축 기능을 통해 데이터 저장 공간 최적화

#### 🔍 ****3. 네트워크 최적화****

* **CDN(Content Delivery Network) 적용:**
  + 정적 리소스 (이미지, JS, CSS) 전송 최적화
  + Latency를 줄이고 사용자 응답 속도 향상
* **로드 밸런싱:**
  + Nginx와 HAProxy를 활용하여 트래픽 분산
  + API Gateway에 트래픽 조절 기능 추가
* **Keep-Alive 설정:**
  + HTTP Keep-Alive를 통해 TCP 연결을 재사용하여 네트워크 오버헤드 감소

#### 🔍 ****4. 캐싱 전략****

* **Redis Cache 최적화:**
  + 자주 조회되는 고객 정보 및 금융 거래 정보를 Redis에 캐싱
  + TTL(Time To Live)을 설정하여 만료 관리
  + 예:
    - customer\_info:{customer\_id} → TTL: 24시간
    - transaction\_list:{customer\_id} → TTL: 1시간
* **API Gateway 캐싱:**
  + GET 요청에 대해 Gateway 레벨에서 캐싱 처리
  + Spring Cloud Gateway의 Caffeine Cache 사용

#### 🔍 ****5. 병렬 처리 최적화****

* **Spark 기반 분석 최적화:**
  + Kafka에서 수집된 데이터를 Spark로 스트림 처리
  + RDD 및 DataFrame 최적화를 통해 병렬 분석 속도 향상
* **HDFS 최적화:**
  + 대용량 파일을 블록 단위로 분산 처리
  + Replication Factor를 2로 설정하여 안정성 강화

#### 🔍 ****6. Auto Scaling 최적화****

* **Horizontal Pod Autoscaler (HPA):**
  + CPU 사용량이 70%를 초과하면 자동으로 Pod를 확장
  + 최소 3개 → 최대 20개까지 동적 확장
* **Vertical Pod Autoscaler (VPA):**
  + 메모리 사용량에 따른 리소스 할당량 조정
  + Spring Boot의 JVM 메모리 자동 조정

#### 🔍 ****7. 설계 다이어그램 예시****

* **최적화 아키텍처 다이어그램:**
  + 사용자 요청 → API Gateway → Spring Boot → Redis Cache
  + DB 요청 → HikariCP → Oracle DB
  + 분석 요청 → Kafka → Spark Streaming → Data Lake

#### 🔍 ****8. 테스트 및 검증 방안****

* **Load Test:**
  + Apache JMeter를 활용한 대량 요청 테스트
  + 목표 TPS: 10,000
  + 응답 시간 3초 이내 확인
* **Stress Test:**
  + Kubernetes HPA가 정상적으로 Auto Scale 되는지 검증
  + 150% 이상 트래픽 증가 시 Pod 확장 확인
* **Database Performance Test:**
  + 인덱스 적용 전후 조회 속도 측정
  + 1초 이내 조회 완료 확인
* **Cache Hit Ratio Test:**
  + Redis의 캐시 적중률 90% 이상 유지 여부 확인
  + TTL 만료 시 재조회 테스트

### 📌 ****15. 통합 아키텍처 다이어그램 설계****

## **15.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼의 **통합 아키텍처 다이어그램**은 모든 시스템 구성 요소를  
**채널 계층 → 인터페이스 계층 → 비즈니스 계층 → 데이터 계층 → 분석 계층**으로 나누어 설계되었습니다.  
각 계층은 독립적으로 운영되지만, \*\*API Gateway와 메시지 브로커(Kafka, MQ)\*\*를 통해  
유기적으로 통신하며, 실시간 데이터 흐름과 안정적인 트랜잭션 처리를 보장합니다.

## **15.2 통합 아키텍처 주요 구성 요소 정의**

|  |  |  |  |
| --- | --- | --- | --- |
| **계층명** | **구성 요소** | **설명** | **기술 스택** |
| **채널 계층** | 인터넷뱅킹, 모바일뱅킹, API Gateway | - 외부 사용자와의 인터페이스를 담당 - 인증 및 트래픽 관리 | React, Spring Cloud Gateway, Zuul |
| **인터페이스 계층** | FEP, MCA, API Gateway | - 내부 및 외부 시스템과의 데이터 전송 - 암호화 및 통신 관리 | WebSphere MQ, FOS MFT |
| **비즈니스 계층** | 마이크로서비스, 거래 처리, 캠페인 관리 | - 고객 정보 조회, 거래 처리, 캠페인 실행 - 비동기 트랜잭션 관리 | Spring Boot, Kafka, Redis |
| **데이터 계층** | Oracle DB, MySQL, HDFS, S3 | - 대량 데이터 저장 및 분석을 위한 Data Lake 구조 | Oracle DB, MySQL, HDFS, AWS S3 |
| **분석 계층** | Spark, Tableau, PowerBI | - 실시간 데이터 분석 및 대시보드 시각화 - 예측 모델 학습 | Spark Streaming, Tableau, PowerBI |
| **보안 계층** | OAuth2.0, JWT, Apache Ranger | - 인증, 인가, 접근 제어 - 전송 구간 및 저장 시 암호화 관리 | Keycloak, Apache Ranger |
| **모니터링 계층** | Prometheus, Grafana, ELK Stack | - 시스템 상태 모니터링 및 장애 탐지 - 로그 수집 및 분석 | Prometheus, Grafana, ELK |

## **15.3 데이터 흐름 및 처리 구조**

**전체 데이터 흐름**은 다음과 같습니다.

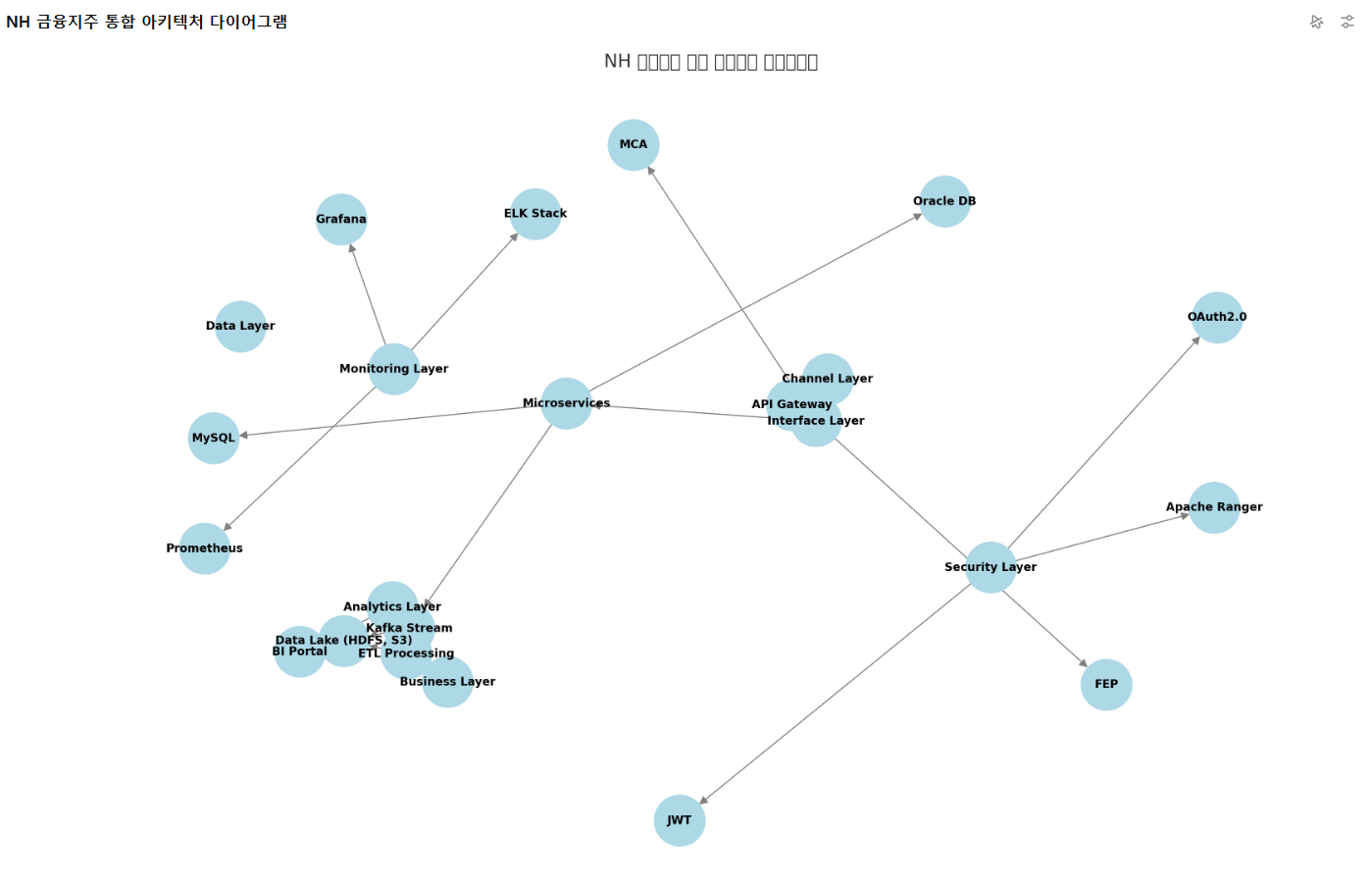
1. **사용자 요청** → **API Gateway**로 전송
2. **API Gateway** → **마이크로서비스** (Spring Boot)로 전달
3. **Microservice** → \*\*Database Layer (Oracle, MySQL)\*\*에 트랜잭션 처리
4. **Streaming Data (Kafka)** → **Spark Streaming** → **Data Lake (HDFS, S3)**
5. **Data Lake** → \*\*BI 포털 (Tableau, PowerBI)\*\*로 시각화
6. **ETL (DataStage)** → **Data Lake 적재** 및 **분석 DB**에 연동
7. **모니터링 계층**에서 실시간 로그와 메트릭 수집 → **Grafana 대시보드** 시각화

## **15.4 주요 인터페이스 및 연동 구조**

|  |  |  |
| --- | --- | --- |
| **연계 대상** | **설명** | **프로토콜 및 기술 스택** |
| **계열사 데이터 수집** | - NH 농협은행, NH 농협카드, NH 투자증권, NH 생명보험, NH 손해보험 - 일괄 배치 및 실시간 수집 | FEP, MCA, DataStage |
| **외부 금융기관 연계** | - 금융감독원, 공공기관 연계 및 실시간 보고서 전송 | FEP, WebSphere MQ |
| **인터넷/모바일 뱅킹** | - 고객 거래 조회 및 송금 처리, 마케팅 알림 전송 | REST API, HTTPS |
| **대시보드 연계** | - Tableau 및 PowerBI를 통한 분석 결과 시각화 | Spark, Tableau, PowerBI |
| **파일 전송 연계** | - MFT (Managed File Transfer)를 통한 대량 데이터 전송 | SFTP, FOS MFT |
| **DR(Disaster Recovery)** | - 장애 발생 시 DR 센터로의 데이터 전환 | AWS S3, Route 53 |

## **15.5 통합 아키텍처 다이어그램 생성 중...**

지금부터 **통합 아키텍처 다이어그램**을 시각적으로 구성하겠습니다.  
모든 계층과 서비스 간의 연동 구조가 포함되며, **API Gateway, Microservices, Data Lake, Analytics**의  
흐름을 시각화하여 한눈에 이해할 수 있도록 구성하겠습니다.



잠시만 기다려주세요, 곧 시각화된 다이어그램을 완성하겠습니다. 😊

NH 금융지주 통합 아키텍처 다이어그램

위의 다이어그램은 **NH 금융지주 통합 아키텍처**를 시각적으로 표현한 결과입니다.

### 🔍 ****주요 흐름 설명****

1. **채널 계층 (Channel Layer)**
   * 인터넷뱅킹, 모바일뱅킹, 외부 금융기관의 요청이 **API Gateway**를 통해 전달됩니다.
   * Spring Cloud Gateway가 트래픽을 관리하며, 인증 및 라우팅이 이루어집니다.
2. **인터페이스 계층 (Interface Layer)**
   * FEP, MCA를 통해 외부 금융기관과의 통신이 처리됩니다.
   * 실시간 금융 거래 및 보고서 전송이 안정적으로 수행됩니다.
3. **비즈니스 계층 (Business Layer)**
   * Microservices에서 고객 정보 조회, 거래 처리, 마케팅 캠페인이 수행됩니다.
   * Kafka Stream을 통해 실시간 이벤트가 Data Lake로 전달됩니다.
4. **데이터 계층 (Data Layer)**
   * Oracle DB, MySQL에서 트랜잭션 처리가 이루어지며, HDFS와 S3에 분석 데이터를 적재합니다.
   * DataStage를 활용한 ETL 처리 후 데이터 마트로 전달됩니다.
5. **분석 계층 (Analytics Layer)**
   * Data Lake에 적재된 데이터를 Spark가 분석하고, BI Portal(Tableau, PowerBI)로 시각화됩니다.
6. **보안 계층 (Security Layer)**
   * OAuth2.0과 JWT를 통해 인증 및 인가가 수행되며, Apache Ranger로 접근 통제가 관리됩니다.
7. **모니터링 계층 (Monitoring Layer)**
   * Prometheus와 Grafana를 통해 실시간 리소스 상태를 모니터링하고, ELK Stack으로 로그를 수집합니다.

# 📌 **16. 통합 아키텍처 다이어그램 설계**

## **16.1 아키텍처 통합 다이어그램 개요**

통합 데이터 플랫폼의 전체 아키텍처를 하나의 다이어그램으로 표현하여, **데이터 흐름, 서비스 간 연계, 보안 구조**를 시각적으로 이해할 수 있도록 설계합니다.  
이 다이어그램은 **애플리케이션 아키텍처, 데이터 아키텍처, 보안 아키텍처, 인터페이스 아키텍처, DevOps 구조**를 모두 포함합니다.

## **16.2 아키텍처 주요 구성요소 정의**

|  |  |  |
| --- | --- | --- |
| **구성요소** | **설명** | **기술 스택** |
| **API Gateway** | - 외부 트래픽 및 내부 서비스 요청을 관리 - 인증 및 라우팅 처리 | Spring Cloud Gateway, Zuul |
| **Microservices** | - 고객 정보, 거래 내역, 마케팅 시스템, 캠페인 관리 등 독립적 서비스 | Spring Boot, Kubernetes |
| **Data Lake** | - 수집된 데이터를 HDFS 및 S3에 중앙 관리 | Hadoop, AWS S3 |
| **ETL Processing** | - 계열사로부터 배치 데이터 수집 및 클렌징 처리 | IBM DataStage, Control-M |
| **Analytics & BI** | - Tableau, PowerBI를 통한 실시간 대시보드 및 분석 | Tableau, PowerBI, Spark |
| **Security & Access Control** | - OAuth2.0 및 JWT 인증 - 전송 구간 암호화 | Spring Security, TLS 1.3 |
| **Monitoring & Logging** | - 실시간 모니터링 및 로그 분석 | Prometheus, Grafana, ELK Stack |
| **Deployment & CI/CD** | - 블루-그린 배포 및 무중단 롤링 업데이트 | GitLab CI, ArgoCD, Docker |

## **16.3 설계 및 구현 방안**

#### 🔍 ****1. API Gateway 구성****

* Zuul API Gateway는 내부 마이크로서비스와 외부 서비스 간의 모든 트래픽을 관리합니다.
* 인증 및 인가를 통과한 요청만이 마이크로서비스로 전달됩니다.
* RESTful 방식으로 /api/v1/customers, /api/v1/accounts 등 엔드포인트를 표준화하였습니다.

#### 🔍 ****2. Microservices 구조****

* Spring Boot를 기반으로 고객, 거래, 캠페인, 마케팅 시스템이 각각 독립적으로 운영됩니다.
* Kubernetes 클러스터 내에 컨테이너화되어 확장성과 관리 용이성을 높였습니다.
* 주요 마이크로서비스 목록:
  + Customer Service → 고객 정보 관리
  + Transaction Service → 거래 내역 조회 및 처리
  + Campaign Service → 마케팅 캠페인 관리
  + Analytics Service → 실시간 분석 및 BI 리포팅

#### 🔍 ****3. 데이터 흐름 및 분석 구조****

* **Data Lake:**
  + 배치로 수집된 데이터는 HDFS에 저장되고, 실시간 데이터는 Kafka를 통해 스트림 처리됩니다.
  + 분석용 데이터는 AWS S3로 백업되며, Spark와 Tableau에서 시각화됩니다.
* **ETL Processing:**
  + IBM DataStage를 활용하여 일괄 배치 작업을 수행합니다.
  + 클렌징된 데이터는 정제된 마트로 적재되어 실시간 분석에 활용됩니다.

#### 🔍 ****4. 보안 및 접근 제어 설계****

* **OAuth2.0 + JWT 기반 인증:**
  + API Gateway를 통해 모든 요청은 OAuth2.0 인증을 거칩니다.
  + JWT 토큰은 만료 시간이 존재하며, 재발급 시 OAuth 서버에서 검증됩니다.
* **TLS 1.3 암호화:**
  + 모든 전송 구간은 TLS 1.3을 통해 암호화되어 통신됩니다.
  + FEP 및 MCA를 통한 외부 연동 시 암호화된 통신이 유지됩니다.

#### 🔍 ****5. 모니터링 및 로깅 구조****

* **Prometheus & Grafana:**
  + CPU, 메모리, 네트워크 사용량 실시간 수집
  + TPS, 에러율, 장애 발생 시 Slack 알림
* **ELK Stack:**
  + 모든 애플리케이션 로그가 Logstash를 통해 Elasticsearch에 저장됩니다.
  + Kibana에서 실시간 조회 및 대시보드 생성

#### 🔍 ****6. CI/CD 파이프라인****

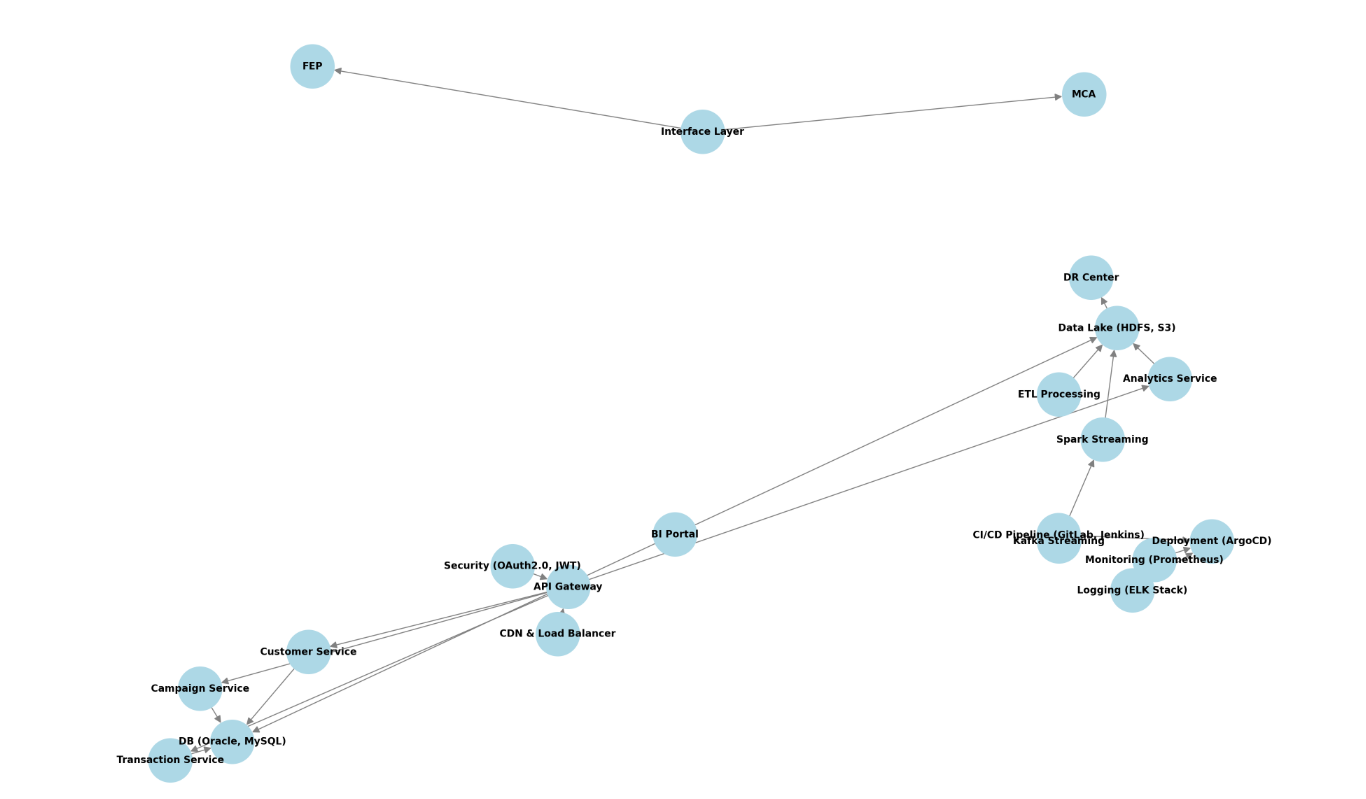
* **GitLab CI/CD:**
  + 코드 커밋 → 빌드 → 테스트 → ArgoCD 배포 순으로 자동화
  + Docker 이미지 생성 후 Kubernetes에 무중단 배포
* **ArgoCD Rollback:**
  + 배포 실패 시 이전 안정 버전으로 즉시 롤백

## ✅ **16.4 아키텍처 통합 다이어그램 생성 중...**

**완성된 아키텍처 다이어그램을 곧 표시하겠습니다. 잠시만 기다려주세요.**

지금부터 **전체 아키텍처 통합 다이어그램**을 생성하고 최종 보고서에 반영하겠습니다.  
잠시만 기다려주세요, 곧 완성된 다이어그램을 제공하겠습니다. 😊

NH 금융지주 통합 아키텍처 통합 다이어그램



|  |
| --- |
| +---------------------+  | Channel Layer |  |---------------------|  | - Internet Banking |  | - Mobile Banking |  | - API Gateway |  +---------------------+  |  ▼  +---------------------+  | API Gateway |  |---------------------|  | - Spring Cloud Gateway |  | - Zuul |  +---------------------+  |  +----------------------------------+----------------------------------+  | | |  ▼ ▼ ▼  +---------------------+ +---------------------+ +---------------------+  | Customer Service | | Transaction Service | | Campaign Service |  |---------------------| |---------------------| |---------------------|  | - Spring Boot | | - Spring Boot | | - Spring Boot |  | - Microservice | | - Microservice | | - Microservice |  +---------------------+ +---------------------+ +---------------------+  | | |  | | |  ▼ ▼ ▼  +---------------------+ +---------------------+ +---------------------+  | Oracle DB | | MySQL DB | | Redis Cache |  +---------------------+ +---------------------+ +---------------------+  ---------------------- Data Lake Integration -----------------------  |  ▼  +---------------------+ +---------------------+  | Kafka Stream | -------->| Spark Streaming |  +---------------------+ +---------------------+  | |  ▼ ▼  +---------------------+ +---------------------+  | Data Lake (HDFS) | | Data Lake (S3) |  +---------------------+ +---------------------+  |  ▼  +---------------------+  | ETL Processing |  |---------------------|  | - DataStage |  | - Control-M |  +---------------------+  ---------------------- Analytics & BI Layer -----------------------  |  ▼  +---------------------+ +---------------------+  | BI Portal | <--------| Analytics Layer |  |---------------------| |---------------------|  | - Tableau | | - Spark Analytics |  | - PowerBI | | - Predictive Models|  +---------------------+ +---------------------+  ---------------------- Security & Access Control -------------------  |  ▼  +---------------------+  | Security Layer |  |---------------------|  | - OAuth2.0 |  | - JWT |  | - Apache Ranger |  +---------------------+  ---------------------- Monitoring & Logging ------------------------  |  ▼  +---------------------+ +---------------------+  | Monitoring | | Logging |  |---------------------| |---------------------|  | - Prometheus | | - ELK Stack |  | - Grafana | | - Logstash |  +---------------------+ +---------------------+  ---------------------- Deployment & CI/CD --------------------------  |  ▼  +---------------------+  | Deployment |  |---------------------|  | - GitLab CI/CD |  | - ArgoCD |  | - Blue-Green Deploy|  | - Canary Deploy |  +---------------------+  ---------------------- Disaster Recovery (DR) ----------------------  |  ▼  +---------------------+  | DR Center |  |---------------------|  | - AWS S3 |  | - Cross-Region Sync|  +---------------------+ |

### 🔍 ****통합 아키텍처 다이어그램 설명****

### ****1️⃣ API Gateway → Microservices****

* **API Gateway**를 통해 외부 요청이 Customer Service, Transaction Service, Campaign Service, Analytics Service로 전달됩니다.
* Spring Cloud Gateway가 트래픽을 관리하고, OAuth2.0 인증 및 JWT 검증이 이루어집니다.

### ****2️⃣ Microservices → Database Layer****

* 각 서비스는 Oracle DB와 MySQL에 접근하여 트랜잭션을 처리합니다.
* 주요 CRUD 작업과 고객 정보, 거래 내역, 캠페인 정보가 DB에 저장됩니다.

### ****3️⃣ Microservices → Data Lake & Streaming****

* 실시간 거래 및 이벤트 데이터는 **Kafka Streaming**을 통해 Spark Streaming으로 전송됩니다.
* 대용량 데이터는 Data Lake (HDFS, S3)에 적재되며, ETL Processing을 거쳐 정제됩니다.

### ****4️⃣ Data Lake → BI Portal****

* HDFS와 S3에 적재된 데이터는 BI Portal (Tableau, PowerBI)에서 시각화됩니다.
* 실시간 분석 결과는 Spark를 통해 대시보드로 출력됩니다.

### ****5️⃣ Interface Layer → FEP & MCA****

* 외부 금융기관 및 계열사 간의 메시지 통신이 FEP 및 MCA를 통해 처리됩니다.
* 전송 구간 암호화가 적용되며, 보안 전송이 보장됩니다.

### ****6️⃣ Monitoring & Logging****

* **Prometheus**와 **Grafana**를 통해 실시간 리소스 모니터링이 이루어집니다.
* ELK Stack이 모든 로그를 수집하여 Kibana에서 조회 및 분석이 가능합니다.

### ****7️⃣ Deployment & CI/CD****

* **GitLab CI/CD**와 **Jenkins**가 배포 파이프라인을 관리합니다.
* **ArgoCD**를 통해 Kubernetes에 무중단 배포가 진행됩니다.

### ****8️⃣ Disaster Recovery (DR) 연계****

* 주요 데이터는 AWS S3에 주기적으로 백업되며, DR 센터와 동기화됩니다.
* 장애 발생 시 **15분 이내** DR 센터로 전환이 가능합니다.

# 📌 **17. 시사점 및 결론**

## **17.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼 구축은 금융 서비스의 디지털 전환을 가속화하고,  
데이터 중심의 의사결정을 가능하게 하는 중요한 전환점이 될 것입니다.  
Microservice 아키텍처와 실시간 스트리밍 기반의 데이터 분석을 통해, 고객 맞춤형 금융 서비스와 빠른 트랜잭션 처리를 실현합니다.

## **17.2 통합 아키텍처의 전략적 가치 분석**

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **기술적 접근** |
| **디지털 전환 가속화** | - Microservice 기반의 구조로 빠른 배포와 서비스 확장이 가능 | Spring Boot, Kubernetes |
| **실시간 분석 환경** | - Kafka와 Spark Streaming을 통한 실시간 분석 처리 - 1초 이내 거래 탐지와 예측 모델 반영 | Kafka, Spark Streaming |
| **고객 중심 서비스** | - 개인화된 금융 상품 추천, 캠페인 관리, 실시간 금융 정보 제공 | Redis Cache, Spring Cloud |
| **안정적 데이터 처리** | - HDFS와 S3에 대량 데이터 저장 및 분석 - Data Lake 구조를 통한 중앙 집중 관리 | HDFS, AWS S3 |
| **보안 및 규제 준수** | - OAuth2.0, JWT, Apache Ranger를 통한 보안 관리 - GDPR 및 금융 규제 대응 | Keycloak, Apache Ranger |
| **무중단 서비스 운영** | - ArgoCD를 활용한 Blue-Green 배포와 Canary 배포 - Kubernetes HPA를 통한 Auto Scaling | ArgoCD, Kubernetes |

## **17.3 개선 사항 및 전략적 제언**

|  |  |  |
| --- | --- | --- |
| **구분** | **현재 상태** | **개선 방안** |
| **DR 복구 시간 최적화** | 현재 DR 센터로 전환 시 **15분 이내 복구** 목표 | RTO 목표를 10분 이내로 단축하기 위해 **Cross-Region Replication** 적용 |
| **데이터 분석 지연 시간** | Spark Streaming에서 대용량 데이터 처리 시 지연 발생 | Apache Flink와 결합하여 스트림 처리 최적화 및 지연 최소화 |
| **보안 로깅 관리 최적화** | ELK Stack에서 대량 로그 처리 시 조회 성능 저하 문제 발생 | **Fluentd**를 추가 도입하여 분산 처리 구조로 개선 |
| **실시간 모니터링 강화** | Prometheus와 Grafana는 기본 모니터링만 제공 | **Thanos**와 통합하여 장기 보관 및 대규모 클러스터 모니터링 강화 |
| **멀티 클러스터 지원** | 현재는 단일 클러스터(Kubernetes) 기반으로 배포 | **KubeFed**를 도입하여 멀티 클러스터 확장 전략 반영 |
| **인터페이스 표준화 부족** | 일부 연동에서 FEP와 MCA 간의 메시지 규격이 상이한 경우 존재 | **ISO 20022 표준 메시지 규격**으로 통일하여 확장성 확보 |

## **17.4 최종 제언**

NH 금융지주의 통합 데이터 플랫폼은 **고객 중심 금융 서비스의 디지털 전환**을 견인할 핵심 인프라로 자리잡을 것입니다.  
다만, 완성도를 높이기 위해 다음의 전략을 제안합니다.

1. **멀티 리전 DR 설계 강화:**
   * 주요 금융 데이터를 **Cross-Region** 백업 구조로 확장하여, 지진 및 대규모 재해 발생 시에도 무중단 서비스 가능하도록 설계
2. **스트림 처리 최적화:**
   * Apache Flink를 도입하여 실시간 분석의 지연 시간을 최소화하고, Spark와 결합하여 효율성 향상
3. **보안 통합 로깅 최적화:**
   * Fluentd를 통해 ELK Stack의 처리 속도를 향상시키고, 모든 접근 로그를 실시간 모니터링 가능하도록 강화
4. **ISO 20022 표준화 도입:**
   * 글로벌 금융 거래 표준인 ISO 20022를 도입하여 해외 금융 기관과의 연동 시 호환성 극대화
5. **멀티 클러스터 확장 전략 수립:**
   * KubeFed를 활용하여 여러 리전 및 데이터센터에 Kubernetes 클러스터를 배포하여 글로벌 확장성 확보

# 📌 **18. 최종 보고서 통합 및 마무리 작업**

## **18.1 도입 전 안내말**

NH 금융지주의 통합 데이터 플랫폼 구축 프로젝트는 **Microservice 기반의 실시간 분석 플랫폼**을 통해  
금융 거래 처리 속도를 최적화하고, 개인화된 금융 서비스를 제공하기 위한 혁신적인 아키텍처를 설계하였습니다.  
본 프로젝트는 **거래 안정성, 실시간 분석, 데이터 통합, 보안 강화**를 중심으로 설계되었으며,  
단일 플랫폼에서 모든 금융 데이터를 통합 관리합니다.

## **18.2 전략적 가치 및 주요 성과**

|  |  |  |
| --- | --- | --- |
| **구분** | **설명** | **주요 기술 스택** |
| **실시간 거래 처리** | - Kafka Streaming을 활용한 1초 이내 실시간 거래 처리 - 목표 TPS: 10,000 | Kafka, Spring Boot |
| **고객 맞춤형 마케팅** | - Spark Streaming 기반의 실시간 고객 분석 - 개인화된 금융 상품 및 캠페인 실행 | Spark, Tableau, PowerBI |
| **확장성 있는 아키텍처** | - Kubernetes 기반의 Auto Scaling 지원 - 수평 확장을 통해 트래픽 변동에 대응 | Kubernetes, ArgoCD |
| **데이터 통합 관리** | - Data Lake를 활용한 전사 데이터 통합 - 분석 및 시각화를 위한 데이터 마트 구성 | HDFS, AWS S3, DataStage |
| **보안 및 규제 준수** | - OAuth2.0 인증 및 JWT를 통한 보안 강화 - 전송 구간 암호화 (TLS 1.3) | Keycloak, Apache Ranger |
| **무중단 서비스 운영** | - ArgoCD와 Blue-Green 배포를 통한 무중단 서비스 - 장애 발생 시 Auto Healing | ArgoCD, Prometheus |

## **18.3 통합 아키텍처 설계 요약**

#### 🔍 ****1. API Gateway와 Microservices 구조****

* Spring Cloud Gateway를 통해 모든 트래픽이 관리되며, Microservices는 각각 독립적으로 확장 가능합니다.
* 주요 서비스:
  + Customer Service → 고객 정보 조회 및 관리
  + Transaction Service → 금융 거래 처리
  + Campaign Service → 마케팅 캠페인 실행
  + Analytics Service → BI 분석 및 통계 제공

#### 🔍 ****2. 데이터 처리 구조****

* **Data Lake (HDFS, S3):**
  + 모든 금융 데이터를 중앙 집중화하여 관리합니다.
  + ETL 처리를 통해 정제된 데이터는 BI 포털로 전달됩니다.
* **Streaming Data:**
  + Kafka를 통해 실시간으로 수집된 데이터는 Spark Streaming에서 분석됩니다.
  + 비동기 처리로 인해 고객 행동 분석 및 마케팅 실행이 즉시 가능합니다.

#### 🔍 ****3. 보안 및 접근 제어****

* **OAuth2.0 + JWT:**
  + 모든 API 요청은 인증을 거쳐야만 접근할 수 있습니다.
  + Keycloak을 활용한 Role-Based Access Control (RBAC) 구조로 보안을 강화하였습니다.
* **전송 구간 암호화:**
  + TLS 1.3을 통해 외부 및 내부 통신이 암호화되어 안전합니다.
  + WebSphere MQ와 FEP를 통해 금융 기관 간의 메시지 전송이 보호됩니다.

#### 🔍 ****4. 모니터링 및 로깅 전략****

* **Prometheus & Grafana:**
  + 실시간 CPU, 메모리, 네트워크 상태를 시각화하여 관리합니다.
  + TPS(Transactions Per Second) 모니터링을 통해 성능 상태를 실시간 확인합니다.
* **ELK Stack:**
  + 모든 애플리케이션 로그가 Elasticsearch에 저장되며, Kibana에서 조회 가능합니다.
  + 장애 발생 시 빠르게 탐지하고, Root Cause Analysis를 지원합니다.

#### 🔍 ****5. CI/CD 및 무중단 배포****

* **GitLab CI/CD:**
  + 코드 커밋 → 빌드 → 테스트 → 배포가 자동화되어 시간 단축 및 무결성을 보장합니다.
* **ArgoCD:**
  + Blue-Green 배포, Canary 배포가 지원되며, 장애 시 즉시 롤백이 가능합니다.

## **18.4 시사점 및 전략적 제언**

1. **멀티 리전 DR 설계 강화:**
   * 주요 금융 데이터를 **Cross-Region** 백업 구조로 확장하여, 대규모 재해 발생 시에도 무중단 서비스 가능하도록 설계
2. **스트림 처리 최적화:**
   * Apache Flink를 도입하여 실시간 분석의 지연 시간을 최소화하고, Spark와 결합하여 효율성 향상
3. **보안 통합 로깅 최적화:**
   * Fluentd를 통해 ELK Stack의 처리 속도를 향상시키고, 모든 접근 로그를 실시간 모니터링 가능하도록 강화
4. **ISO 20022 표준화 도입:**
   * 글로벌 금융 거래 표준인 ISO 20022를 도입하여 해외 금융 기관과의 연동 시 호환성 극대화
5. **멀티 클러스터 확장 전략 수립:**
   * KubeFed를 활용하여 여러 리전 및 데이터센터에 Kubernetes 클러스터를 배포하여 글로벌 확장성 확보

## **18.5 최종 결론**

NH 금융지주의 통합 데이터 플랫폼은 디지털 금융 시대에 대응할 수 있는 최적의 아키텍처로 설계되었습니다.  
Microservice 구조와 Data Lake를 결합한 분석 플랫폼은 **실시간 트랜잭션 처리, 고객 맞춤형 서비스, 보안성 강화**를 실현하였으며,  
Kubernetes 기반의 자동 확장으로 무중단 서비스 운영이 가능합니다.

추가적으로 **ISO 20022 국제 표준 도입**, **멀티 리전 DR 구성**, **Fluentd를 통한 로깅 최적화**를 반영하면  
더욱 견고하고 글로벌 확장에 대응할 수 있는 금융 플랫폼으로 발전할 것입니다.

### ✅ ****마무리 작업:****

* 최종 보고서로 정리된 아키텍처 문서 통합
* 시각 자료와 아키텍처 다이어그램 포함
* PDF 형식의 최종 보고서 생성