**📌 아키텍처 제안서 작성 방안**

지금 제공해주신 정보는 매우 상세하고 체계적이기 때문에, 이를 기반으로 **아키텍처 제안서**를 작성할 수 있도록 목차를 구성하겠습니다.  
제안서는 크게 두 가지 목표를 가지고 진행됩니다:

1. **NH 계열사 통합 데이터 플랫폼 구축을 위한 아키텍처 설계**
2. **설계 및 구현 방안의 구체화**

**📑 아키텍처 제안서 목차 구성안**

**00. 프로젝트 개요**

* 00.1 대상 계열사 정의
* 00.2 시스템별 주요 기능 정의
  + 데이터 수집 및 저장
  + 데이터 분석 및 활용
  + 은행 빅스퀘어
  + 마케팅 시스템

**01. 기술 아키텍처 (TA)**

* 01.1 인프라 환경 정의
* 01.2 HW, SW, 인력 계획
* 01.3 예산 계획
* 01.4 스토리지 및 임시 보관 영역 설계
* **01.5 설계 및 구현 방안**

**02. 애플리케이션 아키텍처 (AA)**

* 02.1 시스템 연계 및 수집 프로세스
* 02.2 개인화 마케팅 분석 및 실행 구조
* 02.3 데이터 전송 및 암호화 설계
* 02.4 통합고객번호 채번 및 관리
* 02.5 배치 로깅 및 실시간 연계 서비스
* 02.6 금융지주 OLAP 환경 설계
* **02.7 설계 및 구현 방안**

**03. 비즈니스 아키텍처 (BA)**

* 03.1 개인화 마케팅 서비스 설계
* 03.2 데이터 수집 및 활용 프로세스
* 03.3 데이터 전처리 및 임시 보관
* 03.4 목적별 마트 구성 및 고객번호 관리
* 03.5 데이터 분석 및 모델 운영
* **03.6 설계 및 구현 방안**

**04. 데이터 아키텍처 (DA)**

* 04.1 데이터 주제 영역 정의
* 04.2 원천 데이터 적재 및 목적별 마트 구성
* 04.3 메타 데이터 관리 및 조회
* 04.4 분석용 목적 DB 구성
* 04.5 데이터 보관 주기 및 파기 정책
* **04.6 설계 및 구현 방안**

**05. 보안 아키텍처 (SA)**

* 05.1 제3자 동의 기반 통합 데이터 인증
* 05.2 전송구간 암호화
* 05.3 전송파일 암호화
* 05.4 개인정보 처리 시 암호화/마스킹/파기
* 05.5 계정 및 권한 관리 체계
* **05.6 설계 및 구현 방안**

**06. 인터페이스 아키텍처 (IA)**

* 06.1 계열사 데이터 수집 인터페이스 설계
* 06.2 올원뱅크 행동 데이터 수집
* 06.3 데이터 연계 표준 및 프로토콜 설계
* 06.4 분석 플랫폼과의 연계 설계
* **06.5 설계 및 구현 방안**

**07. 기대효과 및 전략적 가치**

* 07.1 비용 절감 효과
* 07.2 통합 데이터 기반 의사결정 지원
* 07.3 분석 모델을 통한 마케팅 효과 증대

**08. 아키텍처 주요 정보**

* 08.1 도입 시스템 정의
  + 신규 도입
  + 기존 시스템 활용
* 08.2 인터페이스 구성도
* 08.3 데이터 주제 영역 정의
* 08.4 도입 솔루션 목록

**🔄 진행 순서**

1. **00. 프로젝트 개요**부터 시작하여 단계적으로 작성합니다.
2. 각 항목에 대해 **설계 및 구현 방안**을 구체적으로 정의합니다.
3. **실사례와 코드 예시**를 포함하여 실질적인 이해를 돕습니다.
4. **문서의 일관성**을 유지하며, 통합된 목차로 관리합니다.

\*\*지금부터 "00. 프로젝트 개요"\*\*부터 시작하겠습니다.

* 대상 계열사 정의
* 시스템별 주요 기능 정의

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 시스템 구성 및 기능 설명
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 00. 프로젝트 개요**

**00.1 대상 계열사 정의**

NH 금융지주 그룹 내 금융 및 보험사를 포함한 다양한 계열사가 통합 데이터 플랫폼의 구축 대상입니다.

* **대상 계열사:**
  + NH농협은행
  + NH농협카드
  + NH투자증권
  + NH생명보험
  + NH손해보험

이들 계열사의 데이터를 중앙 집중화하고, 통합된 분석 및 관리 플랫폼을 구축함으로써, 전사적 데이터 활용을 극대화하고 금융 서비스의 확장성을 확보하는 것을 목표로 합니다.

**00.2 시스템별 주요 기능 정의**

NH 금융 계열사의 데이터 수집, 처리, 분석 및 마케팅 활용을 위한 주요 기능을 정의합니다.

|  |  |  |
| --- | --- | --- |
| 시스템 분류 | 기능 설명 | 비고 |
| 데이터 수집 및 저장 | - 계열사 데이터 배치 수집 - 올원뱅크 행동 데이터 수집 - 표준 인터페이스 적용 - 단기 보관용 임시 영역 구성 | 배치 수행, 실시간 수집 |
| 데이터 처리 | - 파기 대상 데이터 삭제 - 배치 작업 이력 로깅 - 개인정보 포함 데이터 암호화/마스킹/파기 - 실시간/배치 데이터 연계 | ETL, EAI 연동 |
| 데이터 저장 | - 원천 데이터 적재 및 마트 개발 - 통합고객번호 채번 - 데이터 저장 레이어 표준 수립 - 분석 목적 DB 구성 | OLAP, DW 연계 |
| 데이터 분석 및 활용 | - 통합 데이터 포털 제공 - 메타 데이터 조회 및 관리 - 데이터 품질 관리 - 계열사/사용자별 권한 관리 | BI 솔루션, OLAP |
| 은행 빅스퀘어 | - 쿼리 기반 데이터 분석 - ML/DL 모델 개발 - 데이터 분석 모델 운영 | MSTR, Spark 기반 |
| 마케팅 시스템 | - 캠페인 설정(타깃 마케팅, 우선순위 설정) - 실시간 EBM 지원 - 자동화 서비스 지원 - 개인화 마케팅 실행 | 실시간 분석 엔진 |
| 성과 분석 | - 과거 이력 관리 - 오퍼/시나리오 분석 - 패널 분석 결과 제공 | BI 포털과 연계 |

**00.3 설계 및 구현 방안**

**🔍 1. 데이터 수집 및 저장 설계**

* **배치 수집:**
  + ETL(DataStage) 기반으로 계열사 데이터를 정기적으로 수집
  + 수집된 데이터는 HDFS 및 RDBMS에 적재
* **실시간 수집:**
  + 표준 인터페이스(MCA, FEP)를 통한 실시간 메시지 수집
  + 올원뱅크 행동 데이터를 Kafka 스트림 처리
* **임시 보관 영역 설계:**
  + 수집된 원본 파일을 HDFS의 임시 스토리지에 24시간 보관 후 이동

**🔍 2. 데이터 처리 설계**

* **이력 로깅:**
  + 배치 작업의 성공/실패 여부를 ELK(ElasticSearch, Logstash, Kibana)로 실시간 모니터링
  + 실패 시 즉시 Rollback 및 알림 전송
* **암호화 및 마스킹:**
  + 개인정보는 AES-256 및 RSA로 암호화 처리
  + 주요 금융 정보는 마스킹 규칙에 따라 일부만 노출
* **파기 정책 적용:**
  + 데이터 보존 주기 만료 시 자동 파기
  + 파기 이력은 5년간 보관

**🔍 3. 데이터 저장 및 활용 설계**

* **저장 레이어 설계:**
  + **원본 레이어:** 수집된 원본 데이터를 저장
  + **정제/통합 레이어:** ETL 후 클렌징 및 통합된 데이터 저장
  + **요약/분석 레이어:** OLAP, 분석 마트, BI 리포팅 용도로 구성
* **통합고객번호 관리:**
  + 고객의 모든 거래 정보를 통합하여 Master ID 생성
  + Redis를 활용한 분산 캐싱 처리
* **분석 목적 DB 구성:**
  + ML/DL 모델 학습 및 실시간 분석을 위한 전용 DB 구축

**🔍 4. 분석 및 마케팅 시스템 설계**

* **OLAP 환경:**
  + OLAP 큐브를 활용한 다차원 분석
  + Tableau 및 PowerBI를 활용한 리포팅
* **빅스퀘어 분석:**
  + Spark 기반의 분산 처리와 ML 모델 학습
  + 실시간 스트림 분석을 통해 개인화 마케팅 대응
* **마케팅 시스템 설계:**
  + 실시간 타깃팅, 우선순위 설정, 피로도 관리
  + Kafka 스트림으로 이벤트 기반 실시간 캠페인 실행

**🔍 5. 설계 다이어그램 예시**

* **데이터 수집 및 처리 아키텍처:**
  + 계열사 → ETL (DataStage) → 통합 DB
  + 실시간 연계: Kafka → Spark Streaming → Data Lake
* **분석 및 마케팅 구조:**
  + Data Lake → OLAP Cube → BI Report
  + Kafka → 실시간 마케팅 엔진 → 타겟 고객 알림

**✅ 다음 진행 계획:**

이제 \*\*01. 기술 아키텍처 (TA)\*\*부터 작성하겠습니다.

* 인프라 환경 정의
* HW, SW, 인력 계획
* 예산 계획
* 스토리지 및 임시 보관 영역 설계
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 인프라 구성도 및 설명
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 01. 기술 아키텍처 (TA)**

**01.1 인프라 환경 정의**

NH 금융지주 그룹의 통합 데이터 플랫폼을 위한 기술 아키텍처는 **NH농협은행의 인프라를 중심으로 구축**됩니다.  
주요 목표는 대량의 금융 데이터를 안정적으로 수집, 처리, 저장하고, 이를 실시간 분석과 개인화 마케팅에 활용할 수 있도록 하는 것입니다.

|  |  |
| --- | --- |
| 구성 요소 | 상세 설명 |
| 인프라 구축 범위 | - 데이터 수집, 저장, 처리, 분석 및 마케팅 시스템을 포함한 전사 플랫폼 |
| 주요 기술 스택 | - Spring Boot, Oracle DB, DataStage, Apache Kafka |
| 클라우드 환경 | - NH농협은행의 PaaS 및 IaaS 기반 환경 활용 |
| 배치 처리 환경 | - DataStage ETL 처리, Control-M 스케줄링 관리 |
| 실시간 처리 환경 | - Apache Kafka 스트림 처리 |
| 보안 및 접근 제어 | - SSO, OAuth2.0, JWT 기반 인증 체계 구축 |
| 스토리지 구조 | - HDFS와 Oracle DB를 통한 이중화 저장 |
| 가용성 및 확장성 | - Active-Standby 이중화, 로드 밸런싱을 통한 무중단 서비스 |

**01.2 HW, SW, 인력 계획**

인프라를 안정적이고 확장성 있게 유지하기 위해 필요한 하드웨어(HW), 소프트웨어(SW), 인력 구성을 정의합니다.

|  |  |
| --- | --- |
| 구성 요소 | 스펙 및 요구 사항 |
| 서버 노드 수 | 8대 (2대 Active-Standby, 6대 분석 및 배치 처리 전용) |
| CPU | 2 x 8 Core per node |
| 메모리 | 256 GB per node |
| 스토리지 | HDFS: 10TB / Oracle DB: 3TB |
| 네트워크 대역폭 | 10Gbps 전용 라인 |
| DBMS | Oracle HEX, MySQL |
| ETL 솔루션 | IBM DataStage |
| 분석 솔루션 | Tableau, PowerBI, Spark ML |
| 통합 관리 도구 | Prometheus, Grafana, ELK Stack (ElasticSearch, Logstash, Kibana) |
| 필요 인력 | 개발자(5명), 데이터 분석가(3명), 인프라 엔지니어(2명), DevOps(2명) |

**01.3 예산 계획**

인프라 구축 및 유지보수에 필요한 예산을 추산합니다.

|  |  |  |
| --- | --- | --- |
| 항목 | 내용 | 예산 (억 원) |
| HW 구매 비용 | 서버, 스토리지, 네트워크 장비 | 5억 |
| SW 라이선스 비용 | Oracle HEX, DataStage, MSTR | 3억 |
| 클라우드 사용 비용 | PaaS, IaaS, Storage | 2억 |
| 운영 유지보수 비용 | 모니터링, 백업, 장애 복구, 유지보수 | 1억 |
| 인력 비용 | 개발, 분석, 인프라, DevOps | 4억 |
| 총 예산 | **13억 원** | **13억** |

**01.4 스토리지 및 임시 보관 영역 설계**

금융 데이터를 안정적으로 저장하고, 분석 목적으로 활용하기 위한 스토리지 설계입니다.

|  |  |  |
| --- | --- | --- |
| 영역 | 구성 설명 | 기술 스택 |
| 임시 보관 영역 | 수집된 데이터가 정제되기 전 임시로 저장되는 공간 | HDFS, Local Disk |
| 원본 저장 영역 | 계열사에서 수집된 원본 데이터를 보존하는 공간 | Oracle HEX |
| 정제/통합 영역 | ETL을 통해 처리된 정제된 데이터 저장 | MySQL, Oracle |
| 요약/분석 영역 | OLAP, BI 분석을 위한 데이터 마트 구성 | HDFS, Spark ML |
| 백업 및 복구 설계 | 장애 발생 시 스냅샷 및 로그를 통해 복구 | SnapShot, Point-in-Time Recovery |
| 데이터 보존 기간 | 금융 규정에 따른 5년간 보관 후 파기 | S3 Glacier |

**01.5 설계 및 구현 방안**

**🔍 1. 인프라 구축 방안**

* **Active-Standby 이중화:**
  + 데이터센터 간 Active-Standby 구조로 이중화 구성
  + 장애 시 자동으로 Standby 노드로 Failover
* **PaaS 및 IaaS 활용:**
  + NH농협은행의 PaaS 환경을 기반으로 마이크로서비스 배포
  + IaaS는 ETL 처리 및 OLAP 분석 전용으로 활용
* **네트워크 구조 설계:**
  + 10Gbps 전용 라인으로 내부 통신 최적화
  + 로드 밸런서 (Nginx, HAProxy) 적용

**🔍 2. 데이터 처리 및 분석 설계**

* **DataStage를 통한 ETL 처리:**
  + 계열사로부터 수집된 데이터를 HDFS 및 Oracle DB에 적재
  + 중복 데이터 필터링, 데이터 정규화 처리
* **OLAP 분석 환경:**
  + Tableau 및 PowerBI를 통해 실시간 대시보드 구성
  + Spark ML을 활용한 ML/DL 모델 학습
* **BI 포털 구성:**
  + 통합 포털에서 분석 리포트를 실시간 조회
  + 금융지주 전체에 대한 데이터 시각화 제공

**🔍 3. 보안 및 접근 통제 설계**

* **SSO 기반 인증:**
  + 모든 사용자 접근 시 SSO를 통해 통합 인증
  + OAuth2.0과 JWT를 활용한 보안 처리
* **네트워크 보안:**
  + 모든 통신은 HTTPS (TLS 1.3)로 암호화
  + 전송 구간은 VPN을 통해 암호화 전송

**🔍 4. 설계 다이어그램 예시**

* **인프라 구성도:**
  + PaaS (Spring Boot, Tomcat) → IaaS (ETL, OLAP) → Data Lake
  + 실시간 스트림: Kafka → Spark Streaming → HDFS
* **네트워크 아키텍처:**
  + Load Balancer → API Gateway → Microservice Cluster

**✅ 다음 진행 계획:**

이제 \*\*02. 애플리케이션 아키텍처 (AA)\*\*를 작성하겠습니다.

* 시스템 연계 및 수집 프로세스
* 개인화 마케팅 분석 및 실행 구조
* 데이터 전송 및 암호화 설계
* 통합고객번호 채번 및 관리
* 배치 로깅 및 실시간 연계 서비스
* 금융지주 OLAP 환경 설계
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 애플리케이션 구조 및 처리 프로세스
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 02. 애플리케이션 아키텍처 (AA)**

**02.1 시스템 연계 및 수집 프로세스**

NH 금융지주 그룹의 데이터를 통합 수집하여 표준화된 데이터 레이크(Data Lake)에 적재하고, 이를 다양한 분석 및 마케팅 시스템에서 활용할 수 있도록 설계합니다.

|  |  |  |
| --- | --- | --- |
| 프로세스명 | 설명 | 기술 스택 |
| 계열사 데이터 배치 수집 | - NH 농협은행, NH 농협카드, NH 투자증권 등에서 배치 수집 - DataStage ETL 도구 활용하여 적재 | DataStage, Control-M |
| 올원뱅크 행동 데이터 수집 | - 실시간 행동 데이터를 Kafka를 통해 스트림 처리 - 고객 행동에 대한 실시간 분석 가능 | Apache Kafka, Spark Streaming |
| 표준 인터페이스 적용 | - 표준 프로토콜(EAI, FEP)을 통한 데이터 수집 및 전송 - MCA 기반의 은행 내/외부 통신 | MCA, FEP, EAI |
| 임시 보관 영역 구성 | - 임시 스토리지에 24시간 동안 원본 파일 보관 후 이동 - 보관 정책에 따라 자동 파기 | HDFS, Oracle |

**02.2 세부 내용**

* **ETL 배치 수집:**
  + DataStage를 통해 계열사의 데이터를 매일 2회 수집
  + 금융 거래 내역, 고객 정보, 상품 정보 등 다양한 데이터 수집
* **Kafka 실시간 수집:**
  + Kafka 토픽을 활용하여 올원뱅크 사용자의 행동 정보를 스트림 처리
  + 고객이 금융 서비스를 이용할 때 실시간으로 분석에 반영
* **표준 인터페이스 연계:**
  + 은행 내/외부 시스템 간 표준 인터페이스(FEP, EAI) 통신
  + 예를 들어, 올원뱅크에서 발생한 결제 내역이 실시간으로 은행 시스템에 반영
* **단기 보관 영역 구성:**
  + HDFS를 기반으로 수집된 원본 파일을 24시간 동안 보관
  + 데이터 유실 방지를 위해 중복 백업이 활성화됨

**02.3 설계 및 구현 방안**

**🔍 1. 수집 설계 및 구현**

* **배치 수집 설계:**
  + DataStage ETL 도구를 활용하여 매일 두 차례 배치 수행
  + Control-M 스케줄러를 통해 정기적인 데이터 적재
  + 실패 시 자동 재시작 및 로깅 처리
* **실시간 수집 설계:**
  + Apache Kafka를 활용하여 실시간 스트림 수집
  + 고객의 금융 거래 발생 시 즉시 전송 → Stream Processor에서 처리
  + Spark Streaming을 통해 실시간 분석 후 Data Lake에 적재
* **인터페이스 통신 설계:**
  + EAI (Enterprise Application Integration) 방식을 통해 실시간 데이터 전송
  + 내부망(MCA) → 외부망(FEP) → 데이터 레이크 구조
  + FEP 구간에서는 암호화 전송 (TLS 1.3) 적용

**🔍 2. 데이터 적재 및 관리 설계**

* **HDFS 임시 보관 설계:**
  + 배치 수집된 데이터를 HDFS에 임시로 저장
  + 임시 보관 후 24시간 경과 시 정제 레이어로 이동
* **Oracle DB 적재 설계:**
  + 정제된 데이터는 Oracle DB에 최종 저장
  + 정기적인 백업 및 무결성 검증
* **로그 관리 및 감사 기록 설계:**
  + 모든 수집 작업에 대해 ELK Stack을 활용한 실시간 모니터링
  + 배치 작업의 성공/실패 내역 및 에러 내역 로깅

**🔍 3. 보안 및 암호화 설계**

* **전송 구간 암호화:**
  + TLS 1.3 기반 전송 보안 적용
  + EAI, FEP 구간의 데이터는 AES-256으로 암호화
* **전송 파일 암호화:**
  + Secure File Transfer Protocol (SFTP) 적용
  + 금융 거래 정보는 암호화된 ZIP 파일로 전송
* **로그 접근 제어:**
  + SSO 기반 인증을 통해 로깅 접근 제한
  + 모든 접근 이력은 5년간 보관

**🔍 4. 설계 다이어그램 예시**

* **데이터 수집 및 처리 아키텍처:**
  + **배치 수집:**  
    계열사 → DataStage ETL → HDFS → Oracle DB
  + **실시간 수집:**  
    올원뱅크 → Kafka → Spark Streaming → Oracle DB
* **네트워크 구조:**
  + EAI → FEP → MCA → Data Lake
  + TLS 1.3 암호화 채널을 통해 보안 강화

**🔍 5. 테스트 및 검증 방안**

* **배치 처리 테스트:**
  + Control-M의 정기적 배치 테스트 수행
  + 정상 처리, 실패 처리, 롤백 테스트 수행
* **실시간 처리 테스트:**
  + Kafka Consumer가 모든 이벤트를 정상 처리하는지 검증
  + 1초에 1000 TPS 이상의 데이터를 수집 시 안정성 테스트
* **암호화 테스트:**
  + TLS 및 AES-256이 정상적으로 암호화 되는지 검증
  + 전송 중 데이터 변조 여부 탐지

**✅ 다음 진행 계획:**

이제 \*\*03. 비즈니스 아키텍처 (BA)\*\*를 작성하겠습니다.

* 개인화 마케팅 서비스 설계
* 데이터 수집 및 활용 프로세스
* 데이터 전처리 및 임시 보관
* 목적별 마트 구성 및 고객번호 관리
* 데이터 분석 및 모델 운영
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 비즈니스 프로세스 정의
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 03. 비즈니스 아키텍처 (BA)**

**03.1 개인화 마케팅 서비스 설계**

개인화 마케팅 시스템은 NH 농협은행, NH 카드, NH 투자증권의 고객 데이터를 통합하여 **개인화된 마케팅 전략**을 실시간으로 실행합니다.  
주요 목표는 **정확한 타겟팅, 실시간 캠페인 실행, 고객 반응 분석**을 통해 마케팅 ROI를 극대화하는 것입니다.

| **기능 분류** | **주요 설명** | **기술 스택** |
| --- | --- | --- |
| **타겟 고객 선정** | - 빅데이터 분석을 통한 고객 세분화 - 금융 거래 내역 기반 타겟팅 | Spark, Kafka |
| **캠페인 설정 및 관리** | - 캠페인 우선순위 설정 - 피로도 정책 관리 - 실시간 타겟 마케팅 실행 | Spring Boot, Apache Kafka |
| **실시간 이벤트 실행** | - 고객 행동에 따른 실시간 이벤트 트리거 - 예: 결제 시 프로모션 알림 발송 | Event Listener, Webhook |
| **성과 분석** | - 캠페인 결과 추적 및 ROI 분석 - 오퍼 시나리오 및 패널 분석 결과 제공 | Tableau, PowerBI |
| **이력 관리 및 로깅** | - 캠페인 실행 이력 관리 - 분석 결과 및 고객 반응 로그 보관 | ELK Stack (Elastic, Kibana) |

**03.2 세부 내용**

* **타겟팅 전략:**
  + 고객의 금융 거래 내역, 방문 이력, 카드 사용 패턴 등을 분석하여 마케팅 대상 선정
  + Spark 기반의 실시간 분석을 통해 잠재 고객을 식별
* **캠페인 설정:**
  + 개인화 마케팅 시나리오 설정
  + 고객의 라이프사이클에 맞춘 타겟팅 (예: 예적금 만기, 보험 갱신 등)
  + 피로도 관리 정책 적용 (30일 내 동일 오퍼 중복 방지)
* **실시간 이벤트 처리:**
  + 고객이 특정 금융 거래(예: 카드 결제, 대출 신청) 발생 시 실시간 이벤트 처리
  + Kafka 스트림 기반으로 은행 마케팅 허브와 연동
* **성과 분석 및 이력 관리:**
  + 모든 캠페인의 성과를 실시간으로 분석하고, Tableau를 통해 시각화
  + 오퍼 시나리오, 패널 분석 등 다양한 분석 결과 제공

**03.3 설계 및 구현 방안**

**🔍 1. 타겟 고객 분석 및 선정 설계**

* **Spark ML을 활용한 고객 세분화:**
  + 금융 거래 내역을 기반으로 고객의 사용 패턴 분석
  + KMeans 클러스터링을 통해 고객의 유형을 자동 분류
  + 고가치 고객, 이탈 위험 고객, 신규 고객으로 세분화
* **Kafka 스트림을 통한 실시간 타겟팅:**
  + 고객의 행동 이벤트가 발생할 때마다 Kafka 토픽에 전송
  + 실시간 분석 후 적합한 마케팅 오퍼를 생성

**🔍 2. 캠페인 설정 및 관리 설계**

* **Spring Boot 기반의 마케팅 엔진:**
  + REST API를 통해 마케팅 시나리오 설정
  + 캠페인 우선순위 및 피로도 정책을 DB에 저장
  + Redis를 활용한 실시간 정책 조회
* **캠페인 시나리오 예시:**
  + **Case 1:** 신규 가입 고객 → 30일 이내 카드 사용 유도
  + **Case 2:** 예적금 만기 도래 고객 → 연장 프로모션 알림
  + **Case 3:** 대출 만기 고객 → 이자율 할인 캠페인

**🔍 3. 실시간 이벤트 처리 설계**

* **이벤트 리스너(Event Listener):**
  + 고객의 카드 사용, 대출 신청 등 주요 이벤트를 리스닝
  + Kafka에 전송된 이벤트는 Event Listener가 처리
* **실시간 트리거:**
  + 예를 들어, **카드 결제 시 1초 내로 프로모션 메시지 전송**
  + 마케팅 허브와 연동하여 실시간 타겟팅 실행

**🔍 4. 성과 분석 및 리포팅 설계**

* **BI 포털 구성:**
  + Tableau 및 PowerBI를 통해 실시간 캠페인 성과 시각화
  + 금융지주 전체에 대한 마케팅 ROI 분석
* **로그 및 이력 관리:**
  + ELK Stack을 활용한 실시간 로깅
  + 캠페인 실행 기록은 5년간 보관 후 파기

**🔍 5. 설계 다이어그램 예시**

* **마케팅 시스템 아키텍처**
  + 데이터 수집 → 분석 (Spark ML) → 타겟팅 (Kafka) → 캠페인 실행 (Spring Boot)
  + 실시간 연계: Kafka → 마케팅 허브 → 고객 접점
* **네트워크 구조:**
  + Web → API Gateway → Campaign Engine → Marketing Hub

**🔍 6. 테스트 및 검증 방안**

* **타겟팅 정확성 테스트:**
  + 고객의 거래 이력에 따른 세분화가 정확히 작동하는지 검증
  + 1000명의 가상 데이터를 생성하여 시나리오 테스트
* **캠페인 실행 속도 테스트:**
  + 실시간 이벤트에서 고객에게 전달되는 시간 측정
  + SLA 기준 1초 이내 전달 목표
* **성과 분석 테스트:**
  + BI 포털에서 조회된 데이터와 실제 캠페인 성공률 비교
  + 마케팅 ROI 분석 정확성 검증

**✅ 다음 진행 계획:**

이제 \*\*04. 데이터 아키텍처 (DA)\*\*를 작성하겠습니다.

* 데이터 주제 영역 정의
* 원천 데이터 적재 및 목적별 마트 구성
* 메타 데이터 관리 및 조회
* 분석용 목적 DB 구성
* 데이터 보관 주기 및 파기 정책
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 데이터 구조 정의 및 관리 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 04. 데이터 아키텍처 (DA)**

**04.1 데이터 주제 영역 정의**

NH 금융지주 그룹의 데이터 아키텍처는 계열사의 데이터 주제영역을 기반으로 통합 데이터 플랫폼을 구성합니다.  
데이터 주제영역은 주요 금융 활동에 대한 데이터를 그룹화하여 관리하며, 통합 분석 및 마케팅 활용을 최적화합니다.

|  |  |  |
| --- | --- | --- |
| 주제영역 | 세부 항목 | 설명 |
| 고객정보 | 고객기본정보, 고객실적정보, 고객관리정보 | 개인 및 법인 고객의 금융 거래 정보와 실적, 관리 정보를 포함 |
| 상품정보 | 상품기본정보, 상품분류정보 | 금융 상품(예적금, 대출, 카드, 보험 등)에 대한 기본 정보 및 분류 정보 |
| 거래정보 | 수신거래, 여신거래, 카드거래, 외환거래, 연금거래, 보험거래, 금융투자거래 | 고객의 금융 거래 이력을 포함하며, 상품별로 세분화하여 관리 |
| 채널정보 | 가입채널정보, 채널이용정보, 상담정보 | 고객이 금융 서비스를 이용한 채널 정보 (ATM, 인터넷뱅킹, 모바일 등) |
| 공통정보 | 공통코드, 사무소/가맹점정보, 기타 공통정보 | 금융 서비스 전반에 걸쳐 공통적으로 사용되는 참조 데이터 관리 |

**04.2 세부 내용**

* **원천 데이터 적재:**
  + 계열사에서 배치 또는 실시간으로 수집된 원천 데이터를 통합 레이크에 적재
  + Oracle HEX 기반의 DB에 이중화 구조로 저장
* **목적별 마트 구성:**
  + 원본 레이어 (Raw Layer) → 정제 레이어 (Cleansed Layer) → 분석 레이어 (Analytics Layer)
  + 실시간 분석을 위한 Kafka Stream → Spark 처리 후 적재
* **메타 데이터 관리:**
  + NH농협은행의 메타 데이터 정보는 Oracle HEX와 Redis에 저장
  + 주요 정보: 테이블 스키마, 컬럼 타입, 인덱스 구조, 접근 권한 정보
* **분석용 DB 구성:**
  + Tableau, PowerBI에서 실시간 조회가 가능한 구조로 설계
  + Spark SQL과 연동하여 대용량 데이터를 빠르게 처리
* **데이터 보관 주기 및 파기 정책:**
  + 금융규정에 따라 주요 데이터는 5년간 보관 후 안전하게 삭제
  + 암호화된 백업을 통해 유출 위험 방지

**04.3 설계 및 구현 방안**

**🔍 1. 데이터 적재 및 저장 설계**

* **Oracle HEX 기반 적재:**
  + 실시간 수집된 데이터는 Streaming Layer에서 Oracle DB로 적재
  + 배치 수집된 데이터는 Batch Layer에서 HDFS로 먼저 적재 후 Oracle에 이관
* **Kafka & Spark Streaming:**
  + 실시간 고객 행동 정보는 Kafka Topic에 등록
  + Spark Streaming이 이를 실시간으로 수집하여 분석
* **ETL(DataStage) 처리:**
  + 배치 수집된 데이터를 DataStage로 정제 및 클렌징
  + 중복 레코드 제거, NULL 값 처리 후 분석용 DB에 저장

**🔍 2. 데이터 구조 설계**

* **3-Layer Architecture:**
  + **Raw Layer:** 수집된 원본 데이터 (HDFS에 적재)
  + **Cleansed Layer:** 정제 및 통합 처리된 데이터 (Oracle DB)
  + **Analytics Layer:** 분석과 리포팅을 위한 데이터 (OLAP DB)
* **데이터 마트 구성:**
  + 분석 목적에 따라 Customer Mart, Product Mart, Transaction Mart로 분리
  + OLAP 연동 시 빠른 조회가 가능하도록 설계

**🔍 3. 데이터 처리 및 관리 전략**

* **메타 데이터 관리:**
  + 테이블 스키마, 컬럼 타입, 접근 권한 정보를 메타 레지스트리로 관리
  + Redis를 활용하여 빠른 조회 지원
* **데이터 보관 및 파기 정책:**
  + 금융권 규정에 따라 거래 데이터는 5년간 보관 후 파기
  + 파기 이력은 암호화된 로그로 저장되며, 감사 대응이 가능
* **암호화 및 마스킹 처리:**
  + 개인정보는 AES-256으로 암호화
  + 사용자 조회 시 마스킹 적용 (예: 010-\*\*\*\*-1234)

**🔍 4. 설계 다이어그램 예시**

* **데이터 적재 및 처리 아키텍처:**
  + **수집:** 계열사 → DataStage → HDFS (Raw Layer)
  + **정제:** Spark → Oracle DB (Cleansed Layer)
  + **분석:** Spark SQL → OLAP DB (Analytics Layer)
* **네트워크 구성:**
  + PaaS (Spring Boot) → Oracle DB → BI 포털
  + 실시간 스트림: Kafka → Spark Streaming → HDFS

**🔍 5. 테스트 및 검증 방안**

* **데이터 적재 테스트:**
  + 배치 작업 및 실시간 수집 시 데이터 유실 없이 적재되는지 검증
  + 1만 건 이상의 대량 데이터 수집 테스트
* **데이터 정합성 검증:**
  + 적재된 데이터가 원본과 동일한지 체크
  + 중복 데이터 처리 및 NULL 값 처리 확인
* **암호화 및 접근 제어 테스트:**
  + AES-256 암호화가 올바르게 적용되는지 확인
  + SSO 및 RBAC(Role Based Access Control)로 권한 제어

**✅ 다음 진행 계획:**

이제 \*\*05. 보안 아키텍처 (SA)\*\*를 작성하겠습니다.

* 제3자 동의 기반 통합 데이터 인증
* 전송구간 암호화
* 전송파일 암호화
* 개인정보 처리 시 암호화/마스킹/파기
* 계정 및 권한 관리 체계
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 보안 체계 정의 및 암호화 방안
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 05. 보안 아키텍처 (SA)**

**05.1 제3자 동의 기반 통합 데이터 인증**

NH 금융지주 그룹의 통합 데이터 플랫폼은 고객의 민감한 금융 정보를 다루기 때문에, 제3자 동의 기반의 강력한 보안 체계가 필요합니다.

* **정보 제공 동의서 관리:** 고객이 개인정보를 제3자(계열사)와 공유할 수 있도록 동의서 관리
* **SSO (Single Sign-On) 인증:** 사용자 인증 시 금융지주 전 계열사에 대한 통합 인증
* **OAuth2.0 기반 권한 관리:** 각 계열사별 API 호출 시 권한을 OAuth2.0으로 제어
* **PKI (Public Key Infrastructure):** 데이터 전송 시 공개키 기반의 암호화 적용

**05.2 전송구간 암호화**

데이터가 계열사와 통합 플랫폼 간 전송될 때 발생할 수 있는 보안 위험을 차단하기 위해 **전송구간 암호화**를 적용합니다.

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 설명 | 기술 스택 |
| TLS 1.3 암호화 | - 은행 내/외부 간 전송 시 TLS 1.3으로 전송 계층 보호 | OpenSSL, Let's Encrypt |
| VPN 연결 | - 주요 계열사 간 전송 시 VPN 터널링을 통해 보안성 확보 | OpenVPN, Cisco VPN |
| MFT(Managed File Transfer) | - 파일 전송 시 EZGATOR를 통한 암호화 전송 | EZGATOR, FOS MFT |
| PKI 인증서 기반 암호화 | - 전송 시 공개키 기반의 데이터 암호화 적용 | HashiCorp Vault |

**05.3 전송파일 암호화**

계열사 간 전송되는 대용량 파일은 **암호화된 ZIP 파일** 형식으로 관리됩니다.

* **AES-256 암호화:** 모든 전송 파일을 AES-256 방식으로 암호화
* **파일 전송 시 JWT 토큰 사용:** JWT 토큰으로 사용자 인증 후 전송 허용
* **파일 무결성 검증:** 전송 완료 후 SHA-256 해시 검증

**05.4 개인정보 처리 시 암호화/마스킹/파기**

금융 데이터를 안전하게 관리하기 위해, **암호화, 마스킹, 파기 정책**을 일관되게 적용합니다.

|  |  |  |
| --- | --- | --- |
| 보안 처리 유형 | 설명 | 적용 기술 |
| 암호화 | - 개인정보 및 금융 거래 내역은 AES-256, RSA 방식으로 암호화 | AES-256, RSA |
| 마스킹 | - 조회 시 민감 정보는 일부 마스킹 처리 (예: 010-\*\*\*\*-1234) | Oracle Masking, Redis |
| 파기 | - 금융 규정에 따른 보관 주기 만료 시 안전하게 삭제 | Data Lifecycle Management |
| 접근 제어 | - RBAC(Role-Based Access Control)로 접근 제어 관리 | Spring Security |
| 로그 감사 | - 모든 접근 기록은 ELK Stack으로 로깅하고 감사 기록 보관 | ElasticSearch, Kibana |

**05.5 계정 및 권한 관리 체계**

통합 데이터 플랫폼의 계정 및 권한 관리는 \*\*SSO(Single Sign-On)\*\*를 통해 일원화되며, **Role-Based Access Control (RBAC)** 모델로 관리됩니다.

|  |  |  |
| --- | --- | --- |
| 계정 유형 | 설명 | 접근 권한 |
| 관리자 (Admin) | - 모든 데이터 조회 및 수정 권한 - 사용자 관리 및 로그 조회 | 데이터 삭제, 권한 변경, 로그 조회 |
| 운영자 (Operator) | - 분석 결과 조회 및 캠페인 실행 - OLAP 접근 및 보고서 생성 | 데이터 조회, 분석 리포트 생성 |
| 사용자 (User) | - 개인화 마케팅 정보 및 금융 데이터 조회 - API를 통한 데이터 접근 | 데이터 조회만 가능 |
| 외부 사용자 (External) | - 특정 API만 접근 가능 - 뷰(view) 권한만 허용 | 제한된 API 호출 |

**05.6 설계 및 구현 방안**

**🔍 1. 전송 구간 보안 설계**

* **TLS 1.3 암호화:**
  + 모든 통신은 HTTPS를 통해 암호화
  + 외부 인터페이스 구간(FEP, MCA)은 VPN을 통해 추가 보안
* **PKI 기반 인증:**
  + 공공 인증서를 사용하여 데이터 무결성 및 전송 중 변경 방지
  + EZGATOR를 통한 대용량 파일 전송 시 암호화된 ZIP 사용

**🔍 2. 파일 전송 보안 설계**

* **파일 암호화:**
  + AES-256 방식으로 모든 파일 암호화 후 전송
  + 복호화 시 권한 인증이 완료된 사용자만 접근
* **파일 전송 시점 검증:**
  + 전송 중 해시값(SHA-256) 검증을 통해 무결성 확인
  + 전송 실패 시 자동 재전송 및 알림 시스템

**🔍 3. 개인정보 보호 및 파기 설계**

* **마스킹 처리:**
  + 조회 시 이름, 연락처, 주민번호 등 주요 정보 마스킹
  + 예: 010-\*\*\*\*-1234, 홍\*\*
* **자동 파기 정책:**
  + 금융 거래 데이터는 5년 보관 후 안전하게 삭제
  + 자동화된 배치 작업으로 주기적 클린업 실행
* **RBAC 접근 제어:**
  + 관리자(Admin), 운영자(Operator), 사용자(User)로 구분된 권한 체계
  + Spring Security를 활용한 Role Mapping

**🔍 4. 설계 다이어그램 예시**

* **보안 아키텍처 다이어그램:**
  + 데이터 수집 → 암호화 (AES-256) → 전송 (TLS 1.3) → 통합 DB
  + 권한 인증 → SSO → API Gateway → RBAC 처리

**🔍 5. 테스트 및 검증 방안**

* **전송구간 암호화 테스트:**
  + HTTPS 통신을 통해 데이터 암호화 전송 여부 검증
  + 패킷 분석 도구를 활용하여 데이터 유출 여부 체크
* **파일 전송 검증:**
  + AES-256 암호화된 ZIP 파일의 무결성 검증
  + 파일 전송 후 해시값이 동일한지 확인
* **접근 권한 검증:**
  + RBAC(Role-Based Access Control) 권한이 올바르게 설정되었는지 테스트
  + Unauthorized Access 시도 시 차단 로그 생성

**✅ 다음 진행 계획:**

이제 \*\*06. 인터페이스 아키텍처 (IA)\*\*를 작성하겠습니다.

* 계열사 데이터 수집 인터페이스 설계
* 올원뱅크 행동 데이터 수집
* 데이터 연계 표준 및 프로토콜 설계
* 분석 플랫폼과의 연계 설계
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 인터페이스 설계 구조 정의
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 06. 인터페이스 아키텍처 (IA)**

**06.1 계열사 데이터 수집 인터페이스 설계**

NH 금융지주 그룹의 다양한 계열사로부터 데이터를 수집하기 위한 표준화된 인터페이스 구조를 정의합니다.  
**배치(Batch) 및 실시간(Real-time)** 두 가지 방식으로 데이터를 수집하며, 안정적이고 신속한 전송을 보장합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 방식 | 기술 스택 | 설명 |
| 배치 수집 | 정기적 수집 (Batch) | DataStage, Control-M | - 매일 2회 정기 배치로 계열사 데이터 수집 - 금융 거래 내역, 고객 정보, 카드 정보 |
| 실시간 수집 | 실시간 전송 (Streaming) | Apache Kafka, Spark Streaming | - 고객 거래 발생 시 실시간 수집 - 올원뱅크 행동 데이터 실시간 처리 |
| 표준 인터페이스 | HTTP, SFTP, JDBC | FEP, MCA, EAI | - 외부망 및 내부망 간 표준 인터페이스 프로토콜 준수 - 파일 전송, 데이터 동기화 |

**06.2 올원뱅크 행동 데이터 수집**

올원뱅크(모바일 뱅킹) 사용자의 행동 데이터를 실시간으로 수집합니다.

* **주요 수집 항목:** 로그인 정보, 거래 정보, 페이지 이동, 클릭 이벤트
* **전송 방식:** Kafka Topic을 활용한 실시간 스트림
* **분석 구조:** Spark Streaming을 통해 실시간 분석 후 Data Lake에 저장

**06.3 데이터 연계 표준 및 프로토콜 설계**

NH 금융지주 내의 데이터 전송은 표준화된 프로토콜에 따라 보안 및 일관성을 유지합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 프로토콜 | 기술 스택 | 설명 |
| 내부 연계 | MCA, EAI | WebSphere MQ | - 내부 시스템 간 메시지 전송 및 실시간 연계 |
| 외부 연계 | FEP, MFT | EZGATOR, FOS MFT | - 외부 금융기관 및 공공기관과의 데이터 전송 |
| 실시간 전송 | HTTP, WebSocket | Apache Kafka | - 모바일 뱅킹 사용자의 실시간 거래 내역 수집 |
| 대용량 파일 | SFTP, FTPS | FileZilla, OpenSSH | - 대용량 금융 데이터 파일 전송 |

**06.4 분석 플랫폼과의 연계 설계**

계열사에서 수집된 데이터를 OLAP, BI 포털, Big Square와 연동하여 실시간 분석 및 마케팅 활용이 가능합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 연계 대상 | 연계 방식 | 기술 스택 | 설명 |
| OLAP Platform | JDBC, ODBC | Tableau, PowerBI | - 대용량 데이터 분석 및 시각화 |
| BI Portal | REST API, GraphQL | Spring Boot, Tomcat | - 실시간 리포팅, 통합 분석 대시보드 |
| Big Square | Spark Streaming, Kafka | Apache Spark, HDFS | - ML/DL 모델 학습 및 실시간 분석 지원 |
| Data Lake | HDFS, S3 | Hadoop, AWS S3 | - 수집된 모든 데이터를 중앙 집중화하여 관리 |

**06.5 설계 및 구현 방안**

**🔍 1. 인터페이스 설계 및 구현**

* **EAI 기반 연계:**
  + WebSphere MQ를 사용하여 은행 내 여러 시스템 간 실시간 데이터 동기화
  + 송신자는 송신 큐에 데이터를 전송하고, 수신자는 수신 큐에서 데이터를 읽음
* **FEP (Front-End Processor) 연계:**
  + 계열사 외부망과 연결된 시스템 간 안전한 데이터 전송
  + EZGATOR를 통한 대용량 파일의 안전한 암호화 전송
* **MFT (Managed File Transfer):**
  + 정기적으로 발생하는 대용량 데이터는 SFTP를 통해 전송
  + OpenSSH를 통해 암호화된 파일 송수신

**🔍 2. 실시간 스트림 처리 설계**

* **Kafka Stream:**
  + 올원뱅크 행동 데이터를 실시간으로 전송
  + Spark Streaming이 이를 실시간 분석하고 Data Lake에 저장
* **실시간 분석 예시:**
  + **카드 결제 발생:**
    - Kafka → Spark → Data Lake
    - 실시간으로 결제 정보가 마케팅 허브에 전달
  + **대출 신청 시 이벤트 발생:**
    - Kafka → Spring Boot → CRM 시스템에 등록

**🔍 3. 데이터 표준화 및 관리 설계**

* **통합 고객번호 관리:**
  + 모든 고객 데이터는 Master ID를 기준으로 통합 관리
  + Redis Cache를 활용하여 조회 속도 최적화
* **메타 데이터 관리:**
  + 전송되는 데이터의 메타 정보를 관리 (Schema, Type, Access Control)
  + Spring Data JPA를 통해 메타 정보를 CRUD

**🔍 4. 설계 다이어그램 예시**

* **인터페이스 아키텍처 다이어그램:**
  + **내부망:** WebSphere MQ → Oracle DB
  + **외부망:** FEP → MCA → HDFS
  + **실시간 처리:** Kafka → Spark Streaming → Data Lake

**🔍 5. 테스트 및 검증 방안**

* **배치 처리 테스트:**
  + Control-M을 통한 배치 테스트 진행
  + 예기치 않은 실패 발생 시 자동 Rollback
* **실시간 연계 테스트:**
  + Kafka에서 이벤트 발생 시 1초 내로 소비되는지 검증
  + Spring Boot Listener가 적절히 수신하는지 확인
* **보안 테스트:**
  + SFTP 전송 시 파일이 암호화된 상태로 송수신 되는지 확인
  + TLS 1.3 암호화 채널이 정상 동작하는지 검증

**✅ 다음 진행 계획:**

이제 **07. 기대효과 및 전략적 가치**를 작성하겠습니다.

* 비용 절감 효과
* 통합 데이터 기반 의사결정 지원
* 분석 모델을 통한 마케팅 효과 증대

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 기대효과 정의
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 07. 기대효과 및 전략적 가치**

**07.1 비용 절감 효과**

NH 금융지주의 통합 데이터 플랫폼 구축을 통해, 각 계열사가 개별적으로 데이터 수집, 처리, 분석을 수행하던 기존 방식에서 벗어나 **중앙 집중화된 플랫폼**에서 모든 작업을 통합적으로 관리할 수 있습니다.

| **항목** | **기존 방식** | **통합 플랫폼 도입 후** |
| --- | --- | --- |
| **데이터 수집 비용** | 개별 배치 및 실시간 수집, 중복된 인프라 비용 발생 | 통합된 ETL(DataStage) 및 Kafka를 활용한 일원화 |
| **분석 및 마케팅 비용** | 개별 BI 시스템 사용으로 비용 중복 발생 | Tableau, PowerBI 중앙 분석으로 비용 절감 |
| **시스템 유지보수 비용** | 은행, 카드, 증권 별 개별 시스템 관리로 높은 유지보수 비용 | 중앙 통합 관리로 운영 효율화 및 유지보수 비용 절감 |
| **스토리지 비용** | 중복된 데이터 저장으로 스토리지 사용량 과다 | HDFS 및 S3 기반 통합 데이터 레이크 구성으로 최적화 |
| **보안 관리 비용** | 각 시스템 별 별도의 보안 정책 수립 | SSO 및 OAuth2.0 기반 통합 인증 및 접근 관리 |

**07.2 통합 데이터 기반 의사결정 지원**

통합 데이터 플랫폼을 통해 실시간 분석 및 신속한 의사결정이 가능합니다.

* **OLAP 플랫폼:** Tableau와 PowerBI를 통해 실시간 대시보드 제공
* **실시간 금융 데이터 분석:** Spark Streaming을 통해 실시간 거래 내역 분석
* **대시보드 예시:**
  + 고객 유형별 금융 상품 선호도
  + 실시간 거래 빈도 분석
  + 지역별 대출 및 예적금 현황

**07.3 분석 모델을 통한 마케팅 효과 증대**

통합된 데이터 플랫폼은 개인화 마케팅을 위한 실시간 데이터 분석을 가능하게 하며, **타겟 마케팅, 우선순위 설정, 피로도 정책**을 반영한 캠페인을 실시간으로 실행합니다.

| **마케팅 전략** | **설명** | **기술 스택** |
| --- | --- | --- |
| **실시간 타겟 마케팅** | - 고객의 금융 거래 시 실시간 분석 후 맞춤형 오퍼 제공 | Kafka, Spark Streaming |
| **우선순위 설정** | - 고객의 금융 거래 이력을 분석하여 우선순위 마케팅 실행 | Spring Boot, Redis |
| **피로도 정책 관리** | - 반복적인 마케팅 노출을 피하기 위해 피로도 정책 반영 | Spring Security |
| **성과 분석 리포트** | - 모든 마케팅 캠페인의 성과를 실시간으로 분석하여 리포팅 | Tableau, PowerBI |
| **캠페인 자동화 실행** | - 특정 조건이 만족되면 자동으로 캠페인이 실행되며 실시간 전송 | Kafka, Spring Scheduler |

**07.4 설계 및 구현 방안**

**🔍 1. 비용 절감 방안**

* **클라우드 리소스 최적화:**
  + PaaS 환경에서 Spring Boot, Tomcat 클러스터링으로 비용 절감
  + Oracle DB 및 HDFS 이중화를 통해 안정성 확보
* **배치 및 실시간 수집 최적화:**
  + Control-M을 통해 배치 시간 최적화
  + DataStage로 데이터 정제 및 중복 제거
* **스토리지 최적화:**
  + HDFS와 S3를 활용한 데이터 적재 구조
  + 중복 저장 방지 및 시점 복원 기능

**🔍 2. 실시간 분석 및 의사결정 방안**

* **OLAP 환경 구축:**
  + Tableau 및 PowerBI를 통한 실시간 분석 환경 제공
  + 고객 세분화 및 금융상품 선호도 분석
* **BI 대시보드 설계:**
  + 실시간 거래 내역 분석
  + 캠페인 성과 분석 (ROI, Conversion Rate)

**🔍 3. 마케팅 캠페인 자동화 방안**

* **Kafka 기반 실시간 처리:**
  + 고객 거래 발생 시 Kafka에 이벤트 등록
  + Spring Boot에서 트리거 실행 → 실시간 캠페인 전송
* **Redis를 활용한 피로도 정책 관리:**
  + 최근 30일 이내 동일 캠페인 노출을 차단
  + 고객의 행동 패턴에 따라 맞춤형 이벤트 실행

**🔍 4. 설계 다이어그램 예시**

* **통합 마케팅 아키텍처 다이어그램:**
  + 데이터 수집 → 분석 (Spark) → 타겟팅 (Kafka) → 캠페인 실행 (Spring Boot)
  + 실시간 연계: Kafka → 마케팅 허브 → 고객 접점
* **BI 리포트 구성도:**
  + OLAP → Tableau → 실시간 대시보드
  + 고객 세분화 → 실시간 ROI 분석

**🔍 5. 테스트 및 검증 방안**

* **비용 최적화 검증:**
  + Cloud 비용 분석 대시보드 구축
  + 실제 사용량 대비 예산 계획 검증
* **실시간 분석 검증:**
  + OLAP 대시보드의 데이터 적재 시간 측정
  + 실시간 트랜잭션 분석 속도 테스트
* **캠페인 실행 검증:**
  + Kafka 이벤트 생성 → Spring Boot 실행 속도
  + 고객 응답 시간 분석 및 최적화

**✅ 다음 진행 계획:**

이제 **08. 아키텍처 주요 정보**를 작성하겠습니다.

* 도입 시스템 정의
  + 신규 도입
  + 기존 시스템 활용
* 인터페이스 구성도
* 데이터 주제 영역 정의
* 도입 솔루션 목록

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 시스템 정의 및 구성 정보
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 08. 아키텍처 주요 정보**

**08.1 도입 시스템 정의**

NH 금융지주 그룹의 통합 데이터 플랫폼 구축을 위한 **신규 도입 시스템**과 **기존 시스템 활용**에 대한 정의입니다.  
모든 시스템은 **Active-Standby 이중화 구조**로 설계되며, **PaaS 및 IaaS 기반**으로 클라우드 환경에서 운영됩니다.

**🔍 1. 신규 도입 시스템**

| **시스템명** | **스펙 및 구성** |
| --- | --- |
| **통합 데이터 DB** | - Oracle HEX 2대 도입 - Active-Standby 구성 - 2CPU \* 8CORE, 256GB MEM, 1.2TB HDD |
| **통합 데이터 운영 DB Storage** | - 3TB 용량으로 설정 - RAID 10 구성 - 금융 규정에 따른 5년 보관 정책 적용 |
| **통합 데이터 ETL 서버** | - DataStage 기반 ETL 처리 - 2대 도입 - 2CPU \* 4CORE, 256GB MEM, 1.2TB HDD |

**🔍 2. 기존 시스템 활용**

| **구성 플랫폼** | **스택 및 솔루션** |
| --- | --- |
| **PaaS 기반** | - WEB 서버: Apache, xFRAME UI 솔루션 - WAS 서버: Tomcat, Spring Boot |
| **IaaS 기반** | - ETL 서버: DataStage, Control-M Client - OLAP 서버: JEUS, MSTR |
| **분석 플랫폼** | - OLAP DB: MySQL - BI 포털 DB: MySQL |
| **마케팅 HUB 서버** | - Tomcat, Spring Boot, 2대 구성 |
| **e-마케팅 시스템** | - Tomcat, Spring Boot |
| **UMS 서버** | - WebLogic 기반 운영 |
| **은행 배치 AP 서버** | - Spring Boot 기반 금융 배치 처리 |
| **은행 RDW/ADW 서버** | - Oracle HEX 구성, 2대로 이중화 운영 |

**08.2 인터페이스 구성도**

통합 데이터 플랫폼의 모든 데이터는 표준화된 인터페이스를 통해 송수신됩니다.

| **연계 대상** | **전송 프로토콜** | **기술 스택** | **설명** |
| --- | --- | --- | --- |
| **계열사 파일 전송 → 통합 데이터 영역** | SFTP, MFT | EZGATOR, FOS | - 계열사 데이터 전송 시 암호화 및 MFT 전송 |
| **은행 파일 전송 → 통합 데이터 영역** | SFTP | FOS, OpenSSH | - 금융 거래 데이터 전송 시 암호화 처리 |
| **통합 데이터 ETL ↔ DB 연계** | JDBC, ODBC | Oracle HEX, MySQL | - ETL 처리 후 통합 DB에 실시간 반영 |
| **OLAP AP ↔ 통합 데이터 DB** | JDBC | Tableau, PowerBI | - 실시간 분석 및 시각화를 위한 연동 |
| **BI 포털 AP ↔ 통합 데이터 DB** | REST API | Spring Boot, Tomcat | - 실시간 대시보드 및 리포팅 |
| **대외 MCA → FOS Storage** | HTTP, WebSocket | FEP, MCA, EZGATOR | - 금융 규정에 따른 암호화 및 전송 관리 |
| **Big Square ↔ FOS Storage** | FTP, SFTP | Apache Hadoop, Spark | - 금융 데이터 모델 학습 및 분석 환경 |

**08.3 데이터 주제영역 정의**

통합된 데이터 플랫폼에서 관리할 주요 주제영역입니다.

| **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 |

**08.4 도입 솔루션 목록**

통합 데이터 플랫폼에서 사용될 주요 솔루션과 기술 스택을 정리합니다.

| **구분** | **솔루션** | **기술 스택** | **설명** |
| --- | --- | --- | --- |
| **WEB** | Apache | PaaS 기반의 웹 서버 | Frontend 웹 애플리케이션 호스팅 |
| **Application** | Spring Boot | Microservices Framework | 모든 비즈니스 로직 처리 및 API 관리 |
| **WAS** | Tomcat, WebLogic | PaaS 기반의 WAS 서버 | Spring Boot 배포 및 실행 환경 |
| **ETL** | DataStage | IBM ETL Tool | 계열사 데이터의 추출, 변환, 적재 수행 |
| **DBMS** | Oracle HEX, MySQL | RDBMS, OLAP | 데이터 보관 및 분석용 데이터 관리 |
| **Messaging** | Kafka, WebSphere MQ | Message Queue | 실시간 스트림 처리 및 메시지 전달 |
| **Storage** | HDFS, S3 | Hadoop, AWS S3 | 대용량 데이터 보관 및 분산 저장 |
| **BI/분석** | Tableau, PowerBI, Spark | Visualization, Data Analysis | 실시간 대시보드 및 대용량 데이터 분석 |
| **File Transfer** | EZGATOR, FOS MFT | Managed File Transfer | 대용량 파일 전송 및 보안 처리 |

**✅ 다음 진행 계획:**

이제 **09. 배포 및 운영 전략**을 작성하겠습니다.

* CI/CD 파이프라인 설계
* 블루-그린 배포, 롤링 업데이트
* 장애 복구 및 DR(Disaster Recovery)
* 실시간 모니터링 및 로깅 관리
* 장애 대응 및 복구 시나리오
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 배포 구조 정의 및 자동화 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 09. 배포 및 운영 전략**

**09.1 배포 및 운영 개요**

NH 금융지주의 통합 데이터 플랫폼은 **무중단 배포와 안정적인 운영**을 목표로 설계되었습니다.  
모든 서비스는 **CI/CD 파이프라인**을 통해 자동으로 배포되며, 장애 발생 시 **Failover 및 DR(Disaster Recovery)** 전략을 적용합니다.  
배포는 **블루-그린(Blue-Green) 배포**와 **롤링 업데이트(Rolling Update)** 방식을 병행하여 서비스 중단 없이 안정성을 확보합니다.

**09.2 세부 내용**

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **CI/CD 파이프라인** | - GitLab CI, Jenkins를 활용한 빌드, 테스트, 배포 자동화 | GitLab CI, Jenkins |
| **배포 방식** | - 블루-그린 배포: 신규 버전과 기존 버전 동시 운영 - 롤링 업데이트: 순차적 업데이트 적용 | Kubernetes, ArgoCD |
| **무중단 서비스** | - Kubernetes의 Zero-Downtime Deployment - 로드 밸런서를 통한 트래픽 분산 | Nginx, HAProxy |
| **장애 복구 (Failover)** | - Active-Standby 구조로 장애 시 즉시 전환 | Oracle HEX, MySQL Replication |
| **DR (재해 복구)** | - 이중화 구성 및 원격지 백업 - 15분 RPO, 30분 RTO 목표 | S3 Glacier, Oracle Data Guard |
| **모니터링 및 로깅** | - Prometheus, Grafana를 통한 실시간 모니터링 - ELK Stack을 통한 로그 관리 | Prometheus, Grafana, ELK Stack |

**09.3 설계 및 구현 방안**

**🔍 1. CI/CD 파이프라인 설계**

* **GitLab CI/CD:**
  + Git 커밋 → 자동 빌드 → 테스트 → Staging 배포 → 프로덕션 배포
  + Spring Boot 애플리케이션의 빌드 및 Docker 이미지 생성
* **Jenkins Pipeline:**
  + Jenkinsfile을 활용하여 자동화 파이프라인 구성
  + 테스트 자동화 (JUnit, Postman), 배포 검증 후 프로덕션 반영
* **ArgoCD:**
  + Kubernetes 클러스터에 대한 GitOps 방식 배포
  + 버전 컨트롤 기반으로 롤백 및 배포 이력 관리

**🔍 2. 배포 방식 설계**

* **블루-그린 배포:**
  + 기존 서비스(Green)와 신규 버전(Blue)을 동시에 운영
  + 모든 테스트가 완료되면 로드 밸런서를 전환하여 무중단 배포
* **롤링 업데이트:**
  + Kubernetes에서 Pod 단위로 순차적으로 업데이트
  + 한 번에 모든 서비스를 교체하지 않고 순차적 교체
* **Zero-Downtime 보장:**
  + 트래픽 라우팅을 통해 업데이트 중에도 서비스 중단 없이 처리

**🔍 3. 장애 복구 및 재해 복구(DR) 설계**

* **Failover 설계:**
  + Active-Standby 이중화 구성
  + Active 노드 장애 시 Standby 노드가 자동 전환
* **Oracle Data Guard:**
  + 데이터베이스 이중화로 장애 시 즉시 전환
  + Point-in-Time Recovery 지원
* **Disaster Recovery:**
  + AWS S3 Glacier에 15분 단위 스냅샷 백업
  + 원격지 백업 시스템을 통해 RTO 30분 이내 복구

**🔍 4. 모니터링 및 로깅 설계**

* **Prometheus & Grafana:**
  + 실시간 메트릭 수집 및 대시보드 구성
  + CPU, Memory, Network IO 사용량 실시간 모니터링
* **ELK Stack:**
  + Elasticsearch: 로그 수집 및 검색
  + Logstash: 실시간 로그 처리
  + Kibana: 대시보드 시각화
* **알림 체계:**
  + 장애 탐지 시 Slack, PagerDuty로 즉시 알림
  + 미리 정의된 SLA를 벗어날 경우 SMS 및 이메일 전송

**🔍 5. 설계 다이어그램 예시**

* **배포 및 운영 아키텍처 다이어그램:**
  + GitLab → Jenkins → Docker Image 생성 → Kubernetes 클러스터 배포
  + Kubernetes → Prometheus/Grafana → 실시간 모니터링
  + ELK Stack → Log Analytics → 알림 전송
* **무중단 배포 아키텍처:**
  + Nginx 로드 밸런서 → Blue (Current Version)
  + 신규 배포 → Green → 트래픽 전환

**🔍 6. 테스트 및 검증 방안**

* **CI/CD 테스트:**
  + GitLab Pipeline에서 통합 테스트 및 단위 테스트 자동 실행
  + 배포 시 Smoke Test 및 Regression Test 수행
* **무중단 배포 검증:**
  + 블루-그린 배포 전환 시 사용자 요청 누락 없는지 확인
  + 트래픽 전환 시 응답 속도 모니터링
* **재해 복구 테스트:**
  + 원격지 백업에서 30분 이내 복구 테스트
  + RPO (15분 이내), RTO (30분 이내) 목표 검증

**✅ 다음 진행 계획:**

이제 **10. 로그 및 모니터링 전략**을 작성하겠습니다.

* 로그 수집 및 분석 설계
* 모니터링 아키텍처 설계
* 장애 탐지 및 알림 체계
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 로그 수집 구조 및 모니터링 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 10. 로그 및 모니터링 전략**

**10.1 로그 수집 및 분석 설계**

NH 금융지주의 통합 데이터 플랫폼에서는 **실시간 로그 수집, 분석, 모니터링**을 통해 장애 발생 시 신속한 대응이 가능하도록 설계되었습니다.  
로그는 \*\*ELK Stack (Elasticsearch, Logstash, Kibana)\*\*을 통해 통합 관리되며, 실시간으로 모니터링할 수 있습니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **로그 수집** | - 각 시스템(Spring Boot, Tomcat, DataStage)에서 발생하는 로그 수집 | Logstash, Fluentd |
| **로그 전송 방식** | - Fluentd를 활용한 실시간 로그 전송 | Fluentd → Logstash |
| **로그 적재 및 분석** | - Elasticsearch에 저장하여 검색 및 분석 | Elasticsearch |
| **모니터링 대시보드** | - Kibana를 통해 실시간 로그 시각화 및 검색 | Kibana |
| **장애 탐지 및 알림** | - 장애 탐지 시 Slack, Email, PagerDuty 알림 전송 | AlertManager, Prometheus Alert |

**10.2 모니터링 아키텍처 설계**

통합 데이터 플랫폼의 안정적인 운영을 위해 **Prometheus**와 **Grafana**를 활용한 모니터링 체계를 구축합니다.  
각 구성 요소의 \*\*CPU 사용량, 메모리, 네트워크 트래픽, TPS(초당 거래량)\*\*을 실시간으로 수집하여 대시보드에 표시합니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **시스템 모니터링** | - CPU, Memory, Disk 사용량 실시간 모니터링 | Prometheus, Grafana |
| **네트워크 모니터링** | - 트래픽 분석 및 TPS(Transaction per Second) 측정 | Prometheus, Grafana |
| **어플리케이션 모니터링** | - Spring Boot, Kafka, DataStage의 실시간 로그 수집 | Prometheus Exporter |
| **데이터베이스 모니터링** | - Oracle, MySQL의 쿼리 실행 시간 및 Connection Pool 모니터링 | Prometheus SQL Exporter |
| **장애 탐지 및 대응** | - 임계치 초과 시 AlertManager를 통한 Slack, Email 알림 | AlertManager |

**10.3 설계 및 구현 방안**

**🔍 1. 로그 수집 설계**

* **Fluentd를 활용한 실시간 로그 수집:**
  + Spring Boot, Tomcat, Kafka, DataStage에서 발생하는 모든 로그를 Fluentd를 통해 실시간 수집
  + JSON 형식으로 변환 후 Logstash에 전달
* **Logstash 필터링:**
  + 로그의 중요도에 따라 INFO, WARN, ERROR로 분류
  + 에러 로그의 경우 별도로 Elasticsearch에 태깅하여 조회 속도 최적화
* **Elasticsearch 적재:**
  + 로그가 적재되면 빠르게 인덱싱되어 Kibana 대시보드에서 조회 가능
  + 5년 보관 주기 설정

**🔍 2. 실시간 모니터링 설계**

* **Prometheus Exporter 설치:**
  + Spring Boot, Kafka, Tomcat, DataStage에 Prometheus Exporter 설치
  + 실시간으로 CPU, 메모리, 네트워크 상태 수집
* **Grafana 대시보드 구성:**
  + 주요 모니터링 항목:
    - **CPU 사용량:** 80% 이상일 경우 알림
    - **메모리 사용량:** 70% 이상일 경우 리소스 점검
    - **TPS(초당 거래량):** 평소 대비 20% 이상 증가 시 경고
    - **네트워크 트래픽:** 1Gbps 이상 초과 시 확인
* **AlertManager 설정:**
  + 임계치 초과 시 Slack, PagerDuty로 알림 전송
  + 장애 탐지 시 5초 이내 통보

**🔍 3. 장애 탐지 및 복구 설계**

* **자동 복구 프로세스:**
  + 장애 탐지 시 Kubernetes의 Pod Auto-Recovery를 통해 자동 복구
  + 3회 이상 동일 장애 발생 시 알림 전송
* **Failover 처리:**
  + Prometheus가 장애 탐지 시 Standby 노드로 자동 전환
  + 데이터 손실 방지를 위해 Oracle Data Guard와 연동
* **로그 분석 기반 Root Cause 분석:**
  + Elasticsearch에 적재된 로그를 분석하여 Root Cause 추적
  + 장애 시점 로그와 Stack Trace 정보를 Kibana로 조회

**🔍 4. 설계 다이어그램 예시**

* **로그 및 모니터링 아키텍처:**
  + 애플리케이션 → Fluentd → Logstash → Elasticsearch
  + Kibana → 실시간 조회 및 대시보드
  + Prometheus → Grafana → 실시간 모니터링
* **장애 탐지 및 알림:**
  + Prometheus → AlertManager → Slack & PagerDuty

**🔍 5. 테스트 및 검증 방안**

* **로그 수집 테스트:**
  + Spring Boot에서 로그 발생 시 Fluentd → Logstash → Elasticsearch로 정상 전달되는지 검증
  + 로그의 필터링 및 태그 적용 여부 확인
* **모니터링 검증:**
  + CPU, Memory, Network 상태가 실시간으로 반영되는지 대시보드에서 확인
  + Grafana에서 설정한 임계치를 초과할 경우 알림 발생 여부 체크
* **장애 복구 테스트:**
  + Pod Auto-Recovery가 정상 동작하는지 검증
  + Standby 노드로 Failover 시 데이터 손실 없이 복구되는지 확인

**✅ 다음 진행 계획:**

이제 **11. API 관리 및 거버넌스 전략**을 작성하겠습니다.

* API 설계 표준화
* API 버전 관리 및 확장 전략
* API 보안 및 접근 제어
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* API 구조 정의 및 거버넌스 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 11. API 관리 및 거버넌스 전략**

**11.1 API 관리 개요**

NH 금융지주의 통합 데이터 플랫폼에서는 다양한 금융 서비스와 마케팅 시스템을 연계하기 위해 **표준화된 API 설계 및 관리 전략**을 도입합니다.  
모든 API는 RESTful 표준을 따르며, **버전 관리, 보안, 접근 제어, 로깅**이 체계적으로 이루어집니다.  
또한, **OAuth2.0, JWT, API Gateway**를 통해 보안성과 확장성을 동시에 확보합니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **API 표준화 설계** | - RESTful API 설계 - URI Naming Rule 및 Method 표준화 | Spring Boot, OpenAPI |
| **API Gateway** | - 외부 서비스와 내부 마이크로서비스 간 트래픽 관리 - 인증 및 요청 필터링 | Zuul, Spring Cloud Gateway |
| **버전 관리** | - /v1, /v2 형태의 버전 관리 적용 - Backward Compatibility 유지 | Swagger, Redoc |
| **보안 및 인증** | - OAuth2.0 및 JWT 토큰 인증 - API 요청 시 인증 및 권한 검증 | Spring Security, Keycloak |
| **API 로깅 및 모니터링** | - 모든 API 호출 내역을 실시간으로 로깅 - 모니터링 및 장애 탐지 | ELK Stack, Prometheus, Grafana |

**11.2 세부 내용**

* **RESTful API 설계 원칙:**
  + HTTP Method 표준화: GET, POST, PUT, DELETE
  + URI 표준화:
    - /api/v1/customers → 고객 정보 조회
    - /api/v1/accounts → 계좌 정보 조회
    - /api/v1/transactions → 거래 내역 조회
* **API Gateway 설계:**
  + **Zuul API Gateway**를 통해 모든 API 요청을 중앙에서 관리
  + 인증, 로깅, 요청 필터링 처리
  + 내부 API는 Private Gateway, 외부 API는 Public Gateway로 분리
* **버전 관리 및 확장성:**
  + /v1, /v2 버전 구분을 통해 호환성 유지
  + 신규 API 추가 시 이전 버전과의 충돌 방지
* **API 보안 및 인증:**
  + OAuth2.0 인증 방식을 사용하여 보안성 강화
  + JWT 토큰을 활용한 Stateless 인증
  + Spring Security로 Role-based 접근 제어

**11.3 설계 및 구현 방안**

**🔍 1. API 설계 및 구현**

* **Spring Boot REST Controller:**
  + 모든 API는 @RestController로 정의
  + Swagger 문서를 기반으로 API 명세 관리
* **API 예시:**

java

복사편집

@RestController

@RequestMapping("/api/v1/customers")

public class CustomerController {

@GetMapping("/{customerId}")

public ResponseEntity<CustomerDTO> getCustomer(@PathVariable String customerId) {

CustomerDTO customer = customerService.getCustomerById(customerId);

return ResponseEntity.ok(customer);

}

@PostMapping

public ResponseEntity<CustomerDTO> createCustomer(@RequestBody CustomerDTO customerDTO) {

CustomerDTO createdCustomer = customerService.createCustomer(customerDTO);

return ResponseEntity.status(HttpStatus.CREATED).body(createdCustomer);

}

}

**🔍 2. API Gateway 설계 및 구현**

* **Zuul 설정:**
  + 모든 API 호출은 Zuul Gateway를 통해 라우팅
  + 인증 필터 및 로깅 필터 적용

yaml

복사편집

zuul:

routes:

customer-service:

path: /api/v1/customers/\*\*

serviceId: CUSTOMER-SERVICE

stripPrefix: false

* **JWT 인증 필터:**
  + 모든 요청에 대해 JWT 토큰을 검증
  + 유효하지 않은 토큰은 403 Forbidden 처리

**🔍 3. 버전 관리 및 확장 설계**

* **URI 버전 관리:**
  + API는 /v1, /v2로 명시적 버전 관리
  + 신규 버전 출시 시 기존 URI를 유지하여 호환성 보장
* **Backward Compatibility:**
  + /v1 API는 서비스 종료 시점까지 유지
  + /v2 출시 시 신규 필드 및 기능 추가

**🔍 4. API 보안 설계**

* **OAuth2.0 인증:**
  + Resource Server와 Authorization Server 분리
  + Spring Security OAuth2.0 적용
* **JWT 토큰 발급 및 검증:**
  + 로그인 시 JWT 토큰 발급
  + 모든 요청 시 Authorization Header로 토큰 검증

json

복사편집

{

"access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"token\_type": "bearer",

"expires\_in": 3600

}

**🔍 5. API 로깅 및 모니터링**

* **ELK Stack 통합:**
  + Logstash → Elasticsearch → Kibana 대시보드 구성
  + API 요청/응답 시간, 에러 로그 실시간 분석
* **Prometheus & Grafana:**
  + API 호출 수, Latency, Error Rate 실시간 모니터링
  + 임계치 초과 시 Slack, PagerDuty로 알림

**🔍 6. 설계 다이어그램 예시**

* **API Gateway 아키텍처:**
  + Client → Zuul API Gateway → Microservice Cluster
  + JWT 인증 → Resource Server → API Response
* **버전 관리 구조:**
  + /v1/customers → 기존 고객 조회 API
  + /v2/customers → 신규 고객 조회 및 추가 기능

**🔍 7. 테스트 및 검증 방안**

* **API 테스트:**
  + Postman을 활용한 CRUD 테스트
  + Swagger UI를 통한 API 문서화 및 실시간 호출 테스트
* **보안 검증:**
  + JWT 토큰 검증 시 만료 및 위조 여부 확인
  + OAuth2.0 Scope에 따른 접근 제한 테스트
* **성능 테스트:**
  + 10,000 TPS 시나리오 기반 부하 테스트
  + 평균 응답 시간 3초 이내 확인

**✅ 다음 진행 계획:**

이제 **12. DevOps 및 CI/CD 전략**을 작성하겠습니다.

* DevOps 전략 개요
* CI/CD 파이프라인 설계
* 자동화 배포 및 모니터링
* 장애 복구 및 롤백 전략
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* DevOps 구조 정의 및 자동화 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 12. DevOps 및 CI/CD 전략**

**12.1 DevOps 전략 개요**

NH 금융지주 통합 데이터 플랫폼의 DevOps 전략은 \*\*지속적인 통합(CI)\*\*과 \*\*지속적인 배포(CD)\*\*를 통해 안정적인 서비스를 보장합니다.  
또한, **자동화된 모니터링 및 장애 복구**를 통해 무중단 서비스를 목표로 설계되었습니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **지속적 통합 (CI)** | - 개발자의 코드 변경 사항이 Git에 커밋될 때마다 자동 빌드 및 테스트 실행 | GitLab CI, Jenkins, SonarQube |
| **지속적 배포 (CD)** | - 테스트 완료된 애플리케이션을 Kubernetes 클러스터에 무중단 배포 | ArgoCD, Helm, Kubernetes |
| **버전 관리** | - GitLab을 통한 소스 코드 버전 관리 및 마이크로서비스 간의 연동 관리 | GitLab, GitOps |
| **배포 방식** | - Blue-Green Deployment, Rolling Update 방식 적용 | Kubernetes, Docker |
| **모니터링 및 로깅** | - Prometheus, Grafana를 활용한 실시간 모니터링 및 시각화 | Prometheus, Grafana, ELK Stack |
| **장애 대응 및 롤백** | - 실패 시 자동 Rollback 처리 및 서비스 이중화 | ArgoCD Rollback, Kubernetes Auto-Recovery |

**12.2 세부 내용**

* **CI/CD 파이프라인 구성:**
  + GitLab CI를 통해 커밋 시 자동 빌드 및 테스트
  + Jenkins Pipeline으로 통합 테스트 및 코드 품질 분석
  + SonarQube를 통한 코드 스멜 및 보안 취약점 탐지
* **CD 파이프라인 배포 방식:**
  + Blue-Green 배포: 기존 버전과 새로운 버전을 동시에 운영 후 전환
  + Rolling Update: Kubernetes Pod 단위로 순차적으로 업데이트
* **모니터링 체계:**
  + Prometheus를 활용하여 실시간 메트릭 수집
  + Grafana를 통해 대시보드 시각화
* **로그 관리:**
  + Fluentd를 통해 애플리케이션 로그 수집 → Logstash → Elasticsearch
  + Kibana를 통해 실시간 검색 및 장애 탐지

**12.3 설계 및 구현 방안**

**🔍 1. CI/CD 파이프라인 설계**

* **GitLab CI 설정:**
  + git push 시 GitLab CI가 트리거되어 다음 단계 진행
    - Build: Maven을 사용하여 빌드
    - Test: JUnit 및 SonarQube로 테스트 실행
    - Docker Build: Dockerfile을 통해 이미지 생성
    - Push: Docker Hub에 이미지 저장

yaml

복사편집

stages:

- build

- test

- deploy

build:

stage: build

script:

- mvn clean install

- docker build -t myapp:latest .

test:

stage: test

script:

- mvn test

- sonar-scanner

deploy:

stage: deploy

script:

- docker push myapp:latest

**🔍 2. CD 파이프라인 배포 설계**

* **ArgoCD 설정:**
  + Kubernetes 클러스터에 GitOps 방식으로 배포
  + GitLab의 마스터 브랜치가 업데이트되면 ArgoCD가 자동 배포
* **Helm Chart 적용:**
  + Kubernetes의 StatefulSet 및 Deployment 생성
  + 서비스 디스커버리와 로드밸런서 설정

yaml

복사편집

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp-container

image: myapp:latest

ports:

- containerPort: 8080

**🔍 3. Blue-Green 및 Rolling Update 전략**

* **Blue-Green 배포:**
  + Blue 버전: 현재 서비스 중인 버전
  + Green 버전: 신규 배포된 버전
  + 테스트 완료 후 로드 밸런서를 Green으로 전환
* **Rolling Update:**
  + Kubernetes의 Rolling Update 기능으로 Pod 단위 업데이트
  + 한 Pod씩 업데이트하여 가용성을 유지

**🔍 4. 장애 대응 및 롤백 전략**

* **ArgoCD Rollback:**
  + 신규 배포가 실패할 경우 이전 버전으로 자동 롤백
  + Kubernetes Deployment Revision을 활용
* **Auto-Recovery:**
  + Prometheus에서 메트릭 수집 후 장애 탐지 시 자동으로 복구
  + SLA 위반 시 Slack 및 PagerDuty에 알림
* **Disaster Recovery:**
  + AWS S3에 주기적으로 상태 저장 및 복원
  + 데이터 손실 방지를 위한 이중화 구성

**🔍 5. 모니터링 및 로깅 설계**

* **Prometheus & Grafana:**
  + CPU, Memory, Disk 사용량 실시간 모니터링
  + TPS, 에러율, 응답 시간 대시보드 제공
* **ELK Stack:**
  + Logstash → Elasticsearch → Kibana
  + 실시간 API 호출 기록, 오류 로그 분석
* **AlertManager 설정:**
  + CPU 사용량 80% 초과 시 알림
  + 메모리 사용량 70% 초과 시 알림

**🔍 6. 설계 다이어그램 예시**

* **CI/CD 아키텍처 다이어그램:**
  + Developer → GitLab → Jenkins → Docker Build → ArgoCD → Kubernetes
  + Prometheus → Grafana → 실시간 모니터링
* **Blue-Green 배포 구조:**
  + Nginx Load Balancer → Blue (Current Version)
  + New Version (Green) → 전환 후 기존 종료

**🔍 7. 테스트 및 검증 방안**

* **CI 테스트:**
  + GitLab Pipeline에서 모든 커밋 시 단위 테스트 자동 실행
  + Maven Test를 통한 코드 검증
* **CD 테스트:**
  + ArgoCD의 상태 변화 실시간 모니터링
  + 배포 후 Kubernetes Pod 상태 확인
* **무중단 배포 테스트:**
  + Blue-Green 전환 시 사용자 요청 누락 여부 확인
  + 실제 트래픽 발생 시 안정성 검증

**✅ 다음 진행 계획:**

이제 **13. 데이터 분석 및 리포팅 전략**을 작성하겠습니다.

* 데이터 분석 개요
* 분석 환경 설계 및 BI 포털 구성
* 데이터 리포팅 및 실시간 대시보드
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 데이터 분석 구조 정의 및 BI 설계
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 13. 데이터 분석 및 리포팅 전략**

**13.1 데이터 분석 개요**

NH 금융지주의 통합 데이터 플랫폼은 \*\*OLAP, BI 포털, 빅스퀘어(Big Square)\*\*를 통해 실시간 분석 및 리포팅이 가능하도록 설계되었습니다.  
모든 금융 데이터는 분석 목적에 맞게 실시간으로 집계되며, 통계 분석, 예측 모델링, 고객 행동 분석 등에 활용됩니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **OLAP 분석** | - 다차원 분석을 통한 금융 거래 현황, 고객 정보, 마케팅 성과 분석 | Tableau, PowerBI, Apache Kylin |
| **BI 포털 구성** | - 실시간 데이터 조회, 시각화 및 리포팅 - 캠페인 성과 분석 및 리포트 생성 | Spring Boot, Vue.js, Redis |
| **대시보드 설계** | - 거래 내역, 고객별 금융상품 선호도, 지역별 금융 현황 대시보드 | Grafana, Kibana, Prometheus |
| **실시간 데이터 분석** | - 금융 거래 발생 시 실시간 데이터 분석 및 시각화 | Apache Spark, Kafka, HDFS |
| **머신러닝 모델 운영** | - 고객 세분화, 신용평가, 예측 모델링을 실시간 분석으로 제공 | TensorFlow, PyTorch, MLflow |

**13.2 세부 내용**

* **OLAP 분석 환경 구성:**
  + Tableau와 PowerBI를 활용한 실시간 분석 및 대시보드 구성
  + 고객 거래 이력, 상품 선호도, 캠페인 성과 등을 다차원 분석
* **BI 포털 설계:**
  + 통합 데이터 레이크(Data Lake)와 연동하여 실시간 조회
  + 마케팅 캠페인 성과, 타겟 고객 분석, 거래 내역 분석 리포팅
* **실시간 분석 구조:**
  + Kafka → Spark Streaming → HDFS → Tableau
  + 금융 거래 발생 시 실시간 분석을 통해 대시보드 업데이트
* **머신러닝 모델 학습 및 운영:**
  + Spark ML 및 TensorFlow를 활용하여 예측 모델 학습
  + 고객 세분화, 이탈 예측, 이상 거래 탐지

**13.3 설계 및 구현 방안**

**🔍 1. OLAP 및 BI 포털 설계**

* **OLAP 구성:**
  + Oracle DB 및 HDFS에 적재된 데이터를 분석용 DB로 전환
  + Tableau와 PowerBI로 시각화 대시보드 구성
  + 주요 분석 항목:
    - 고객 유형별 거래 내역
    - 상품별 판매 성과
    - 지역별 금융 서비스 이용 현황
* **BI 포털 설계:**
  + Vue.js와 Spring Boot를 활용한 웹 포털
  + 실시간 데이터 조회, 분석 리포트 생성
  + 예를 들어, "신규 가입 고객의 대출 신청 현황"을 실시간 조회
* **대시보드 구성 예시:**
  + **대시보드 1:** 실시간 금융 거래 추적
  + **대시보드 2:** 고객 행동 분석 및 이탈율 예측
  + **대시보드 3:** 상품별 판매 성과 리포트

**🔍 2. 실시간 데이터 분석 설계**

* **Kafka & Spark Streaming:**
  + Kafka에 전송된 금융 거래 데이터를 실시간으로 Spark에서 분석
  + Apache Kylin을 통해 OLAP 큐브 구성
* **실시간 분석 시나리오:**
  + **Case 1:** 고객이 카드 결제 시 → 1초 이내로 대시보드 업데이트
  + **Case 2:** 대출 신청 발생 시 → 고객 리스크 평가 모델 실행
  + **Case 3:** 예적금 만기 도래 시 → 마케팅 캠페인 트리거
* **데이터 적재 구조:**
  + 실시간 분석 → Spark SQL → BI 포털
  + 비정형 데이터는 HDFS에, 정형 데이터는 Oracle DB에 저장

**🔍 3. 머신러닝 모델 운영 설계**

* **모델 학습:**
  + TensorFlow 및 PyTorch 기반으로 모델 학습
  + 고객 행동 분석, 금융 리스크 평가, 마케팅 타겟팅 모델 개발
* **모델 배포:**
  + MLflow를 활용하여 모델 배포 및 관리
  + Spark Streaming과 연계하여 실시간 예측
* **예측 모델 예시:**
  + **신용평가 모델:** 고객의 신용 등급 예측
  + **이탈 예측 모델:** 고객의 서비스 이탈 확률 분석
  + **상품 추천 모델:** 금융 거래 패턴 분석을 통한 맞춤형 상품 추천

**🔍 4. 설계 다이어그램 예시**

* **데이터 분석 아키텍처:**
  + Kafka → Spark Streaming → Data Lake (HDFS) → OLAP DB
  + OLAP DB → Tableau, PowerBI → 실시간 대시보드
* **BI 포털 구조:**
  + Spring Boot → Vue.js → Redis Cache → API Gateway
* **모델 배포 아키텍처:**
  + Spark ML → MLflow → Kubernetes Cluster

**🔍 5. 테스트 및 검증 방안**

* **OLAP 분석 검증:**
  + PowerBI 대시보드에 실시간 데이터가 반영되는지 테스트
  + OLAP 큐브의 집계 정확도 검증
* **BI 포털 테스트:**
  + 실시간 거래 내역 조회 시 2초 이내 반응 속도 확인
  + 권한별 접근 제어 및 로깅 처리 확인
* **모델 운영 검증:**
  + Spark ML 모델의 실시간 예측 정확성 검증
  + TensorFlow 모델 학습 시 학습 손실 및 예측 성능 평가

**✅ 다음 진행 계획:**

이제 **14. 운영 및 유지보수 전략**을 작성하겠습니다.

* 운영 관리 개요
* 장애 대응 및 복구 시나리오
* 보안 업데이트 및 패치 관리
* 데이터 백업 및 복구 전략
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 운영 관리 구조 정의 및 복구 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 14. 운영 및 유지보수 전략**

**14.1 운영 관리 개요**

NH 금융지주의 통합 데이터 플랫폼은 **24/7 무중단 운영**을 목표로 설계되었습니다.  
모든 금융 데이터를 안정적으로 처리하고, 장애 발생 시 즉각적인 복구가 가능하도록 **Active-Standby 이중화 구조** 및 **DR(Disaster Recovery) 전략**을 적용합니다.  
또한, **보안 패치, 버전 업데이트, 데이터 백업**이 자동화된 프로세스에 의해 관리됩니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **무중단 운영 구조** | - Active-Standby 이중화 구성 - Zero-Downtime 배포 | Kubernetes, Docker, ArgoCD |
| **장애 탐지 및 복구** | - Prometheus 실시간 모니터링 - 장애 탐지 시 Auto-Recovery 실행 | Prometheus, Grafana, AlertManager |
| **백업 및 복구 전략** | - S3 Glacier를 활용한 주기적 백업 - 매주 전체 백업, 매일 증분 백업 실행 | AWS S3, Oracle RMAN |
| **보안 패치 및 업데이트** | - Spring Boot, Tomcat, Kafka, DataStage의 최신 패치 관리 | Jenkins, ArgoCD, GitOps |
| **모니터링 및 로깅** | - Grafana 대시보드 및 Kibana 로그 모니터링 | Grafana, Kibana, ELK Stack |

**14.2 세부 내용**

* **Active-Standby 이중화 구성:**
  + 주요 데이터베이스(Oracle, MySQL)와 애플리케이션(Spring Boot)은 Active-Standby 구조로 구성
  + 장애 발생 시 Standby 노드가 즉시 서비스로 전환
* **DR(Disaster Recovery) 전략:**
  + 주 데이터센터 장애 시 DR센터로 15분 내 복구 (RPO: 15분, RTO: 30분)
  + AWS S3에 매일 증분 백업, 주 단위 전체 백업 실행
* **무중단 배포:**
  + ArgoCD를 활용한 Kubernetes의 롤링 업데이트
  + 블루-그린 배포 전략으로 중단 없이 신규 버전 배포
* **보안 패치 관리:**
  + Jenkins와 ArgoCD를 활용한 최신 보안 패치 자동화
  + 주기적인 CVE(Critical Vulnerability Exposure) 검사

**14.3 설계 및 구현 방안**

**🔍 1. 무중단 운영 및 복구 설계**

* **Kubernetes 클러스터 이중화:**
  + 각 애플리케이션(Pod)은 이중화되어 Active-Standby 구성
  + Prometheus에서 헬스 체크 후 이상 탐지 시 즉시 Auto-Recovery 실행
* **Oracle Data Guard를 통한 DB 이중화:**
  + 실시간 복제(Replication)를 통해 장애 시 자동 전환
  + Point-in-Time Recovery 지원

**🔍 2. 장애 탐지 및 대응 설계**

* **Prometheus & Grafana 모니터링:**
  + CPU 사용량, 메모리, Disk IO, 네트워크 트래픽 실시간 모니터링
  + SLA 위반 시 AlertManager가 Slack 및 PagerDuty에 알림 전송
* **자동 복구 시나리오:**
  + 서비스 장애 시 Prometheus → AlertManager → Kubernetes Auto-Recovery
  + 3회 연속 장애 발생 시 ArgoCD를 통한 롤백 실행
* **Kubernetes Self-Healing:**
  + Pod 장애 시 자동 재시작
  + 노드 장애 시 다른 노드로 스케줄링 전환

**🔍 3. 데이터 백업 및 복구 전략**

* **S3 Glacier를 활용한 백업:**
  + 매일 증분 백업, 매주 전체 백업
  + 백업 데이터는 5년간 보관 후 안전하게 삭제
* **Oracle RMAN을 통한 복구:**
  + 장애 발생 시 최단 시간 내 복원
  + 복구 목표 (RPO): 15분, (RTO): 30분
* **백업 자동화 시나리오:**
  + DataStage ETL 완료 시 자동 백업 트리거
  + Jenkins Pipeline을 통해 주기적인 스냅샷 생성

**🔍 4. 보안 업데이트 및 패치 관리**

* **자동화된 패치 관리:**
  + Jenkins Pipeline에 보안 패치 테스트 및 배포 자동화
  + CVE 스캔 및 보안 업데이트 알림
* **Spring Boot, Tomcat, Kafka:**
  + 주기적인 CVE 체크와 자동화된 패치 적용
  + ArgoCD를 통한 무중단 배포

**🔍 5. 설계 다이어그램 예시**

* **운영 및 유지보수 아키텍처:**
  + Kubernetes Cluster → Active-Standby 구조
  + Prometheus & Grafana → 실시간 모니터링
  + ArgoCD → 블루-그린 배포 및 롤링 업데이트
* **DR 및 백업 구조:**
  + Primary DB → Standby DB → S3 Glacier Backup
  + 장애 시 DR 센터로 15분 내 복구

**🔍 6. 테스트 및 검증 방안**

* **무중단 배포 테스트:**
  + ArgoCD 배포 후 서비스 중단 여부 확인
  + 실시간 요청 시 응답 누락 없는지 테스트
* **재해 복구 테스트:**
  + DR 센터로 전환 시 데이터 손실 없이 복구되는지 검증
  + RPO, RTO 기준 내 복구 완료 여부 확인
* **패치 관리 테스트:**
  + CVE 패치 후 정상 동작 여부 검증
  + Jenkins Pipeline에서 자동화 테스트 통과 확인

**✅ 다음 진행 계획:**

이제 **15. 최적화 및 성능 개선 전략**을 작성하겠습니다.

* 최적화 전략 개요
* 성능 최적화 설계
* DB 및 애플리케이션 튜닝
* 스케일링 및 확장 전략
* **설계 및 구현 방안**

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 최적화 구조 정의 및 튜닝 전략
* 설계 및 구현 방안

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

**📌 15. 최적화 및 성능 개선 전략**

**15.1 최적화 전략 개요**

NH 금융지주의 통합 데이터 플랫폼은 금융 거래의 실시간 처리와 대규모 데이터 분석을 위해 **성능 최적화**가 필수적입니다.  
최적화 전략은 **애플리케이션 레벨**, **데이터베이스 레벨**, **네트워크 레벨**, **인프라 레벨**로 구분하여 단계별 튜닝을 수행합니다.  
또한, **오토스케일링, 캐싱, 분산 처리**를 통해 확장성을 극대화하고, 시스템 부하에 유연하게 대응할 수 있는 구조를 설계합니다.

| **항목** | **설명** | **기술 스택** |
| --- | --- | --- |
| **애플리케이션 최적화** | - Spring Boot와 Tomcat의 메모리 최적화 - 비동기 처리 및 Redis 캐싱 | Spring Boot, Redis, Kafka |
| **DB 최적화** | - Index 튜닝, Query 최적화 - Connection Pool 관리 | Oracle DB, MySQL, Hibernate |
| **네트워크 최적화** | - CDN, 로드밸런싱, HTTP2.0 적용 | Nginx, HAProxy, CloudFront |
| **인프라 최적화** | - Kubernetes 오토스케일링 - Node Auto-Recovery 설정 | Kubernetes, Docker |
| **캐싱 및 분산 처리** | - Redis Cache를 활용한 읽기 성능 개선 - Kafka를 통한 이벤트 스트림 처리 | Redis, Kafka, Spark Streaming |

**15.2 세부 내용**

* **애플리케이션 최적화:**
  + Spring Boot 애플리케이션에서 메모리 관리 최적화 (Xmx, Xms 조정)
  + 비동기 처리 (@Async)를 활용한 IO 블로킹 최소화
  + Redis를 활용한 캐싱 전략으로 반복 조회 시 속도 개선
* **데이터베이스 최적화:**
  + 인덱스(Index) 최적화: 사용 빈도가 높은 컬럼에 적절한 인덱스 생성
  + Connection Pool 조정: HikariCP를 사용하여 연결 풀 효율화
  + 복잡한 JOIN 쿼리는 View로 대체하여 성능 향상
* **네트워크 최적화:**
  + CDN(Content Delivery Network)을 통해 정적 리소스 로딩 시간 단축
  + Nginx를 활용한 로드밸런싱 설정으로 부하 분산
  + HTTP2.0 프로토콜로 멀티플렉싱 처리
* **인프라 최적화:**
  + Kubernetes Horizontal Pod Autoscaling (HPA) 적용
  + CPU 사용량이 70%를 초과할 시 Pod 자동 생성
  + Prometheus와 연동하여 실시간 리소스 사용량 분석

**15.3 설계 및 구현 방안**

**🔍 1. 애플리케이션 최적화 설계**

* **Spring Boot 비동기 처리:**
  + Kafka Listener를 통해 실시간 메시지 처리
  + @Async 비동기 서비스로 병렬 처리 최적화

java

복사편집

@Async

public CompletableFuture<String> processTransaction(String transactionId) {

// 비동기 처리 로직

transactionService.process(transactionId);

return CompletableFuture.completedFuture("Transaction Completed");

}

* **Redis Cache 적용:**
  + 자주 조회되는 고객 정보 및 금융 상품 정보는 Redis에 캐싱
  + 1일 주기로 캐시 업데이트

java

복사편집

@Cacheable(value = "customerCache", key = "#customerId")

public CustomerDTO getCustomerInfo(String customerId) {

return customerRepository.findById(customerId);

}

**🔍 2. 데이터베이스 최적화 설계**

* **인덱스 최적화:**
  + 조회 빈도가 높은 컬럼에 Composite Index 생성
  + 예시: customer\_id, transaction\_date 조합으로 인덱스 생성

sql

복사편집

CREATE INDEX idx\_customer\_transaction ON transactions (customer\_id, transaction\_date);

* **Connection Pool 최적화:**
  + HikariCP를 통해 최대 Connection 수 관리
  + 최소 5개, 최대 50개 연결로 설정

yaml

복사편집

hikari:

minimumIdle: 5

maximumPoolSize: 50

idleTimeout: 30000

maxLifetime: 1800000

**🔍 3. 네트워크 최적화 설계**

* **로드 밸런싱 구조:**
  + Nginx 및 HAProxy를 활용하여 트래픽 부하 분산
  + 지역별 최적화된 네트워크 라우팅
* **CDN 적용:**
  + AWS CloudFront를 통해 이미지, CSS, JavaScript 등 정적 파일을 캐싱
  + 로딩 시간 30% 단축
* **HTTP2.0 활성화:**
  + 멀티플렉싱을 통한 동시 요청 처리
  + 서버 푸시 기능으로 주요 자원을 선반영

**🔍 4. 인프라 최적화 설계**

* **Kubernetes Autoscaling:**
  + CPU 사용량이 70% 이상일 경우 자동으로 Pod 확장
  + HPA 설정으로 최적화된 Pod 수 유지

yaml

복사편집

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: customer-service-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: customer-service

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

targetAverageUtilization: 70

**🔍 5. 설계 다이어그램 예시**

* **최적화 아키텍처 다이어그램:**
  + Spring Boot → Redis Cache → Oracle DB
  + Kafka → Spark Streaming → Data Lake
  + Nginx → HAProxy → Microservice Cluster

**🔍 6. 테스트 및 검증 방안**

* **부하 테스트:**
  + JMeter를 활용하여 10,000 TPS 부하 테스트 수행
  + 평균 응답 시간 3초 이내 확인
* **DB 최적화 테스트:**
  + 인덱스 생성 전/후 조회 시간 비교
  + HikariCP 설정 변경 후 Connection Pool 사용량 측정
* **네트워크 최적화 테스트:**
  + CDN 설정 후 페이지 로딩 시간 30% 단축 확인
  + HTTP2.0 활성화 시 멀티플렉싱 처리 확인

**✅ 다음 진행 계획:**

이제 최적화 및 성능 개선 전략까지 완료되었으므로, **정리 및 최종 보고서 작성**을 준비하겠습니다.

* 최종 목차 구성 확인
* 누락된 내용 점검 및 보완
* 통합된 아키텍처 다이어그램 생성

**구성은 다음과 같습니다:**

* 도입 전 안내말
* 전체 아키텍처 정리 및 다이어그램 생성
* 최종 보고서 정리

작업을 시작하겠습니다. 잠시만 기다려주세요. 😊

현재까지 진행된 내용은 다음과 같습니다:

**📌 아키텍처 제안서 구성 및 작성 현황**

1. **프로젝트 개요**
   * 대상 계열사 정의: NH농협은행, NH농협카드, NH투자증권, NH생명보험, NH손해보험
   * 시스템별 주요 기능 정의: 데이터 수집, 처리, 저장, 분석, 마케팅 활용 등
2. **기술 아키텍처 (TA)**
   * 인프라 환경 정의
   * HW, SW, 인력 계획
   * 예산 계획
   * 스토리지 및 임시 보관 영역 설계
   * 설계 및 구현 방안
3. **애플리케이션 아키텍처 (AA)**
   * 시스템 연계 및 수집 프로세스
   * 개인화 마케팅 분석 및 실행 구조
   * 데이터 전송 및 암호화 설계
   * 통합고객번호 채번 및 관리
   * 배치 로깅 및 실시간 연계 서비스
   * 금융지주 OLAP 환경 설계
   * 설계 및 구현 방안
4. **비즈니스 아키텍처 (BA)**
   * 개인화 마케팅 서비스 설계
   * 데이터 수집 및 활용 프로세스
   * 데이터 전처리 및 임시 보관
   * 목적별 마트 구성 및 고객번호 관리
   * 데이터 분석 및 모델 운영
   * 설계 및 구현 방안
5. **데이터 아키텍처 (DA)**
   * 데이터 주제 영역 정의
   * 원천 데이터 적재 및 목적별 마트 구성
   * 메타 데이터 관리 및 조회
   * 분석용 목적 DB 구성
   * 데이터 보관 주기 및 파기 정책
   * 설계 및 구현 방안
6. **보안 아키텍처 (SA)**
   * 제3자 동의 기반 통합 데이터 인증
   * 전송구간 암호화
   * 전송파일 암호화
   * 개인정보 처리 시 암호화/마스킹/파기
   * 계정 및 권한 관리 체계
   * 설계 및 구현 방안
7. **인터페이스 아키텍처 (IA)**
   * 계열사 데이터 수집 인터페이스 설계
   * 올원뱅크 행동 데이터 수집
   * 데이터 연계 표준 및 프로토콜 설계
   * 분석 플랫폼과의 연계 설계
   * 설계 및 구현 방안
8. **기대효과 및 전략적 가치**
   * 비용 절감 효과
   * 통합 데이터 기반 의사결정 지원
   * 분석 모델을 통한 마케팅 효과 증대
9. **아키텍처 주요 정보**
   * 도입 시스템 정의 (신규 도입 및 기존 시스템 활용)
   * 인터페이스 구성도
   * 데이터 주제 영역 정의
   * 도입 솔루션 목록
10. **배포 및 운영 전략**
    * CI/CD 파이프라인 설계
    * 블루-그린 배포, 롤링 업데이트
    * 장애 복구 및 DR(Disaster Recovery)
    * 실시간 모니터링 및 로깅 관리
    * 장애 대응 및 복구 시나리오
    * 설계 및 구현 방안
11. **로그 및 모니터링 전략**
    * 로그 수집 및 분석 설계
    * 모니터링 아키텍처 설계
    * 장애 탐지 및 알림 체계
    * 설계 및 구현 방안
12. **API 관리 및 거버넌스 전략**
    * API 설계 표준화
    * API 버전 관리 및 확장 전략
    * API 보안 및 접근 제어
    * 설계 및 구현 방안
13. **DevOps 및 CI/CD 전략**
    * DevOps 전략 개요
    * CI/CD 파이프라인 설계
    * 자동화 배포 및 모니터링
    * 장애 복구 및 롤백 전략
    * 설계 및 구현 방안
14. **운영 및 유지보수 전략**
    * 운영 관리 개요
    * 장애 대응 및 복구 시나리오
    * 보안 업데이트 및 패치 관리
    * 데이터 백업 및 복구 전략
    * 설계 및 구현 방안
15. **최적화 및 성능 개선 전략**
    * 최적화 전략 개요
    * 성능 최적화 설계
    * DB 및 애플리케이션 튜닝
    * 스케일링 및 확장 전략
    * 설계 및 구현 방안

**🔍 남은 작업:**

1. **전체 아키텍처 다이어그램 생성**
   * 모든 아키텍처 간 연동과 흐름을 하나의 통합된 다이어그램으로 표현해야 합니다.
2. **최종 보고서 작성 및 정리**
   * 각 섹션별로 누락된 부분이 있는지 점검하고 최종 보고서 형태로 정리합니다.
3. **QA (Quality Assurance) 검토**
   * 아키텍처 상의 보완점, 개선 사항을 체크리스트로 검토합니다.

이제 최종 보고서 작성과 아키텍처 다이어그램 정리만 남았습니다.  
다음 순서로 진행할까요?

1. 아키텍처 통합 다이어그램 작성
2. 최종 보고서 작성
3. QA 체크리스트 작성

진행해도 될까요? 😊

**나의 말:**

네

**ChatGPT의 말:**

**📌 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 다이어그램

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

**🔍 주요 연결 구조 설명**

1. **API Gateway**
   * 모든 외부 및 내부 요청이 API Gateway를 통해 마이크로서비스로 전달됩니다.
   * 인증 및 라우팅이 이루어지며, JWT 및 OAuth2.0을 통한 보안 처리 진행
2. **Microservices**
   * Customer Service, Transaction Service, Campaign Service, Analytics Service로 구성
   * 각 서비스는 독립적으로 배포되며, Kubernetes에 의해 관리됩니다.
3. **Data Lake와 분석 플랫폼**
   * HDFS 및 S3에 데이터를 저장하고, 실시간 스트림(Kafka)을 통해 Spark에서 분석 처리
   * BI 포털(Tableau, PowerBI)을 통해 대시보드 생성
4. **ETL Processing**
   * IBM DataStage를 사용하여 배치 처리 후 Data Lake에 적재
   * 실시간 분석 시 Spark Streaming을 통해 바로 분석 플랫폼으로 전송
5. **보안 및 접근 제어**
   * API Gateway에 보안 인증이 설정되며, 모든 전송은 TLS 1.3 암호화 채널을 사용
   * OAuth2.0과 JWT 인증 방식을 통해 권한 관리
6. **모니터링 및 로깅**
   * Prometheus와 Grafana를 통해 실시간 모니터링
   * ELK Stack으로 모든 시스템 로그 수집 및 분석

**✅ 다음 진행 계획:**

1. **최종 보고서 작성**
   * 지금까지 작성된 모든 내용을 정리하여 최종 문서화
   * 도입 전 안내말, 주요 설계 내용, 시사점 및 기대효과 포함
2. **QA (Quality Assurance) 검토**
   * 전체 아키텍처에 대한 보완 사항 및 개선점 점검
   * 최적화 대상 파악 및 정리

최종 보고서를 작성하겠습니다. 잠시만 기다려주세요. 😊

**📌 최종 보고서 작성 (목차 구성)**

**📍 목차 구성**

**0. 도입 전 안내말**

* 프로젝트 개요
* 목적 및 목표
* 주요 시스템 정의
* 아키텍처 설계의 주요 고려 사항

**1. 기술 아키텍처 (TA)**

* 인프라 환경 정의
* HW, SW, 인력 계획
* 예산 계획
* 스토리지 및 임시 보관 영역 설계
* 설계 및 구현 방안

**2. 애플리케이션 아키텍처 (AA)**

* 시스템 연계 및 수집 프로세스
* 개인화 마케팅 분석 및 실행 구조
* 데이터 전송 및 암호화 설계
* 통합고객번호 채번 및 관리
* 배치 로깅 및 실시간 연계 서비스
* 금융지주 OLAP 환경 설계
* 설계 및 구현 방안

**3. 비즈니스 아키텍처 (BA)**

* 개인화 마케팅 서비스 설계
* 데이터 수집 및 활용 프로세스
* 데이터 전처리 및 임시 보관
* 목적별 마트 구성 및 고객번호 관리
* 데이터 분석 및 모델 운영
* 설계 및 구현 방안

**4. 데이터 아키텍처 (DA)**

* 데이터 주제 영역 정의
* 원천 데이터 적재 및 목적별 마트 구성
* 메타 데이터 관리 및 조회
* 분석용 목적 DB 구성
* 데이터 보관 주기 및 파기 정책
* 설계 및 구현 방안

**5. 보안 아키텍처 (SA)**

* 제3자 동의 기반 통합 데이터 인증
* 전송구간 암호화
* 전송파일 암호화
* 개인정보 처리 시 암호화/마스킹/파기
* 계정 및 권한 관리 체계
* 설계 및 구현 방안

**6. 인터페이스 아키텍처 (IA)**

* 계열사 데이터 수집 인터페이스 설계
* 올원뱅크 행동 데이터 수집
* 데이터 연계 표준 및 프로토콜 설계
* 분석 플랫폼과의 연계 설계
* 설계 및 구현 방안

**7. 기대효과 및 전략적 가치**

* 비용 절감 효과
* 통합 데이터 기반 의사결정 지원
* 분석 모델을 통한 마케팅 효과 증대

**8. 아키텍처 주요 정보**

* 도입 시스템 정의 (신규 도입 및 기존 시스템 활용)
* 인터페이스 구성도
* 데이터 주제 영역 정의
* 도입 솔루션 목록

**9. 배포 및 운영 전략**

* CI/CD 파이프라인 설계
* 블루-그린 배포, 롤링 업데이트
* 장애 복구 및 DR(Disaster Recovery)
* 실시간 모니터링 및 로깅 관리
* 장애 대응 및 복구 시나리오
* 설계 및 구현 방안

**10. 로그 및 모니터링 전략**

* 로그 수집 및 분석 설계
* 모니터링 아키텍처 설계
* 장애 탐지 및 알림 체계
* 설계 및 구현 방안

**11. API 관리 및 거버넌스 전략**

* API 설계 표준화
* API 버전 관리 및 확장 전략
* API 보안 및 접근 제어
* 설계 및 구현 방안

**12. DevOps 및 CI/CD 전략**

* DevOps 전략 개요
* CI/CD 파이프라인 설계
* 자동화 배포 및 모니터링
* 장애 복구 및 롤백 전략
* 설계 및 구현 방안

**13. 운영 및 유지보수 전략**

* 운영 관리 개요
* 장애 대응 및 복구 시나리오
* 보안 업데이트 및 패치 관리
* 데이터 백업 및 복구 전략
* 설계 및 구현 방안

**14. 최적화 및 성능 개선 전략**

* 최적화 전략 개요
* 성능 최적화 설계
* DB 및 애플리케이션 튜닝
* 스케일링 및 확장 전략
* 설계 및 구현 방안

**15. 통합 아키텍처 다이어그램**

* 전체 시스템의 논리적 구조 표현
* 데이터 흐름 및 처리 구조
* 주요 인터페이스 및 연동 구조

**16. 시사점 및 결론**

* 통합 아키텍처의 주요 기대효과
* 금융지주 내 데이터 플랫폼의 전략적 가치
* 향후 개선 및 확장 계획

**✅ 다음 진행 계획:**

1. **각 목차별 작성 진행**
   * 0번부터 16번까지 모든 내용을 최종 보고서 형태로 작성
   * 도입 전 안내말, 설계 상세 내용, 기대효과, 시사점 포함
2. **전체 문서 통합 및 QA 체크리스트 생성**
   * 작성된 모든 내용을 QA 체크리스트로 점검
   * 아키텍처의 개선점, 추가 설계 필요 여부 분석
3. **최종 보고서 완성 및 검토**
   * 완성된 최종 보고서를 검토 및 수정
   * 전략적 제언 및 최종 결론 정리

이제, 목차 순서에 맞춰 **0. 도입 전 안내말**부터 작성하겠습니다.  
잠시만 기다려주세요. 😊