

RTGS 프로토타입 시스템 PoC 테스트 2차 결과보고

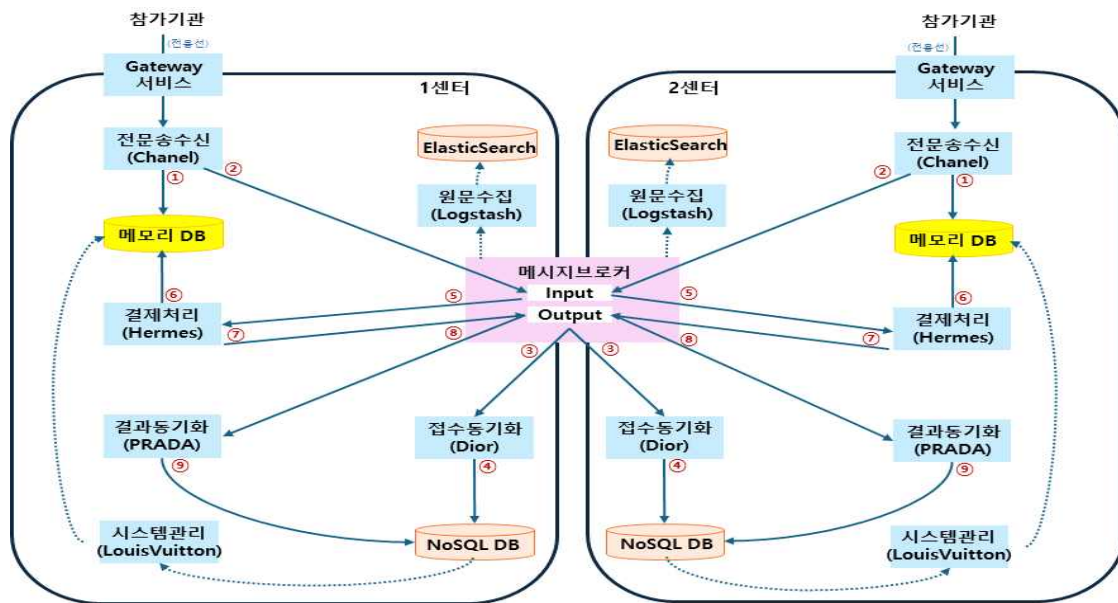
2025. 1. 23.

I. 개요	1
II. 프로토타입 시스템 내용	2
1. 아키텍처 설계	2
2. 업무절차 설계	4
3. 시스템 구축환경 구성	5
4. 프로그램 개발	7
5. 테스트 결과	18
III. 향후 추진계획	21
<별첨1> UML (Unified Modeling Language)	22
<별첨2> 프로토타입 서버 구성	28
<참고1> 쿠버네티스(Kubernetes)	36
<참고2> InfluxDB	37
<참고3> ELK(Elasticsearch, Logstash, Kibana)	38
<참고4> 메시지 브로커(Kafka)	39
<참고5> 인메모리 컴퓨팅(Hazelcast)	40
<참고6> NoSQL(MongoDB)	41

<요약>

□ 전국민이 사용하는 소액결제 업무를 24×365 연중무휴 제공하는 **실시간총액결제 방식의 신속자금이체시스템**(이하 “소액RTGS시스템”)의 기능과 성능 검증을 위한 **프로토타입 시스템을 자체 개발**

- 1 **아키텍처 설계** : 다중 센터의 동시운영(Active-Active) 체제를 목표로, 각 센터에 인프라를 동등하게 배치하여 **단일 클러스터로 구성**하고, 장애에 대비하여 **원격지 센터 간에 데이터 복제가 아닌 분산 저장**
- 2 **업무절차 설계** : 동시다발적으로 송신기관에서 ISO20022전문으로 자금이체를 요청하여 수신기관에 자금이체가 실시간 총액결제 방식으로 처리
- 3 **시스템 구축환경 구성** : **네이버 공공클라우드의 원격지 2개 센터(수도권, 남부권)**를 활용하여 개발 및 테스트 환경을 구성하고, 오픈소스 기반의 솔루션을 사용
- 4 **프로그램 개발** : 자금이체 업무절차를 ① 게이트웨이, ② 전문송수신, ③ 결제처리, ④ 접수동기화, ⑤ 결과동기화, ⑥ 시스템관리 등 6개의 서비스로 구성



5 **테스트 결과** : 실제 참가기관 요청과 유사하게 외부 노드 환경을 구축하여 스트레스 테스트 실시하여 2개 센터 간 계좌잔액, 거래내역 등 데이터 정합성과 성능을 검증

□ **향후 추진 계획** : 시스템 운영에 필요한 전체 거래에 대한 성능 점검을 위해 센터내 서버노드 구성과 솔루션 조합을 변경하면서 다면 테스트(복원력 테스트 포함)를 실시하고 보완을 수행하여 본격적인 시스템 구축사업(2026년~)을 준비

- IT전략국은 금융결제국과 함께 실시간충액결제 방식의 신속자금이체를 24×365 연중무휴로 제공하는 **소액RTGS시스템 구축을 추진중***

* 「RTGS 방식 신속자금이체시스템 구축 종합계획」(결제정책팀-1169, 2023.12.28, 부총재보)
「소액RTGS시스템 구축을 위한 IT부문 기본계획」(결제시스템팀-3, 2024.1.2, 부총재보)

- 일평균 1.1억건 이상의 대량거래를 5초 이내 처리하는 신속성과 특정 IT센터의 전면 가동중단에도 서비스를 지속할 수 있는 수준의 복원력 확보가 주요 목표

- 소액RTGS시스템에 필요한 고성능과 고가용성은 기존의 전통적인 방식의 시스템 아키텍처에는 한계가 있으며, 아직 국내 코어뱅킹 시스템은 IT센터 동시 운영체계(Active-Active) 구현 사례가 부재*

* 「소액RTGS 구축관련 국내사례조사 결과보고」(결제시스템팀-1366, 2023.12.29, 부장)

- RTGS 방식 신속자금이체시스템을 운영중인 유럽, 미국, 호주 등의 중앙은행은 인메모리 기술을 활용하여 순차적으로 거래를 처리하고, 병행처리 방식의 Active-Active를 구현함으로써 처리속도와 복원력을 극대화*

* 「소액RTGS시스템 구축 관련 IT아키텍처 국외사례 조사결과」(RTGS시스템팀-48, 2024.4.29, 부총재보)
「이탈리아 중앙은행과의 정례 회의 결과 보고」(RTGS시스템팀-81, 2024.7.11, 국장)
「유럽 지급결제시스템 기술혁신 현황 및 시사점」(결제정책팀-631, 2024.7.18, 국장)

- 소액RTGS시스템 구성에 필요한 IT요소기술과 비즈니스 요구사항을 분석하고 국내외 사례를 검토하여 **당행의 소액RTGS시스템에 적합한 아키텍처를 수립하고 기술검증을 위한 프로토타입 핵심 프로세스 개발을 완료***

* 「소액RTGS시스템 구축을 위한 기본 아키텍처」(RTGS시스템팀-62, 2024.5.20, 부장)
「소액RTGS시스템 구축 관련 기술검증 방안」(RTGS시스템팀-69, 2024.5.31, 부장)
「소액RTGS시스템 기술검증을 위한 프로토타입시스템 구축계획」(RTGS시스템팀-78, 2024.7.1, 부장)
「소액RTGS시스템 기술검증을 위한 프로토타입 핵심프로세스 개발보고」(RTGS시스템팀-99, 2024.8.23, 부장)

⇒ 지금까지 설계한 소액RTGS시스템의 핵심 프로세스를 기반으로 전체 프로세스에 대한 프로토타입 시스템을 자체개발하여 **센터간 Active-Active 운영에 대한 기능과 성능 1차 테스트를 완료**

II

프로토타입 시스템 내용

1

아키텍처 설계

- 원격지 센터 간에 Active-Active 체제로 소액RTGS시스템의 운영을 검증하기 위하여 솔루션 부문과 전문 송수신과 결제처리를 수행하는 프로그램 개발 부문으로 구분하여 설계한 기존의 기본 아키텍처를 확장

검증대상 구성요소 변경

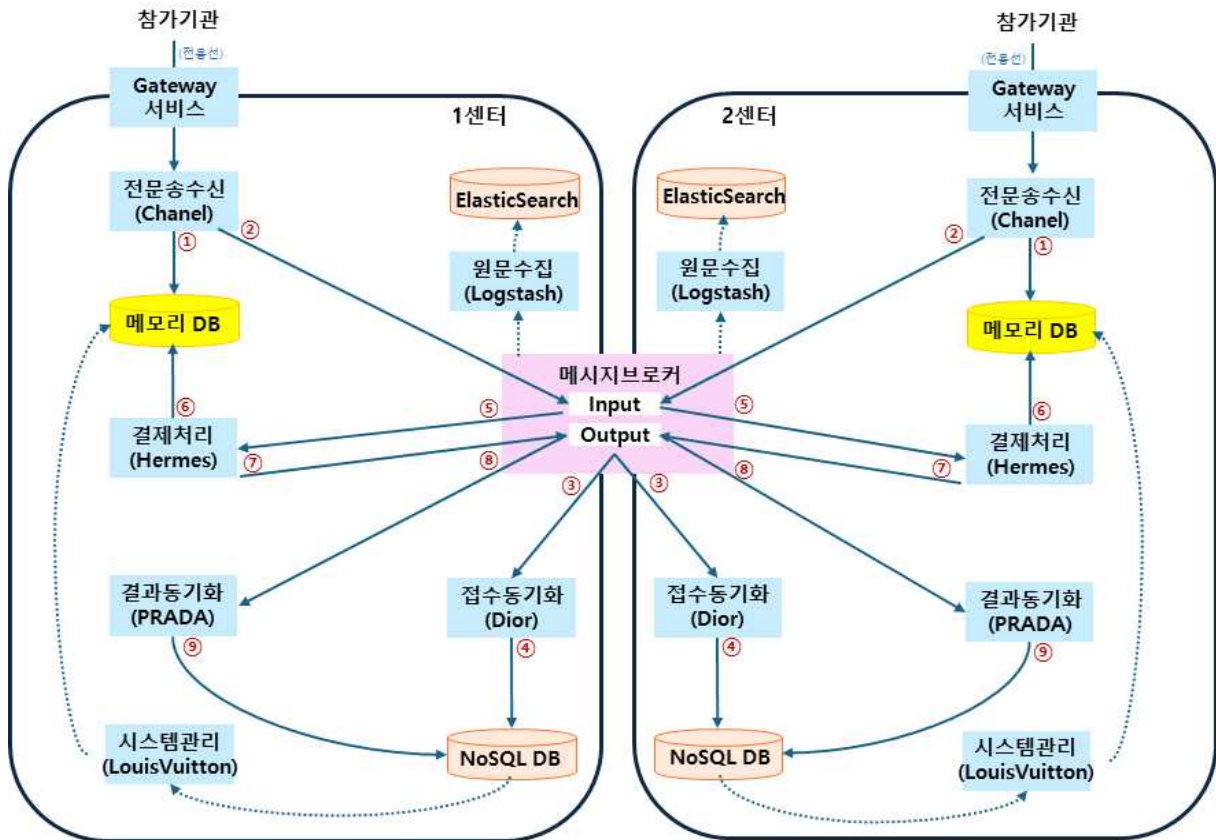
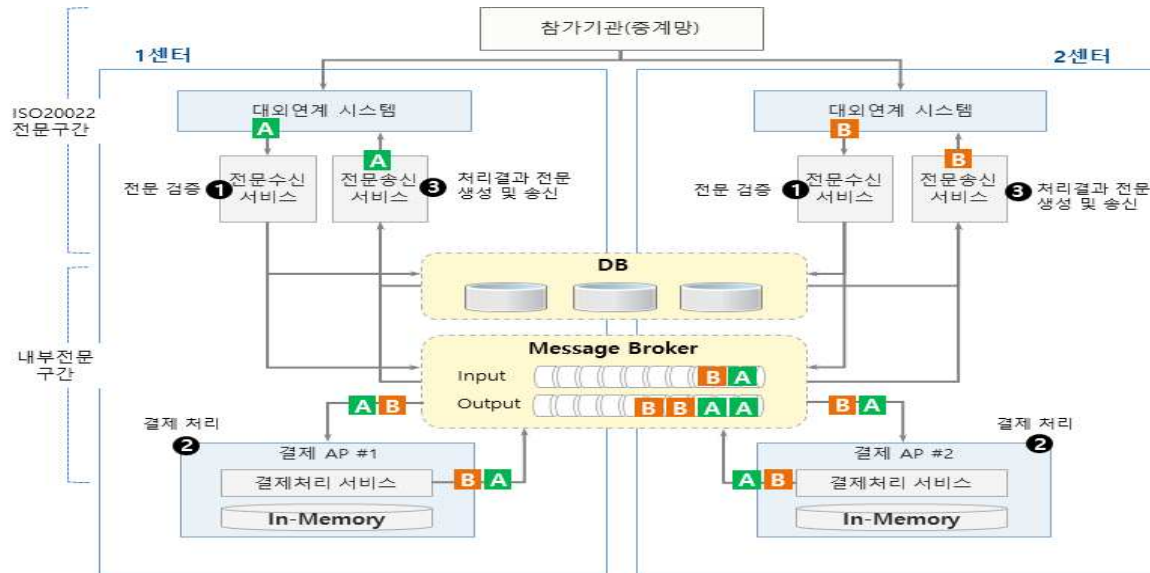
구 분	구 성 요 소	기 능
솔루션	대외연계	참가기관과의 전문(ISO20022) 송수신(이체 요청접수, 결과송부)
	데이터베이스	참가기관과 송수신 전문원장 저장(IT센터간 중복체크 및 동기화)
	메시지 브로커	IT센터간 전문 직렬화 처리(단계별 작업내역 큐에 저장)
	인메모리 저장소	계좌잔액, 기관정보 등을 저장, 신속결제처리
프로그램	전문수신	전문을 파싱, 검증, 변환하고 메시지 브로커로 전달(병렬처리)
	전문송신	메시지 브로커의 처리 결과에 따라 송신 전문 생성(병렬처리)
	결제처리	인메모리 저장소를 통해 결제요청 처리(순차처리)



구 분	구 성 요 소	기 능
솔루션	데이터베이스	이체 내역 및 참가기관의 원장 영구 저장
	메시지 브로커	IT센터간 프로세스 동기화(단계별 작업내역 큐에 저장)
	인메모리 저장소	기관별 원장, 이체 내역 등을 임시 저장, 신속결제처리
	코드 관리	코드 리뷰, 협업, CI/CD를 위한 코드 형상 관리
	이미지 관리	컨테이너 이미지를 저장, 관리, 배포할 수 있는 레지스트리
	컨테이너 관리	워크로드를 자동으로 배포, 확장, 운영하기 위한 오케스트레이션
	로그 관리	로그 수집, 검색, 분석, 시각화를 통합 제공하는 데이터 처리
	모니터링 관리	성능과 오류 추적을 실시간으로 수행, 안정적인 운영 지원
	키 관리	Secrets 관리, 데이터 암호화 및 인증 관리를 제공, 보안 제고
프로그램	대외연계	인증 정책 적용 및 서비스 라우팅 인터페이스
	전문송수신	ISO20022전문을 송신하고 처리 결과를 응답
	결제처리	인메모리 저장소를 통해 결제요청 처리(순차처리)
	수신동기화	센터간 요청 내역을 임시 저장소와 영구 저장소에 동기화
	처리동기화	센터간 결제 처리 결과를 영구 저장소에 동기화
	관리자기능	운영시 필요한 편의 기능 제공

- 다중의 IT센터에 동등하게 배치하여 동시 운영을 원칙으로, 각 센터 간의 정합성 보장을 위해 메시지 브로커로 단일 클러스터를 구성하고 장애에 대비하여 데이터 사본을 분산 저장하는 개념

검증대상 아키텍처 개념도



2 업무절차 설계

□ 검증을 위한 기본 아키텍처에서는 하나의 자금이체 거래를 여러 세부 서비스 처리단계로 분리하여 독립적으로 처리함으로써 병목현상을 축소

○ 2개의 IT센터에서 동시다발적으로 이루어지는 거래에 대한 센터 간 계좌 잔액, 거래내역 등 데이터 정합성과 성능을 시험하고 실제 기관 요청과 유사하게 외부 노드 환경을 구축하여 테스트 실시

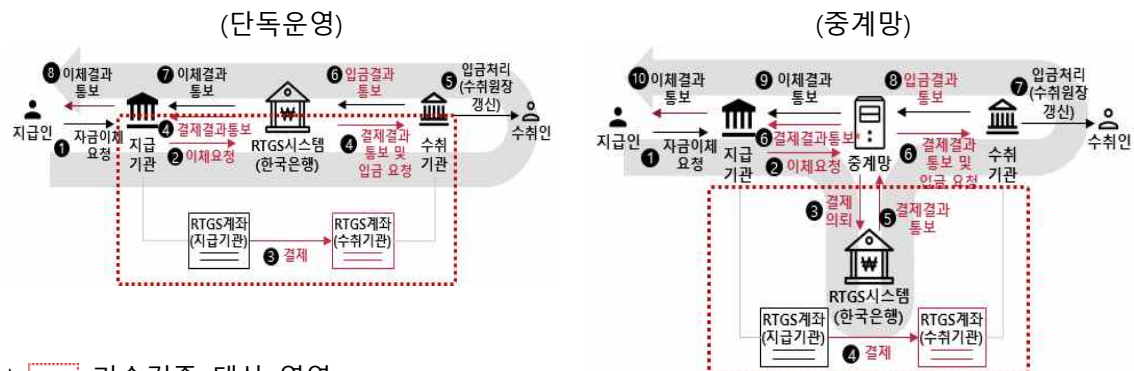
프로토타입 시스템 기술검증 목표

구분	시나리오
부하 모델	송신기관, 수신기관을 랜덤하게 선정하여 자금 이체 요청 ISO20022전문을 실시간 생성, 다수의 노드에서 부하를 발생하여 스트레스 테스트
성능	적정부하 및 최대부하 환경에서 처리량, 응답시간, 자원사용률 확인 서버 노드 및 센터 간 데이터 정합성 검증(순서, 잔액, 건수 대사)
복원력	적정부하 환경에서 노드 중지(프로세스 강제종료, 네트워크 절체) 중지 노드 재기동(중지 노드수를 늘려가며 반복 테스트)

□ 지급결제 거래과정 중 핵심 영역인 전문송수신, 결제처리, 센터간 동기화를 대상으로 구현

○ RTGS시스템 구축 모델에 따른 중계기관의 유무와 상관없이 결제의뢰 전문을 처리하는 단계는 동일하므로 어떤 모델로 구축하더라도 기술 검증 결과를 활용 가능

구축 모델별 전문 흐름



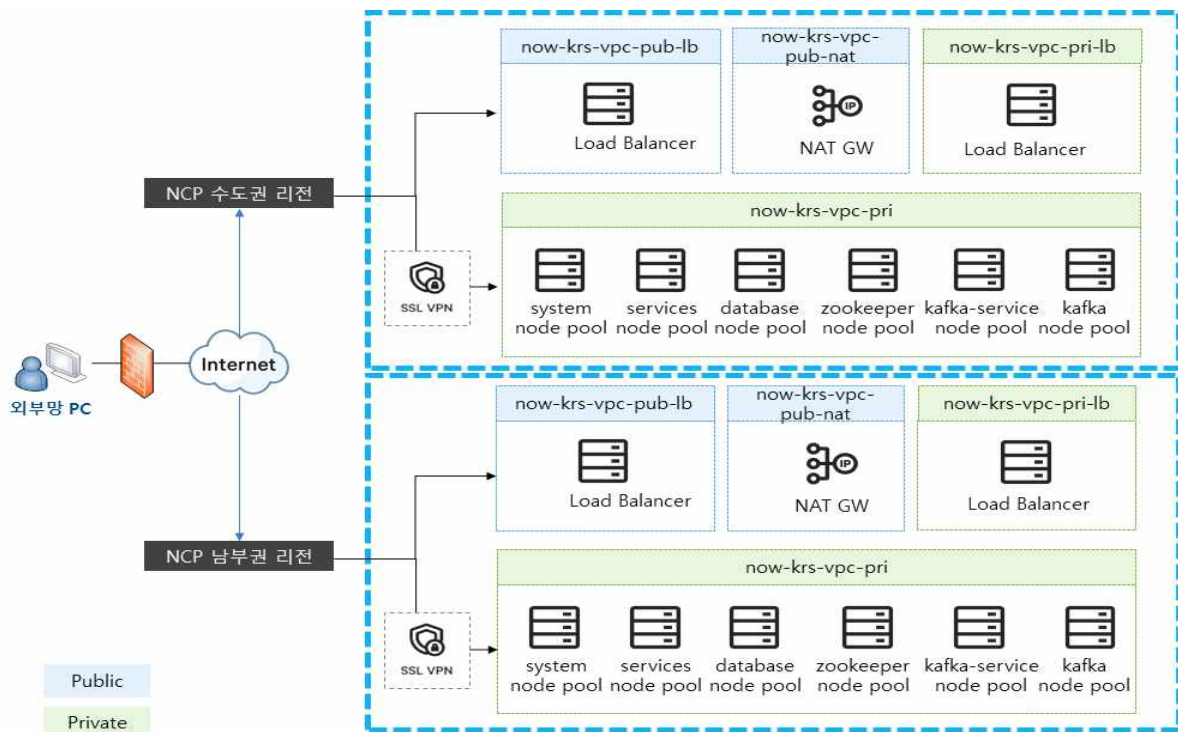
* 기술검증 대상 영역

※ 상세한 설계 내역은 <별첨1> UML(Unified Modeling Language) 참조

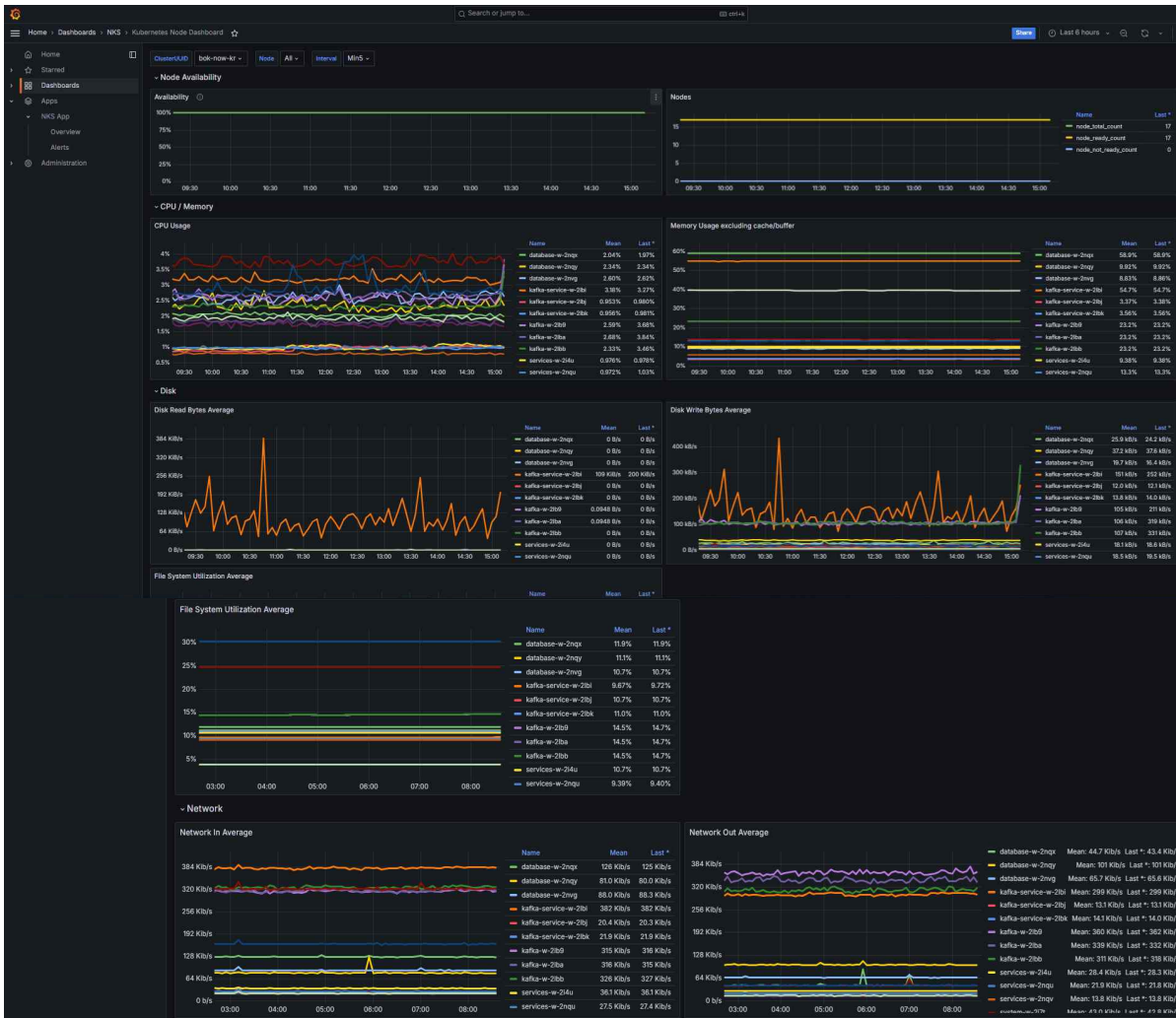
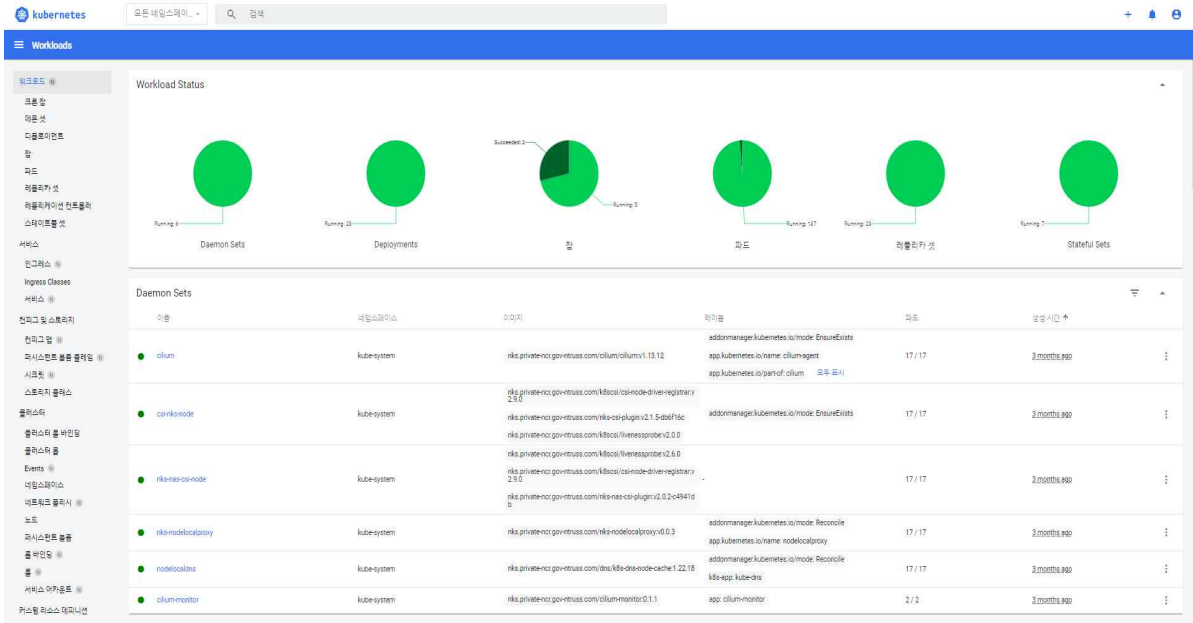
3 시스템 구축환경 구성

- 시스템 확장성, 운영 효율성, 비용 최적화 등을 고려하여 기존 가상서버 (VM) 기반 운영 환경에서 **쿠버네티스 기반 환경으로 전환을 추진**
 - 확장성(Scalability) : 컨테이너 수를 자동으로 조절해 트래픽 급증 상황에 탄력적으로 대응가능하고, 사용량만큼 확장/축소로 자원 활용이 최적화
 - 운영 효율성 : CI/CD 파이프라인과 연계함으로써 배포 자동화 및 빈번한 릴리스 지원, 노드나 컨테이너 장애 시 쿠버네티스 스스로 재시작·재할당으로 가용성을 보장
 - 비용 최적화 : 서비스 사용량에 맞춰 컨테이너 단위로 자원을 할당하여 불필요한 인스턴스 운영 비용을 절감하고, 거래 증가 시에 수평 확장 운영
 - 표준화된 환경 : 컨테이너 이미지를 통해 테스트, 스테이징, 프로덕션 환경을 동일하게 유지
 - 보안 및 관리 용이성 : 컨테이너 보안과 네트워크 정책을 중앙 집중적으로 설정관리하며 서비스별 격리 수준을 높여 보안 위협이 발생하더라도 파급 범위를 최소화

프로토타입 시스템 구축환경



클라우드 환경 시각화 대시보드



※ 상세한 설계 내역은 <별첨2> 프로토타입 서버 구성(쿠버네티스 구성 현황) 참조

4 프로그램 개발

개발 환경구성

- 자체 개발 응용 서비스 프로그램은 회계결제시스템에서 사용중인 Java와 동일한 JVM 기반 언어인 Kotlin*을 사용하여 구현

* Java와 Kotlin은 프로젝트내 동시 사용 가능하고 IDE에서 코드 자동 변환 기능을 제공함

- 개발 생태계 및 대규모 데이터 처리에 적합한 오픈소스 프레임워크인 스프링부트(Spring Boot)*를 이용

* Java 프레임워크인 Spring을 기반으로 웹 어플리케이션과 마이크로 서비스를 개발하는 도구

- 핵심 프로세스 구현을 위한 솔루션*으로 메시지 브로커(Kafka Confluent), NoSQL(MongoDB), In-Memory Data Grid(Hazelcast), ELK stack 채택

* 솔루션은 무료로 사용 가능한 커뮤니티 라이선스를 이용하고 최신 안정화 버전을 설치

프로토타입 시스템 프로그램 개발환경

구분	세부 내용
프레임워크	SpringBoot 3.3.4
개발 언어	Kotlin 2.0.20
JDK	Eclipse Temurin (AdoptOpenJDK HotSpot) 21.0.5
빌드	Gradle 8.10.2 / SourceBuild
개발환경 도구(IDE)	IntelliJ IDEA Ultimate 2024.3.1.1 / DataGrip 2024.3.3
API Test	Postman 11.27.3
로컬 개발환경	Docker Compose 2.32.3 / Docker Desktop
데이터베이스	MongoDB Community 6.0.5
인메모리 데이터그리드	Hazelcast 5.5.0
Message Broker	Kafka Confluent 7.7.0
ELK stack	elasticsearch 8.15.2/ logstash 8.15.3 / kibana 8.15.2
부하 Test	k6 0.56.0 / influxDB 1.9 / grafana 11.4.0
형상 관리	Git (Branch Strategy: Gitflow) / SourceCommit
Source code repository	https://devtools.gov-ncloud.com/10368/RetailRTGS-demo.git
image registry	Container Registry(Ncloud)
container orchestration	kubernetes 1.28.10-nks.2 / k9s
UML 설계	Plant UML

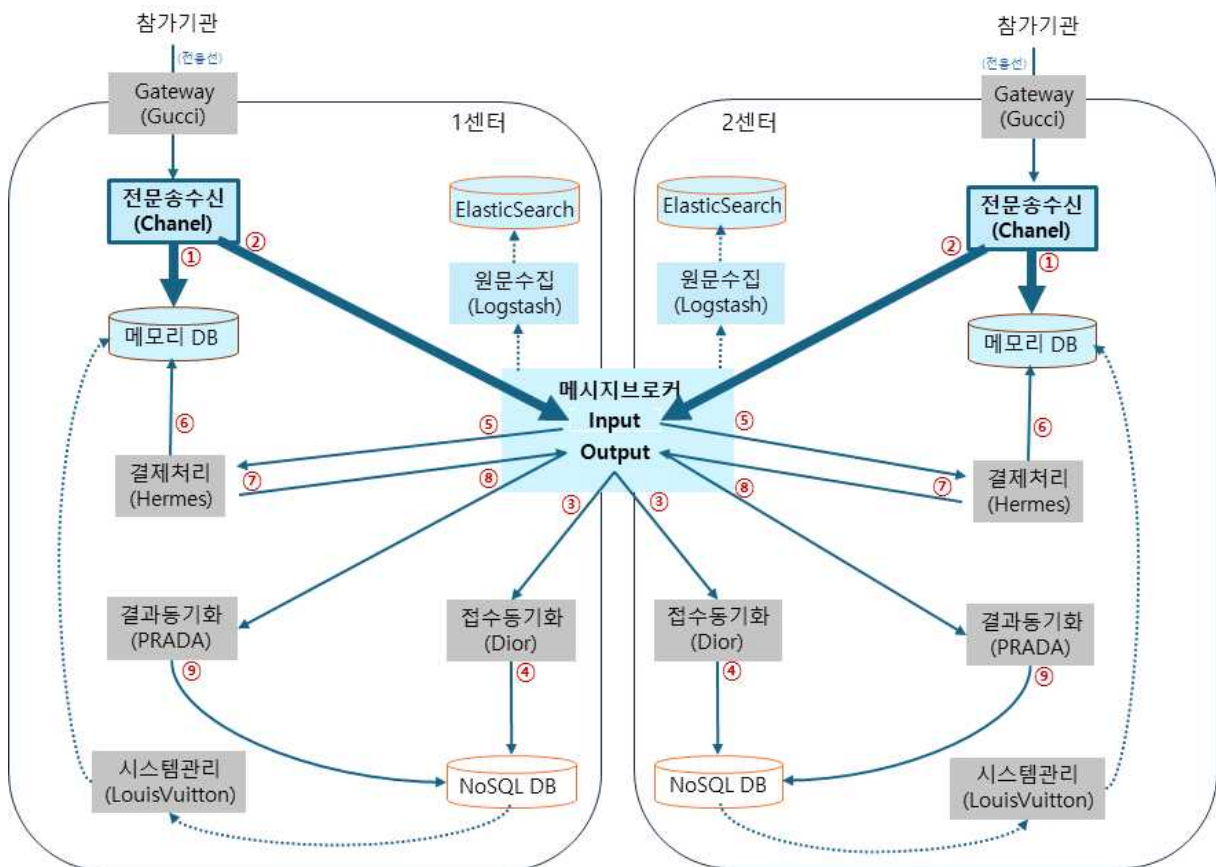
※ 테스트 환경 구성 내역은 <별첨2> 프로토타입 서버 구성(환경 구성 리소스 파일) 참조

전문 송수신 서비스

□ CHANEL(Contact Hub for Advanced Networked Entry Liaison) 참가기관(송신)으로부터 거래요청 건을 접수받아 참가기관(송신, 수신)에게 처리 결과를 통보하는 서비스

- ① HTTP Request Body의 ISO20022 전문(pacs.008) 파일(XML)을 파싱(Parsing)
- ② XSD(XML Schema Definition) 스키마를 이용하여 XML 파일의 유효성 검사
- ③ 요청의 전문 원본은 ElasticSearch에 별도로 저장
- ④ 결제에 필요한 필수항목을 파싱하여 객체로 생성(4.34KB → 75Bytes)
- ⑤ 메모리DB 저장 및 센터간 요청 동기화를 위해 메시지 브로커(Kafka)로 전송

전문송수신 서비스



전문송수신 프로그램

구분		소스코드	기능요약
chanel		ChanelApplication	전문송수신 응용시스템 초기화 실행
	application	ChanelService	전문송수신 서비스(송금 요청 처리, 검증 및 Kafka를 통한 메시지 전송을 수행)
	domain	Transfer	자금이체 엔티티(송금) 데이터를 관리하는 도메인 모델과 서비스 레이어 정의)
	infrastructure	ChanelExceptionHandler	전문송수신 예외처리(REST API의 예외를 일관된 방식으로 처리하여 JSON 형태의 오류 응답을 반환)
	config	ChanelConfig	전문송수신 환경설정
		HazelcastConfig	인메모리 환경설정
		KafkaProducerConfig	메시지브로커 프로듀서 환경설정
	properties	HazelcastProperties	인메모리 DB 환경설정
	presentation	BmiGenerator	거래고유식별자 전문번호(BMI) 생성(날짜, 기관코드, 일련번호, 시간, 난수를 조합하여 고유한 거래 식별자를 생성)
		XmlValidator	XML 유효성 체크(ISO20022 pacs.008의 XSD(스키마) 기반 유효성 검사)
	controller	ChanelController	전문송수신 제어 처리(송금 요청 및 상태 조회 API를 처리)
	dto	RequestEnvelope	요청 처리 : Jackson XML을 사용하여 XML을 객체로 변환(Deserialization)하고, 반대로 객체를 XML로 변환(Serialization)
		ResponseEnvelope	응답 처리 : 송금상태보고서 전문을 XML 형식(pacs.002)으로 표현
	resources	application.yml	응용시스템 환경설정(응용시스템 기본 속성, 프로필, 관리 엔드포인트, 메트릭 수집 및 외부 설정 파일 로딩)
		application-hazelcast.yml	인메모리 응용시스템 환경설정(환경에 따라 다른 Hazelcast 클러스터에 연결)
		application-kafka.yml	메시지브로커 응용시스템 환경설정(Kafka 클라이언트의 프로듀서 및 컨슈머 설정)
		logback-spring.xml	로그관리 : Logback을 사용하여 로그를 Logstash로 전송하고, 개발 환경에 따라 다른 Logstash 서버로 전송
	iso20022 validation	head.001.001.03.xsd	ISO20022 전문 헤더(BAH) 형식 체크
		pacs.002.001.13.xsd	ISO20022 전문(응답) 본문 형식 체크
		pacs.008.001.11.xsd	ISO20022 전문(요청) 본문 형식 체크
		pacs008template.xsd	ISO20022 전문(요청) 본문 체크 템플릿

접수 동기화 서비스

□ DIOR(Data Integration and Organization Relay) 참가기관(송신)에게 접수한 거래 건을 다중 센터에서 동일하게 처리하기 위한 접수 동기화 서비스

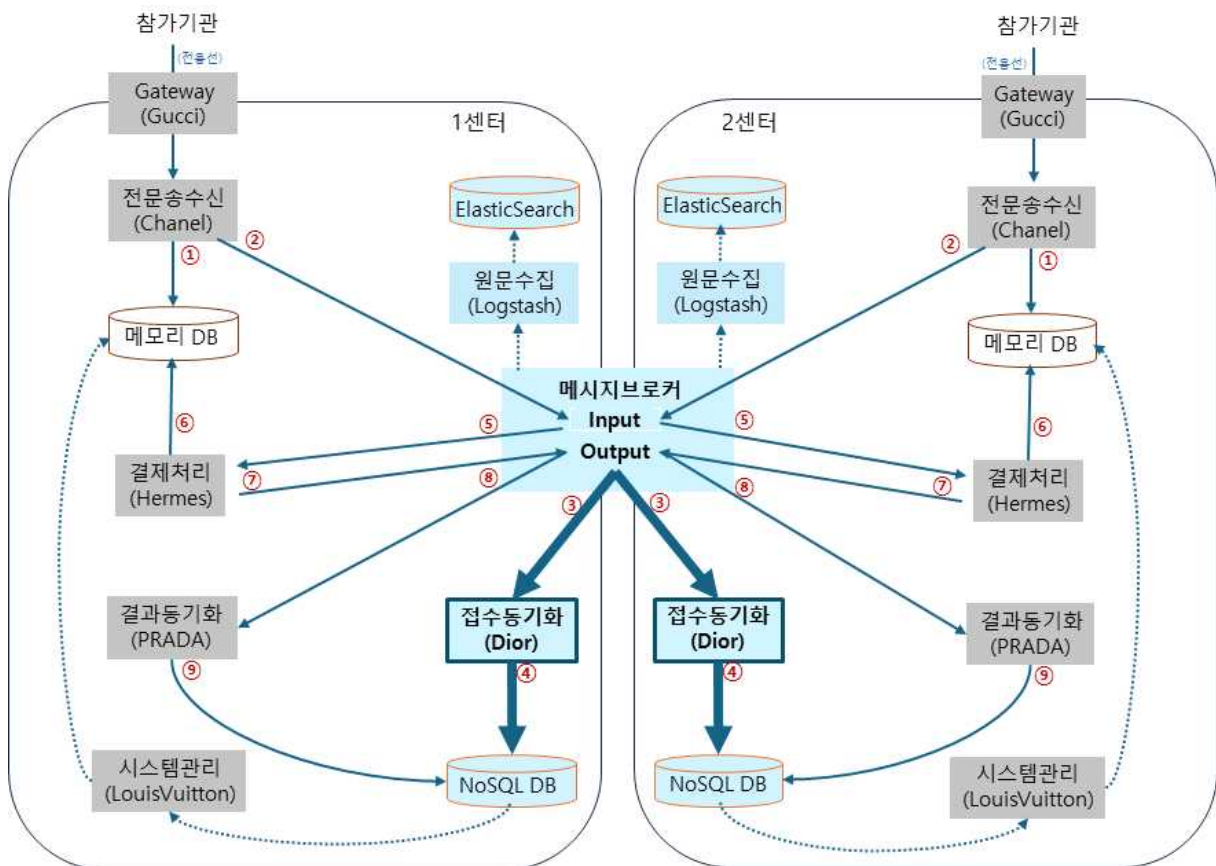
① 메시지 브로커(Kafka)의 결제 요청 토픽*을 구독

* kafka에서 데이터(메시지(이벤트) 스트림)을 저장하는 단위로 데이터베이스의 테이블이나 파일 시스템의 폴더 개념

② 메시지 내용을 메모리 DB에 저장(센터간 동기화)

③ 메시지 내용을 영구 기록 DB에 저장(센터간 동기화 및 재해 복구 대비)

접수 동기화 서비스



접수 동기화 프로그램

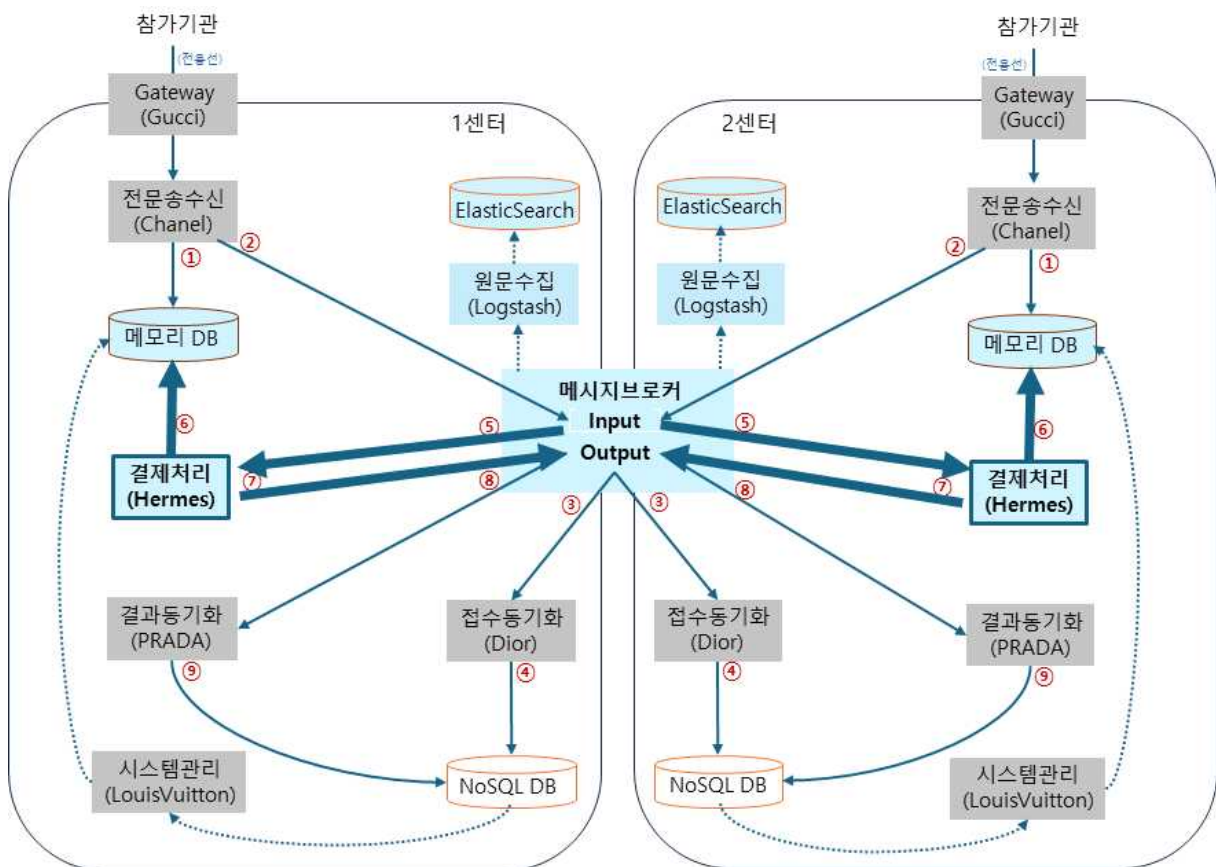
구분		소스코드	기능요약
dior		DiorApplication	접수동기화 응용시스템 초기화 실행
	application	DiorService	접수동기화 서비스
	domain	Transfer	자금이체 엔티티
infrastructure	config	MongoConfig	몽고DB 환경설정
		HazelcastConfig	인메모리 환경설정
		KafkaConsumerConfig	메시지브로커 컨슈머 환경설정
	properties	HazelcastProperties	인메모리 DB 환경설정
resources		application.yml	응용시스템 환경설정
		application-database.yml	응용시스템 DB 환경설정
		application-hazelcast.yml	인메모리 응용시스템 환경설정(환경에 따라 다른 Hazelcast 클러스터에 연결)
		application-kafka.yml	메시지브로커 응용시스템 환경설정 (Kafka 클라이언트의 프로듀서 및 컨슈머 설정)
		logback-spring.xml	로그관리 : Logback을 사용하여 로그를 Logstash로 전송하고, 개발 환경에 따라 다른 Logstash 서버로 전송
적용배포		.gitignore	컨테이너 환경에서 프로젝트 빌드 준비 및 불필요한 파일 정리
		build.gradle.kts	Spring Boot 기반 Kotlin 프로젝트 설정
		Dockerfile	Java Spring Boot 애플리케이션 빌드 실행 컨테이너 이미지 생성
		Dockerfile.gradle-cache	Gradle 빌드 프로세스 캐시 이미지 생성
		gradlew	프로젝트 빌드와 실행 스크립트
		gradlew.bat	Windows 환경에서 Gradle Wrapper 실행 배치 스크립트
		settings.gradle.kts	Gradle 프로젝트의 루트 설정

결제 처리 서비스

□ HERMES(High-Efficiency Rapid Memory Execution for Settlement) 각 센터에서 동일한 거래를 순서대로 처리하여 각 센터 내에 위치한 원장 데이터 베이스에 저장하는 결제처리 서비스

- ① 메시지 브로커의 결제 요청 토픽을 구독
- ② 메모리 DB에서 송신 기관과 수신 기관의 원장을 찾아 결제 결과 계산
- ③ 처리 이력 메모리 DB에 저장
- ④ 센터간 동기화 및 영구 기록을 위해 메시지 브로커로 결과를 전송

결제처리 서비스



결제처리 프로그램

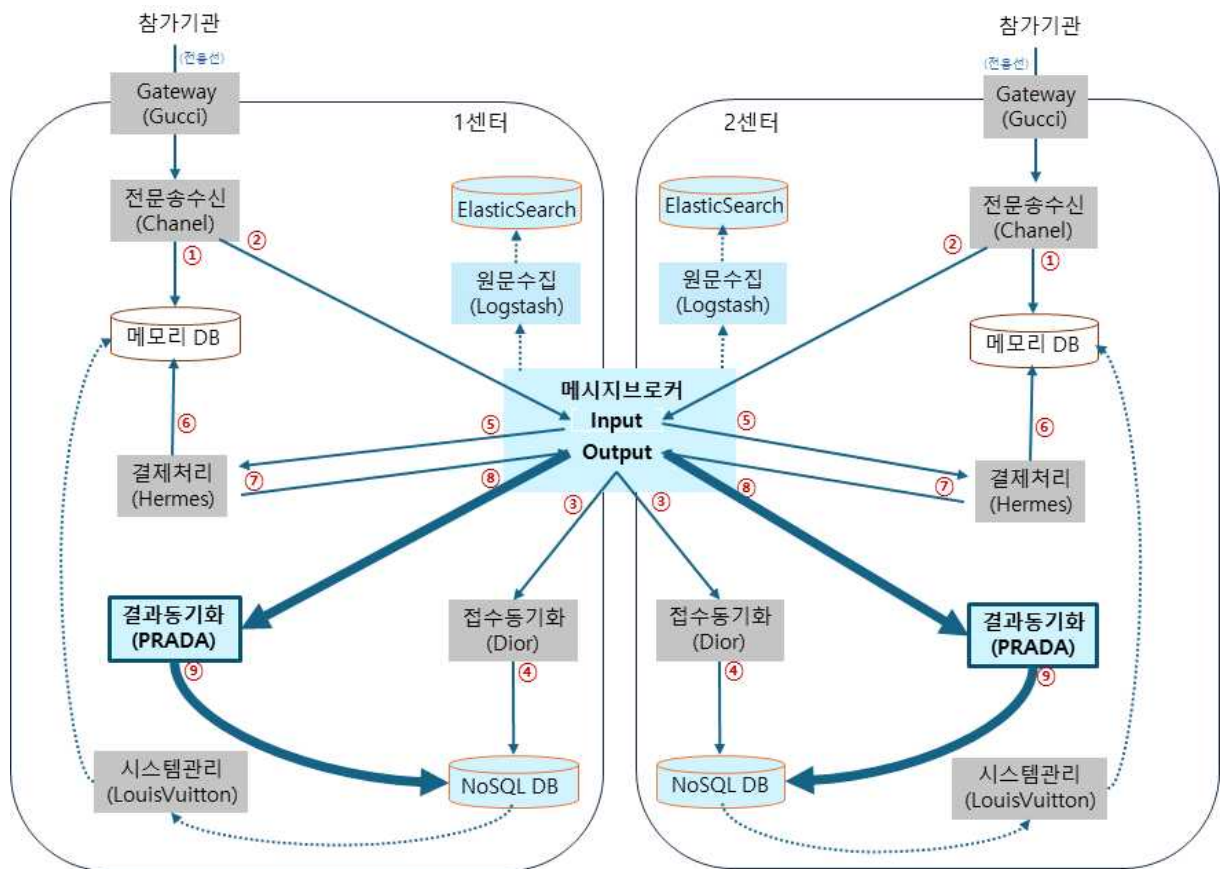
구분		소스코드	기능요약
hermes		HermesApplication	결제처리 응용시스템 초기화 실행
	application	HermesService	결제처리 서비스(Kafka 메시지를 수신하여 자금 이체를 처리하고, 결제 및 동기화를 수행)
	domain	Transfer	자금이체 객체를 관리하고, Hazelcast에 트랜잭션 데이터 저장 및 조회, 업데이트
		Lodger	Hazelcast를 기반으로 원장 엔티티 계좌(Account) 정보를 관리하고, 잔액을 실시간으로 업데이트
infrastructure	config	HermesConfig	결제처리 환경설정
		HazelcastConfig	인메모리 환경설정
		KafkaConsumerConfig	메시지브로커 컨슈머 환경설정
		KafkaProducerConfig	메시지브로커 프로듀서 환경설정
	properties	HazelcastProperties	인메모리 DB 환경설정
resources	application.yml		응용시스템 환경설정
	application-hazelcast.yml		인메모리 응용시스템 환경설정(환경에 따라 다른 Hazelcast 클러스터에 연결)
	application-kafka.yml		메시지브로커 응용시스템 환경설정(Kafka 클라이언트의 프로듀서 및 컨슈머 설정)
	logback-spring.xml		로그관리 : Logback을 사용하여 로그를 Logstash로 전송하고, 개발 환경에 따라 다른 Logstash 서버로 전송
적용배포	.gitignore		컨테이너 환경에서 프로젝트 빌드 준비 및 불필요한 파일 정리
	build.gradle.kts		Spring Boot 기반 Kotlin 프로젝트 설정
	Dockerfile		Java Spring Boot 애플리케이션 빌드 실행 컨테이너 이미지 생성
	Dockerfile.gradle-cache		Gradle 빌드 프로세스 캐시 이미지 생성
	gradlew		프로젝트 빌드와 실행 스크립트
	gradlew.bat		Windows 환경에서 Gradle Wrapper 실행 배치 스크립트
	settings.gradle.kts		Gradle 프로젝트의 루트 설정

결과 동기화 서비스

□ PRADA(Persistent Repository Administrator for Data Aggregation) 각 센터에서 처리한 거래처리 결과에 대한 결과 동기화 서비스

- ① 메시지 브로커의 결제 처리 토픽을 구독
- ② 결제 처리 메시지의 결과를 영구 DB에 저장(센터간 동기화 및 재해 복구 대비)
- ③ 요청에 대한 상태를 처리 완료로 메모리 DB에 최종 업데이트

결과 동기화 서비스



결과 동기화 프로그램

구분		소스코드	기능요약	
prada		PradaApplication	결과동기화 응용시스템 초기화 실행	
	application	PradaService	결과동기화 서비스	
	domain	Transfer	자금이체 엔티티	
		Lodger	원장 엔티티	
	infrastructure	config	PradaConfig	결과동기화 환경설정
			HazelcastConfig	인메모리 환경설정
			KafkaConsumerConfig	메시지브로커 컨슈머 환경설정
	properties	HazelcastProperties	인메모리 DB 환경설정	
	resources	application.yml	응용시스템 환경설정	
		application-database.yml	응용시스템 DB 환경설정	
		application-hazelcast.yml	인메모리 응용시스템 환경설정 (환경에 따라 다른 Hazelcast 클러스터에 연결)	
		application-kafka.yml	메시지브로커 응용시스템 환경설정 (Kafka 클라이언트의 프로듀서 및 컨슈머 설정)	
		logback-spring.xml	로그관리 : Logback을 사용하여 로그를 Logstash로 전송하고, 개발 환경에 따라 다른 Logstash 서버로 전송	
	적용배포	.gitignore	컨테이너 환경에서 프로젝트 빌드 준비 및 불필요한 파일 정리	
		build.gradle.kts	Spring Boot 기반 Kotlin 프로젝트 설정	
		Dockerfile	Java Spring Boot 애플리케이션 빌드 실행 컨테이너 이미지 생성	
		Dockerfile.gradle-cache	Gradle 빌드 프로세스 캐시 이미지 생성	
		gradlew	프로젝트 빌드와 실행 스크립트	
		gradlew.bat	Windows 환경에서 Gradle Wrapper 실행 배치 스크립트	
		settings.gradle.kts	Gradle 프로젝트의 루트 설정	

모니터링

- 전문의 송수신 및 결제처리 서비스의 처리결과는 Kafka와 Hazelcast 관리도구를 통해 저장된 데이터와 처리시간 세부사항 등을 확인 가능

Kafka 관리화면

CONFLUENT

Enterprise trial ends in 29 days

HOME > CONTROLCENTER.CLUSTER >

Cluster overview

Brokers

Topics

Connect

ksqlDB

Consumers

Replicators

Cluster settings

Health+

Consumer groups

Search consumer groups

Consumer group ID	Messages behind	Number of consumers	Number of topics
_confluent-controlcenter-7-8-0-1	6,425	12	8
_confluent-controlcenter-7-8-0-1-command	0	1	1
ConfluentTelemetryReporterSampler--23208395728667..	0	0	0
ConfluentTelemetryReporterSampler--81067258087777..	1,520,187	0	1
dior-group	994,229	1	1
dior-krs-group	1,251,509	1	1
hermes-group	3,090,878	1	1
hermes-krs-group	3,212,784	1	1
prada-group	2,091,311	1	1
prada-krs-group	2,307,657	1	1

Cluster overview

Brokers

Topics

Connect

ksqlDB

Consumers

Replicators

Cluster settings

Health+

dior-group

Consumer lagConsumption

948,949

Total Messages behind

Set up an alert

Current progress in processing

ISO20022.pacs.008

Max lag / consumer: 948,949 messages

1,000,000500,0000

Client ID	Consumer ID	Topic	Partition	Messages behind	Current offset	End offset
consumer-dior-group-1	consumer-dior-group-1-653a744..	ISO20022.pacs.008	0	948949	5022919	5971868

dior-krs-group

Consumer lagConsumption

1,195,144

Total Messages behind

Set up an alert

Current progress in processing

ISO20022.pacs.008

Max lag / consumer: 1,195,144 messages

10,000,0005,000,0000

Client ID	Consumer ID	Topic	Partition	Messages behind	Current offset	End offset
consumer-dior-krs-group-1	consumer-dior-krs-group-1-7772..	ISO20022.pacs.008	0	1195144	4776724	5971868

Hazelcast 관리화면

HAZELCAST

rtgs-hazelcast

5.5

+

Menu

Dashboard

Clients

Client Filtering - OFF

Members

Administration

WAN Replication

Scripting

Healthcheck

Namespaces

Storage

Stream Processing

Compute

Messaging

CP Subsystem

Query 1

×

+

Enter a single query below

Documentation

1 select * from accounts

Press <Ctrl+Space> to complete

EXECUTE QUERY

CLEAR QUERY RESULT

Query Results

History

EXPORT

	_key	name	acronyms	balance	createdAt	lastModifiedAt
✓	8070	한국증권금융	KSF	1000015711	2024-10-10T1...	2025-01-21T16:39:21.547734121
✓	1184	한국스탠다드차타드은행	SCB	1000000751	2024-10-10T1...	2025-01-21T16:39:21.611924993
✓	1051	국민은행	CNB	1000024352	2024-10-10T1...	2025-01-21T16:39:21.5903472

MongoDB 관리화면

The screenshot displays the MongoDB Compass application interface. On the left, the 'Database Explorer' shows a hierarchy for the 'rtgs' database, including collections like 'accounts' and 'transfers'. The main panel shows a table of data from the 'rtgs.transfers' collection, with columns for '_id', '_class', 'createdAt', 'creditorBalance', and 'creditorCode'. The table contains five rows of data. Below the table, the 'console_1 [krs-rtgs]' tab is active, showing a JavaScript playground with the following code:

```
1 use rtgs;
2 db.createCollection("accounts");
3
4 db.accounts.insertMany([
5   {
6     _id: 1020,
7     name: '한국산업은행',
8     acronyms: 'KDB',
9     balance: 1000000000,
10  }
11 ])
```

The bottom status bar shows the current database as 'rtgs' and the current collection as 'transfers'. The bottom right corner displays system information: 4:25, LF, UTF-8, 4 spaces, and a refresh icon.

Kibana 관리화면

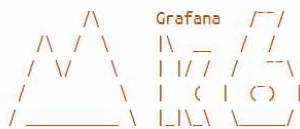
□ 로컬 개발 환경(물리 노드 1대)에서 테스트 수행

- k6*를 사용하여 총 50,000건(가상유저 40개)의 이체요청을 실행한 결과, 건당 처리는 평균 7.95ms** 소요되었으며 초당 5,016건을 처리(가상서버 기반의 2024.8월 테스트에서의 17.64ms, 2,204건 처리에 비해 성능 향상)

* K6 : 성능테스트 도구로 2017년 Load Impact 사에서 출시(2021년 Grafana Labs에 인수)되었으며, 특히 Kubernetes와 같은 클라우드 환경과 DevOps 파이프라인에 적합하며, 숫자 6은 여섯가지 핵심 요소(성능, 확장성, 자동화, 유연성, 통합성, 안정성)를 의미

** ms(millisecond) : 1000분의 1초

성능 테스트 도구(K6) 실행 결과



```

execution: local
script: test.js
output: InfluxDBv1 (http://10.16.16.101:8086)

scenarios: (100.00%) 1 scenario, 40 max VUs, 10m30s max duration (incl. graceful stop):
    * default: 50000 iterations shared among 40 VUs (maxDuration: 10m0s, gracefulStop: 30s)

✓ is status 200

checks.....: 100.00% 50000 out of 50000
data_received.....: 83 MB  8.3 MB/s
data_sent.....: 225 MB  23 MB/s
http_req_blocked.....: avg=28.03µs min=622ns med=2.37µs max=7.07ms p(90)=4.04µs p(95)=5.89µs
http_req_connecting.....: avg=22.91µs min=0s med=0s max=7.03ms p(90)=0s p(95)=0s
http_req_duration.....: avg=7.66ms min=1.37ms med=6.93ms max=33.17ms p(90)=10.98ms p(95)=12.74ms
    { expected_response:true }...: avg=7.66ms min=1.37ms med=6.93ms max=33.17ms p(90)=10.98ms p(95)=12.74ms
http_req_failed.....: 0.00% 0 out of 50000
http_req_receiving.....: avg=511.71µs min=10.03µs med=240.47µs max=15.65ms p(90)=1.3ms p(95)=1.79ms
http_req_sending.....: avg=69.01µs min=-2929631ns med=35.66µs max=11.78ms p(90)=81.46µs p(95)=147.3µs
http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
http_req_waiting.....: avg=7.08ms min=0s med=6.35ms max=32.56ms p(90)=10.24ms p(95)=11.96ms
http_reqs.....: 50000 5016.086859/s
iteration_duration.....: avg=7.95ms min=3.25ms med=7.2ms max=33.51ms p(90)=11.38ms p(95)=13.22ms
iterations.....: 50000 5016.086859/s
vus.....: 40 min=40 max=40
vus_max.....: 40 min=40 max=40

running (00m10.0s), 00/40 VUs, 50000 complete and 0 interrupted iterations
default ✓ [=====] 40 VUs 00m10.0s/10m0s 50000/50000 shared iters
  
```

□ 클라우드 환경(외부 가상 테스트 노드 4대)에서 테스트 수행

- ① 네이버 공공클라우드의 2개(수도권, 남부권) 센터 중에 1개를 교대로 지정
- ② 등록된 19개 참가기관 중에 2개(송신, 수신) 기관을 무작위로 선정
- ③ ISO20022 전문(pacs.008)을 실시간으로 생성(이체 금액은 1~200원 사이)
- ④ 테스트 시나리오 실행

④-1 전문송수신(CHANEL) 서비스

- ISO20022 전문(pacs.008)을 접수하여 저널로 파싱
- 인메모리 DG(Hazelcast)의 해당 이체내역 상태값을 'RCVD'으로 저장
- 저널은 메시지 브로커(Kafka)에 INPUT 토픽으로 등록
- 인메모리 DG(Hazelcast)의 해당 이체내역 상태값을 'ACTC'로 수정
- ISO20022 전문(pacs.002) 응답

④-2 접수동기화(DIOR) 서비스

- 영구저장DB(MongoDB)에 저장하여 접수 동기화

④-3 결제처리(HERMES) 서비스

- 계좌간 이체처리 실시
- 인메모리 DG(Hazelcast)의 원장을 수정
- 처리결과를 메시지 브로커(Kafka)에 OUTPUT 토픽으로 등록
- 인메모리 DG(Hazelcast)의 해당 이체내역 상태값을 'ACSP'로 수정

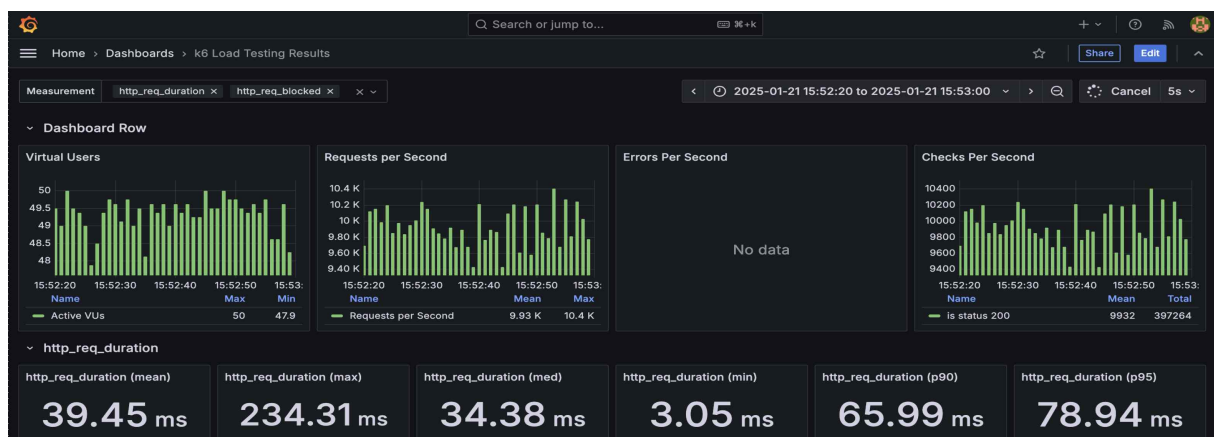
④-4 결과동기화(PRADA) 서비스

- 영구저장DB(MongoDB)에 저장하여 결과 동기화
- 인메모리 DG(Hazelcast)의 해당 이체내역 상태값을 'ACCC'로 수정

⑤ 각 노드별 수행한 테스트 데이터는 InfluxDB에 저장

⑥ 테스트 결과는 Grafana 대시보드로 확인

- 동시 가상서버 4대에서 k6를 사용하여 1분간 이체요청을 실행한 결과, **전당 처리시간은 평균 39.45ms 소요되었으며 평균 초당 9,930건을 처리**



□ 센터 간 건별 처리에 대한 정합성 확인

- 클라우드 환경의 테스트 절차 중에 ④번 테스트 시나리오를 수행한 결과 각 센터*에서 접수한 건이 다른 센터에도 동일하게 처리됨을 확인(300만건 전수확인)

* 네이버 공공클라우드의 2개 센터인 수도권(평촌)과 남부권(부산) 센터는 직선거리 303km 떨어져 있으며, 각 센터 간에는 6.5ms의 Latency(통신 지연) 존재

수동 테스트 A : 수도권에서 접수 처리된 건 (31ms)이 남부권에서도 처리(89ms)

(수도권) 거래요청 [11:22:46 101ms]

POST http://ing-services-chanelingre-3fd04-100803444-e0f627e824db.kr-gov.lb.navemcp.com/transfer

Body (XML):

```
<Cd>KRBOK</Cd>
<ClrSysId>
<MbId>1020</MbId>
<ClrSysMbId>
</ClrSysMbId>
</FinInstId>
</FIId>
</To>
<BizMsgId>20250119100400000030</BizMsgId>
<MsgDefId>pac.008.001.08</MsgDefId>
<BizSvc>bk.rtgs.proto.01</BizSvc>
```

Response: 200 OK, 94 ms, 1.61 KB

(수도권) 거래조회 [11:22:46 132ms]

GET http://ing-services-chanelingre-3fd04-100803444-e0f627e824db.kr-gov.lb.navemcp.com/inquiry/2...

Body (JSON):

```
{
  "transferId": "20250119100400000030",
  "txStatus": "ACCC",
  "txAmount": 100,
  "debtorCode": 1051,
  "debtorBalance": 99999800,
  "creditorCode": 1020,
  "creditorBalance": 1000001100,
  "requestAt": "2025-01-21T11:22:46.097",
  "lastStatusUpdatedAt": "2025-01-21T11:22:46.132547571"
}
```

Response: 200 OK, 49 ms, 371 B

(남부권) 수도권 거래조회 [11:22:46 190ms]

GET http://ing-services-chanelingre-39c1a-100805918-aad7a250772c.krs-gov.lb.navemcp.com/inquiry/2...

Body (JSON):

```
{
  "transferId": "20250119100400000030",
  "txStatus": "ACCC",
  "txAmount": 100,
  "debtorCode": 1051,
  "debtorBalance": 99999800,
  "creditorCode": 1020,
  "creditorBalance": 1000001100,
  "requestAt": "2025-01-21T11:22:46.097",
  "lastStatusUpdatedAt": "2025-01-21T11:22:46.190912203"
}
```

Response: 200 OK, 94 ms, 371 B

수동 테스트 B : 남부권에서 접수 처리된 건 (67ms)이 수도권에서도 빠르게 처리(63ms)

(남부권) 거래요청 [11:28:12 135ms]

POST http://ing-services-chanelingre-39c1a-100805918-aad7a250772c.krs-gov.lb.navemcp.com/transfer

Body (XML):

```
<ClrSysId>
<Cd>KRBOK</Cd>
<ClrSysId>
<MbId>1020</MbId>
<ClrSysMbId>
</ClrSysMbId>
</FinInstId>
</FIId>
</To>
<BizMsgId>20250119100400000031</BizMsgId>
<MsgDefId>pac.008.001.08</MsgDefId>
<BizSvc>bk.rtgs.proto.01</BizSvc>
```

Response: 200 OK, 73 ms, 1.61 KB

(남부권) 거래조회 [11:28:12 202ms]

GET http://ing-services-chanelingre-39c1a-100805918-aad7a250772c.krs-gov.lb.navemcp.com/inquiry/2...

Body (JSON):

```
{
  "transferId": "20250119100400000031",
  "txStatus": "ACCC",
  "txAmount": 100,
  "debtorCode": 1051,
  "debtorBalance": 99999800,
  "creditorCode": 1020,
  "creditorBalance": 1000001200,
  "requestAt": "2025-01-21T11:28:12.122",
  "lastStatusUpdatedAt": "2025-01-21T11:28:12.20272247"
}
```

Response: 200 OK, 27 ms, 370 B

(수도권) 남부권 거래조회 [11:28:12 198ms]

GET http://ing-services-chanelingre-3fd04-100803444-e0f627e824db.kr-gov.lb.navemcp.com/inquiry/2...

Body (JSON):

```
{
  "transferId": "20250119100400000031",
  "txStatus": "ACCC",
  "txAmount": 100,
  "debtorCode": 1051,
  "debtorBalance": 99999800,
  "creditorCode": 1020,
  "creditorBalance": 1000001200,
  "requestAt": "2025-01-21T11:28:12.122",
  "lastStatusUpdatedAt": "2025-01-21T11:28:12.19882509"
}
```

Response: 200 OK, 128 ms, 370 B

□ 1차 테스트 결과의 의의

- 2개의 센터에서 데이터 복제 방식이 아닌 클러스터링 동기화를 기반으로 동일한 처리를 수행하는 분산시스템을 설계하여 핵심기능을 점검함
 - 1개 센터에서 접수 처리된 데이터는 2개 센터에서 모두 동일하게 처리
 - 2개 센터를 모두 Active-Active로 운영
 - 전당 처리시간은 평균 39.45ms, 초당 9,930건을 처리
- 오픈소스 기반에서 수행한 테스트 환경의 한계로, 2개 센터를 다중 클러스터로 적용하지 못하고 메시지 브로커 단일 클러스터를 두 센터에서 공동으로 사용하였으며, 복원력 테스트는 미 실시

Ⅲ

향후 추진 계획

□ 시스템 성능 개선 검토(2025.3Q)

- 초당 처리건수 최대 20,000건*을 목표로 성능 부하 테스트 단계적 확장
 - * 2030년 이후 하루 1.1억건을 처리를 예상하면 이론적으로는 초당 1,851건 처리가 필요하나, 2024년 기준 하루 2.3천만건 처리중인 금융결제원의 피크시간대 초당 대략 4,000건 처리를 참고하여 소액RTGS 프로토타입 시스템의 피크시간대 목표는 최대 초당 20,000건을 설정
- 테스트 결과를 바탕으로 서버 노드 및 네트워크 구성 변경 검토
- 메시지 브로커에 대한 다중 클러스터(MRC)를 구성하여 복원력 점검
- NoSQL 및 메시지 브로커의 기능을 수행하는 솔루션 구성 변경 검토

□ 전체 시스템 구성 점검(2025.4Q)

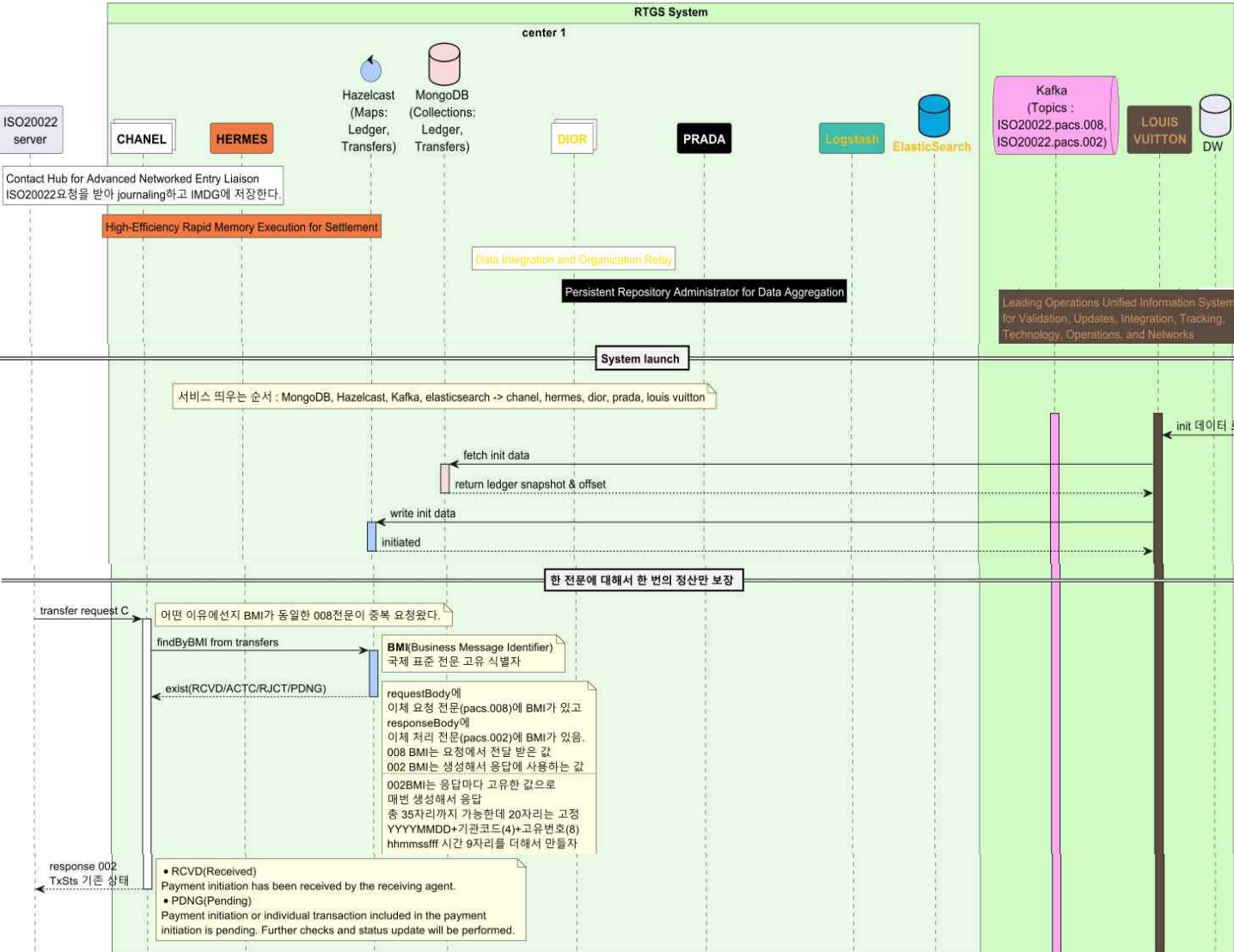
- 게이트웨이(GUCCI), 시스템관리(LUIS VUITTON) 등 서비스 2개 추가 개발
- 참가기관 사용자의 인증키 관리모듈 구성
- 개인정보 DB저장에 대한 암호화 솔루션 적용
- 클라우드 네이티브 환경에서의 변경/형상/배포관리를 위한 쿠버네티스 CI/CD 파이프라인 구성

<별첨1>

UML (Unified Modeling Language*)

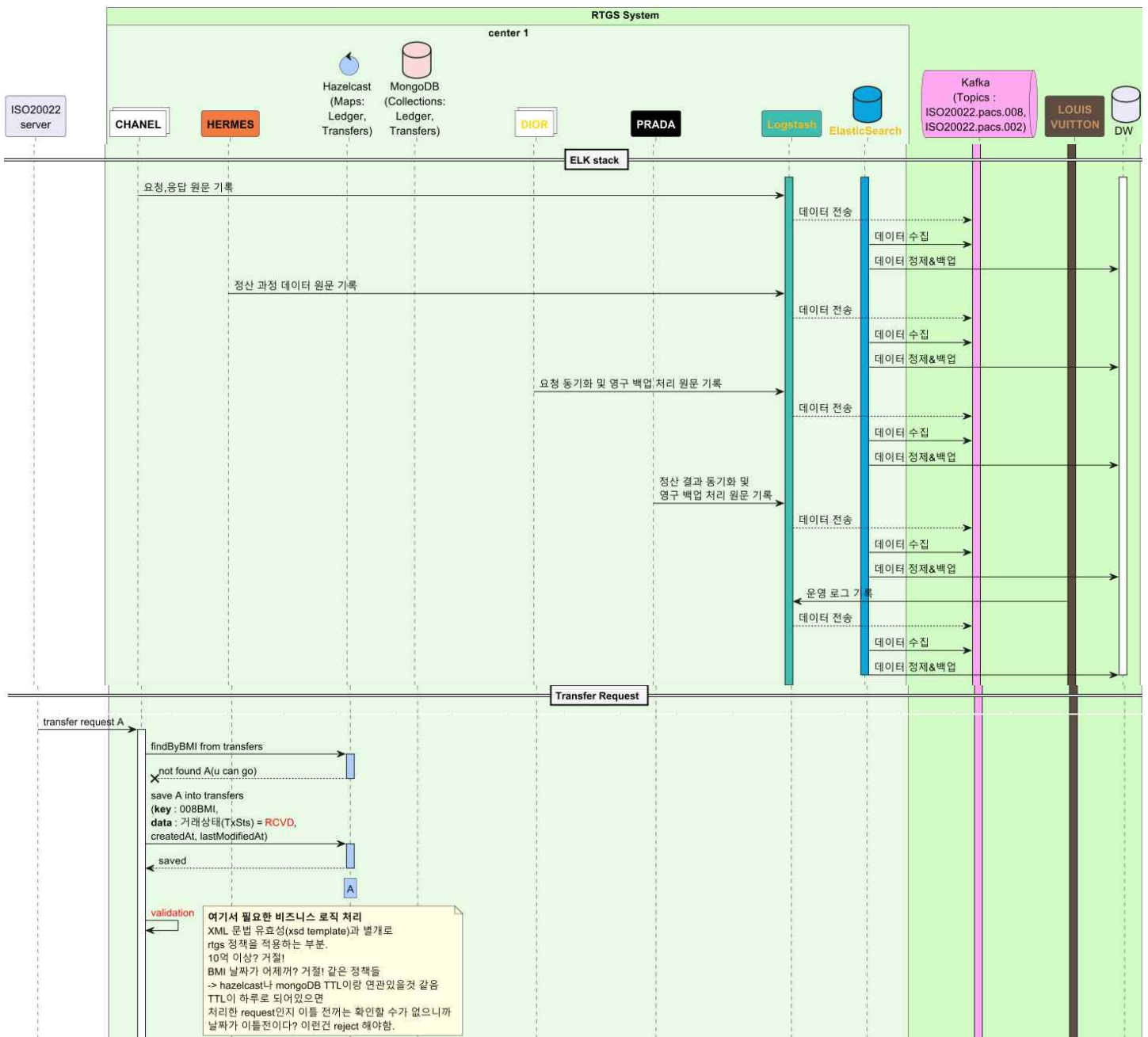
* 소프트웨어 설계를 위한 통합 모델링 언어

- 소액 RTGS시스템의 자금이체 업무는 ① 게이트웨이(GUCCI), ② 전문송수신(CHANEL), ③ 결제처리(HERMES), ④ 접수동기화(DIOR), ⑤ 결과동기화(PRADA), ⑥ 시스템관리(LUIS VUITTON) 등 6개의 서비스로 구성
- 시스템관리(LUIS VUITTON) 서비스에서 전체 시스템에 대한 재기동 및 BCP(DB 복구 포함) 관리를 수행
- 참가기관의 자금이체 요청 건은 ISO20022 전문(pacs.008)으로 접수하여 식별자 BMI (Business Message Identifier)의 중복여부를 체크하고 'RCVD'(타 센터에서 접수된 경우는 'PDNG')로 응답 전문(pacs.002) 생성



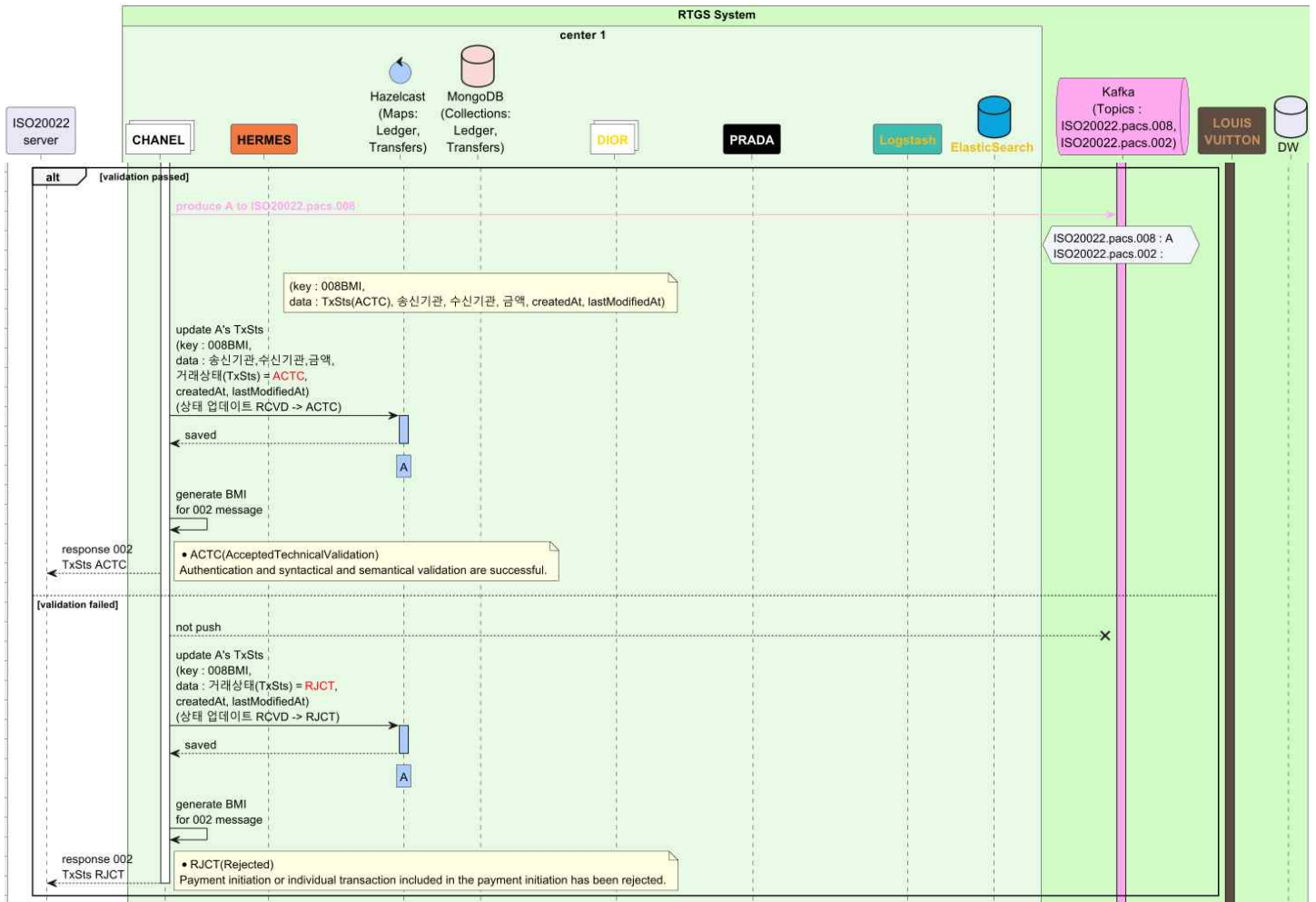
□ 전문송수신(CHANEL), 결제처리(HERMES), 접수동기화(DIOR), 결과동기화(PRADA) 등 4개 서비스에서 수행하는 모든 작업 내역은 로그(Logstash)로 수집하여 데이터(ElasticSearch)로 저장

○ 전문송수신(CHANEL) 서비스에서 접수한 정상 건(상태값 'RCVD' 확인)은 추가적으로 업무 규칙을 확인하고 인메모리 데이터그리드(Hazelcast)에 신속하게 저장

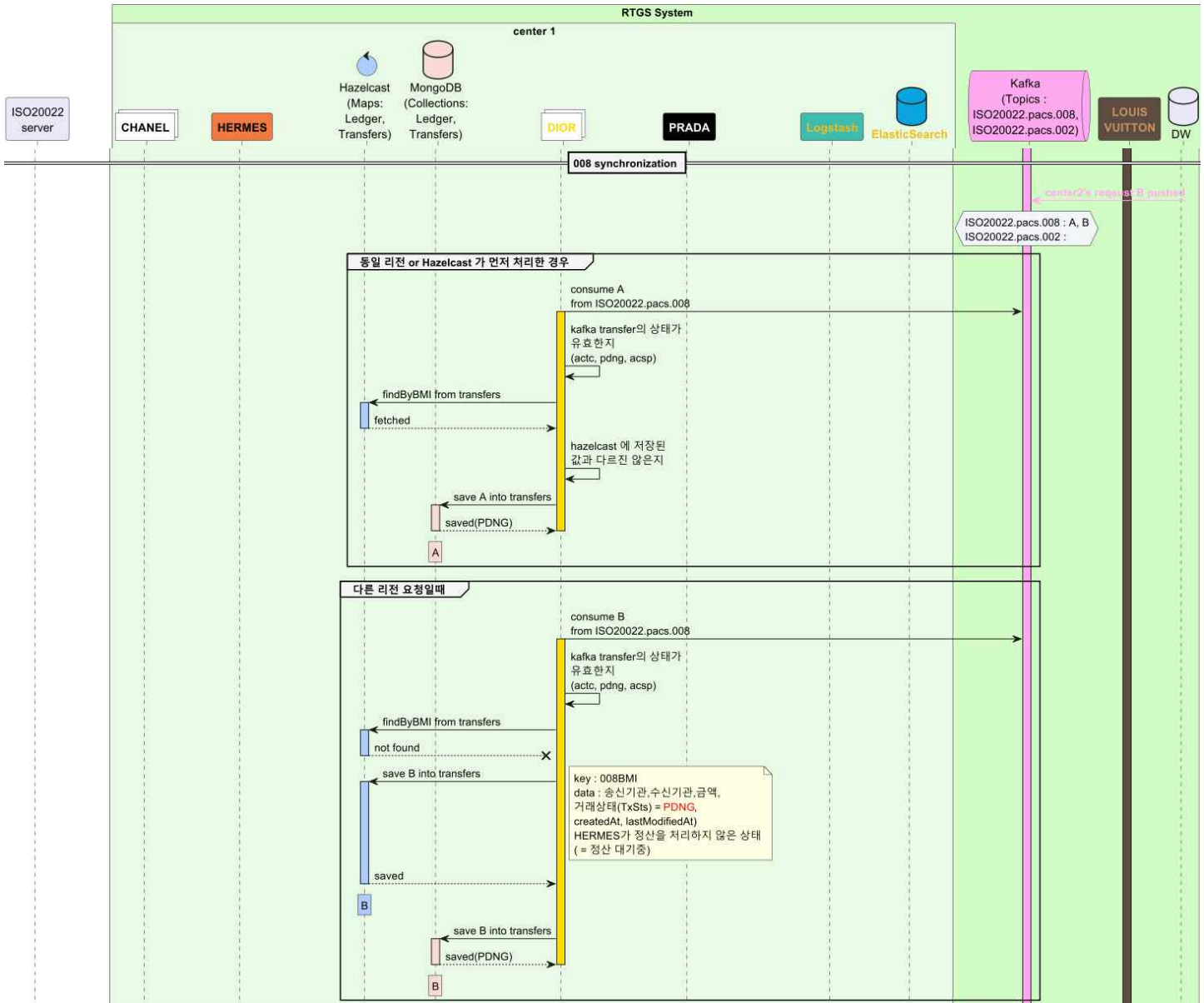


□ 각 센터에서 참가기관으로부터 송신된 전문은 센터 간에 클러스터링으로 구성된 메시지 브로커(Kafka)에 전달하여 각 센터의 작업을 동기화

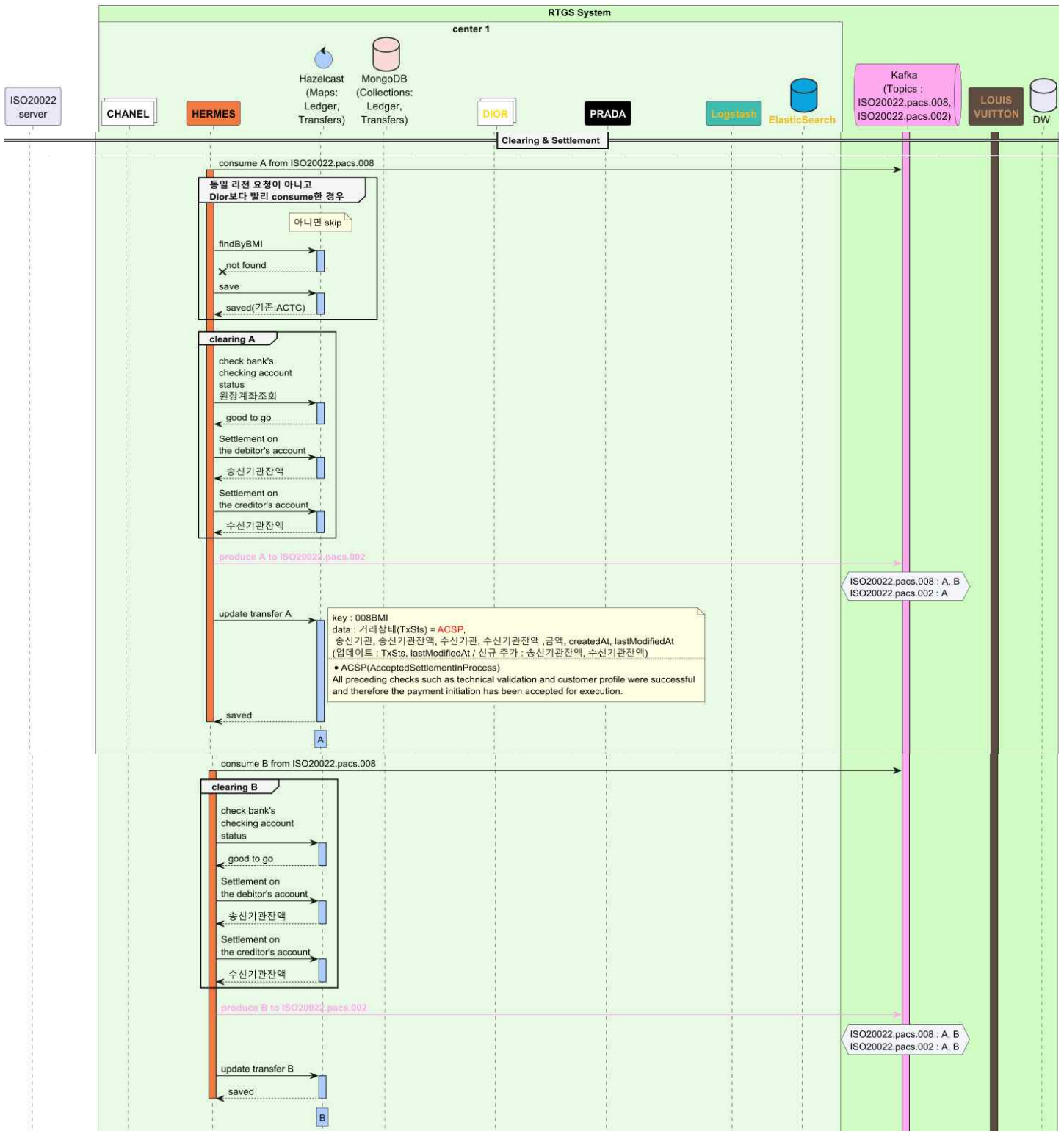
- 전문송수신(CHANEL) 서비스에서 인메모리 데이터그리드(Hazelcast)에 저장된 전문의 상태는 'ACTC'로 수정되고, 비정상 처리되는 전문의 상태는 'RJCT'로 수정하여 송신기관 앞 응답전문(pacs.002) 생성



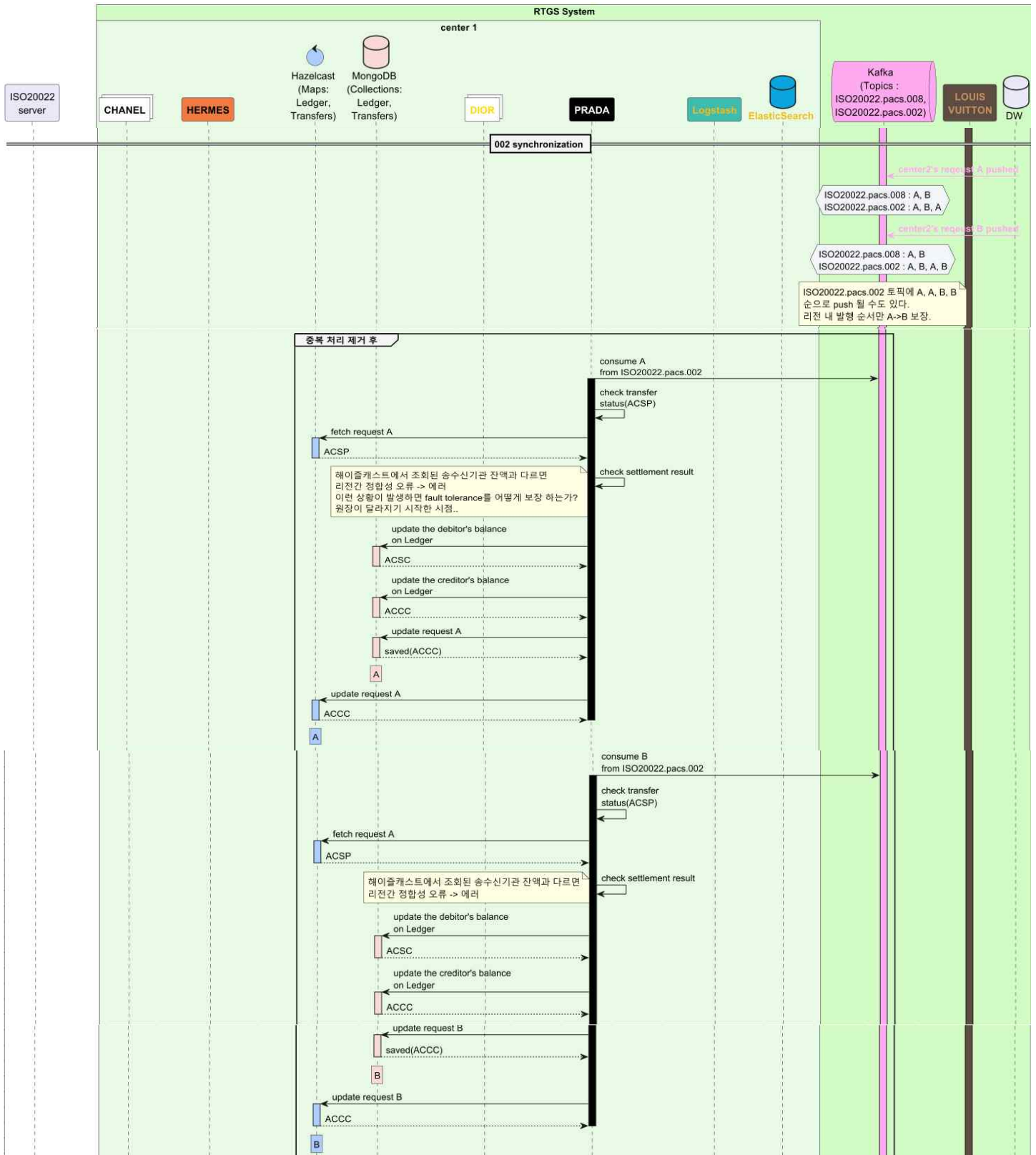
□ 메시지 브로커(Kafka)에 입력된 접수내용은 접수동기화(DIOR) 서비스를 통해 유효성을 체크한 후 원장DB(MongoDB)에 저장



- 메시지 브로커(Kafka)에 입력된 접수내용은 결제처리(HERMES) 서비스에서 결제처리후 인메모리 데이터그리드(Hazelcast)에 저장하고 상태값을 'ACSP'로 저장



- 결과동기화(PRADA) 서비스에서 상태값 'ACSP'을 확인하여 센터 간 처리결과를 동기화 하여 인메모리 데이터그리드(Hazelcast)에 저장하고 상태값을 'ACCC'로 저장



<별첨2>

프로토타입 서버 구성

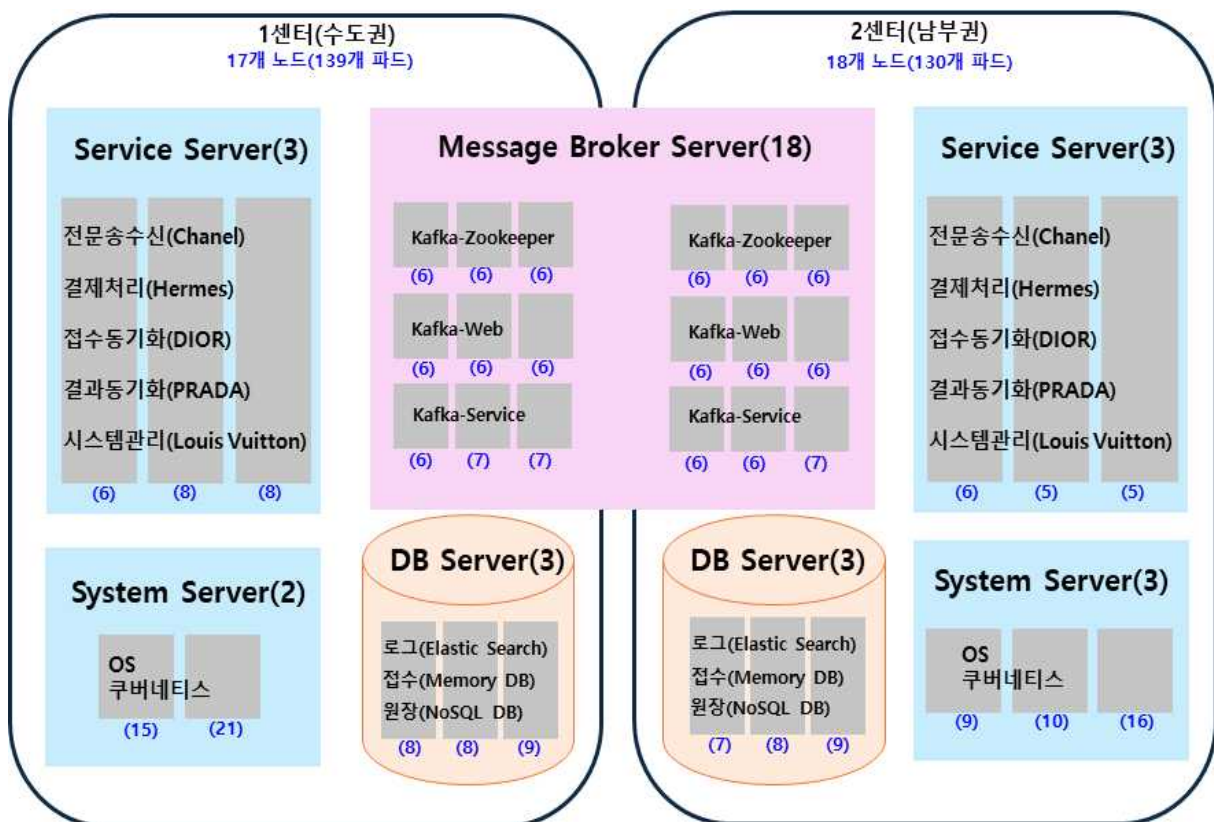
□ 소액RTGS시스템 구축을 위한 프로토타입 시스템은 네이버 공공클라우드에서 제공하는 2개의 센터(수도권, 남부권)에 분산하여 구성

○ 수도권 센터는 17개 서버 노드(Node*)에 139개 파드(Pod**), 남부권 센터는 18개 서버 노드(Node)에 130개 파드(Pod)로 구성

* 노드(Node) : 여러 서버가 함께 협력하여 하나의 큰 시스템을 이루는 경우에 각 서버를 의미하며, 노드는 시스템의 일부로서 협력하여 작업을 처리하거나 서비스를 제공. 서버 노드는 클러스터 또는 분산 시스템에서 여러 서버가 서로 연결되어 하나의 통합된 시스템처럼 동작

** 파드(Pod) : 쿠버네티스에서 가장 작은 배포 단위로 다수의 컨테이너를 묶어서 실행하는 논리적 호스트를 의미하며, 클러스터 내에서 고유한 IP 주소를 할당

서버 노드(파드) 구성도



센터별 서버 노드(파드) 목록

리전	구분	노드		IP주소	vCPU	Memory	파드수
수도권	Database	1	database-w-2nqx	10.0.1.9	4	32GB	8
		2	database-w-2nqy	10.0.1.11	4	32GB	8
		3	database-w-2nvg	10.0.1.12	4	32GB	9
	Kafka	4	kafka-service-w-2lbi	10.0.1.27	4	32GB	7
		5	kafka-service-w-2lbj	10.0.1.28	4	32GB	6
		6	kafka-service-w-2lbk	10.0.1.29	4	32GB	7
		7	kafka-w-2lb9	10.0.1.21	4	32GB	6
		8	kafka-w-2lba	10.0.1.22	4	32GB	6
		9	kafka-w-2lbb	10.0.1.23	4	32GB	6
		10	zookeeper-w-2lbc	10.0.1.24	2	8GB	6
		11	zookeeper-w-2lbd	10.0.1.25	2	8GB	6
		12	zookeeper-w-2lbe	10.0.1.26	2	8GB	6
		services	13	services-w-2i4u	10.0.1.10	4	16GB
	14		services-w-2nqu	10.0.1.6	4	16GB	8
	15		services-w-2nqv	10.0.1.7	4	16GB	6
	system	16	system-w-2i7t	10.0.1.15	2	8GB	21
		17	system-w-2i7z	10.0.1.16	2	8GB	15
남부권	Database	1	database-w-2nx8	10.1.1.22	4	32GB	9
		2	database-w-2nx9	10.1.1.23	4	32GB	8
		3	database-w-2nxa	10.1.1.24	4	32GB	7
	Kafka	4	kafka-service-w-2oqb	10.1.1.15	4	32GB	7
		5	kafka-service-w-2oqc	10.1.1.16	4	32GB	6
		6	kafka-service-w-2oqd	10.1.1.17	4	32GB	6
		7	kafka-w-2oq8	10.1.1.12	4	32GB	6
		8	kafka-w-2oq9	10.1.1.13	4	32GB	6
		9	kafka-w-2oqa	10.1.1.14	4	32GB	6
		10	zookeeper-w-2oq5	10.1.1.9	2	8GB	6
		11	zookeeper-w-2oq6	10.1.1.10	2	8GB	6
		12	zookeeper-w-2oq7	10.1.1.11	2	8GB	6
		services	13	services-w-2nx5	10.1.1.19	2	8GB
	14		services-w-2nx6	10.1.1.20	2	8GB	5
	15		services-w-2nx7	10.1.1.21	2	8GB	5
	system	16	system-w-2kk1	10.1.1.6	2	8GB	10
		17	system-w-2kk2	10.1.1.7	2	8GB	9
		18	system-w-2kk3	10.1.1.8	2	8GB	16
(전체 서버 구성)		(노드수)		35	(파드수)		269

□ 수도권(bok-now-kr)

구분	Server		POD
chanel	services-w-2i4u	1	chanel-89bc44d78-rwgdb
	services-w-2nqu	2	chanel-89bc44d78-ddf4k
confluent	kafka-service-w-2lbi	3	controlcenter-0
		4	schemaregistry-2
	kafka-service-w-2lbi	5	schemaregistry-0
	kafka-service-w-2lbi	6	schemaregistry-1
	kafka-w-2lb9	7	confluent-operator-7556bd4b88-4krfk
	kafka-w-2lba	8	confluent-kafka-dev-0
	kafka-w-2lbb	9	confluent-kafka-dev-1
	zookeeper-w-2lbc	10	confluent-kafka-dev-2
	zookeeper-w-2lbd	11	zookeeper-1
default	zookeeper-w-2lbe	12	zookeeper-2
		13	zookeeper-0
	services-w-2i4u	14	dior-8b6d89b6f-6ssl7
		15	prada-5dbfddd889-x66q5
	services-w-2nqu	16	dior-8b6d89b6f-c2czv
		17	hermes-6f56846986-sk8kc
	services-w-2nqv	18	prada-5dbfddd889-497qx
	system-w-2i7t	19	k6-sample-1-mhrcm
		20	k6-sample-4-v4nvw
elastic	system-w-2i7z	21	k6-sample-2-r9n7x
		22	k6-sample-3-9vrr
		23	k6-sample-5-tdx9z
hazelcast	database-w-2nqx	24	rtgs-elasticsearch-759f95cdc4-ml27t
	database-w-2nqy	25	rtgs-kibana-9fbfc7d88-zfbq4
	database-w-2nvg	26	rtgs-logstash-6f8fbd4d56-6nntj
ingress-nginx	database-w-2nqx	27	rtgs-hazelcast-mc-5d84cf85f7-5q6lc
	database-w-2nvg	28	rtgs-hazelcast-6b54fcd9f9-jrzdq
k6-operator-system	system-w-2i7z	29	ingress-nginx-admission-create-x6rk5
		30	ingress-nginx-admission-patch-v2sdl
		31	ingress-nginx-controller-7d7c989655-skmwg
kafka	system-w-2i7z	32	k6-operator-controller-manager-7c9f6d96fc-zp4hn
mongodb	database-w-2nqy	33	rtgs-kafka-878569b96-v896z
	database-w-2nqx	34	example-mongodb-1
	database-w-2nqy	35	example-mongodb-0
	database-w-2nvg	36	example-mongodb-2
kube-system		37	mongodb-kubernetes-operator-75fb54bf8-xjrl
		38	cilium-kcxs8
	database-w-2nqx	39	csi-nks-node-hzphq
		40	nks-nas-csi-node-8gw6j
		41	nks-nodelocalproxy-6nzhr
		42	nodelocaldns-p8ng2
	database-w-2nqy	43	cilium-qc5wz
		44	csi-nks-node-bjll7
		45	nks-nas-csi-node-lmsfd
		46	nks-nodelocalproxy-2qwgw
		47	nodelocaldns-xrkd4
	database-w-2nvg	48	cilium-m2cfl
		49	csi-nks-node-wrslz
		50	nks-nas-csi-node-swwcm
		51	nks-nodelocalproxy-ncwhf
		52	nodelocaldns-vm7cc
	kafka-service-w-2lbi	53	cilium-w86lh
		54	csi-nks-node-xf4ln
		55	nks-nas-csi-node-24r7x
		56	nks-nodelocalproxy-q6lwj
		57	nodelocaldns-q4cj4
	kafka-service-w-2lbj	58	cilium-vcl2r
		59	csi-nks-node-5wldf
		60	nks-nas-csi-node-ljlsb
		61	nks-nodelocalproxy-m4tgp
		62	nodelocaldns-dkzn9
	kafka-service-w-2lbi	63	cilium-7sqct
		64	csi-nks-node-bq7nx
		65	nks-nas-csi-node-lktpc
		66	nks-nodelocalproxy-nbr9
		67	nodelocaldns-92wjz
	kafka-w-2lb9	68	cilium-v98xw

구분	Server	POD	
		69	csi-nks-node-mtjmk
		70	nks-nas-csi-node-bnps7
		71	nks-nodelocalproxy-5k759
		72	nodelocaldns-bvndm
	kafka-w-2lba	73	cilium-gthv2
		74	csi-nks-node-kl9gl
		75	nks-nas-csi-node-rpmcv
		76	nks-nodelocalproxy-ss4gz
		77	nodelocaldns-gg2q5
	kafka-w-2lbb	78	cilium-lsjsg
		79	csi-nks-node-pqdh8
		80	nks-nas-csi-node-tlftk
		81	nks-nodelocalproxy-c54qr
	services-w-2i4u	82	nodelocaldns-x45h5
		83	cilium-fwm6b
		84	csi-nks-node-9xsrsv
		85	nks-nas-csi-node-2qc9x
	services-w-2nqu	86	nks-nodelocalproxy-vmj5
		87	nodelocaldns-hms6f
		88	cilium-gxv8m
		89	csi-nks-node-hgnkq
		90	nks-nas-csi-node-9q6tt
	services-w-2nqv	91	nks-nodelocalproxy-5wjx8
		92	nodelocaldns-5w6zh
		93	cilium-85k6t
		94	csi-nks-node-25ns4
	system-w-2i7t	95	nks-nas-csi-node-r65vh
		96	nks-nodelocalproxy-dd8nd
		97	nodelocaldns-bv469
		98	cilium-monitor-mf6px
	system-w-2i7z	99	cilium-operator-6f856c7688-prnfm
		100	cilium-plgtz
		101	coredns-69c4df9f67-mls9d
		102	coredns-69c4df9f67-s5fxd
		103	csi-nks-node-zp5x5
		104	dns-autoscaler-ff7d667b4-7gnbl
		105	konnectivity-agent-9d967fc6c-5ngdw
		106	konnectivity-agent-9d967fc6c-f2gxt
		107	konnectivity-agent-9d967fc6c-xgjdd
		108	konnectivity-agent-autoscaler-974bf5567-z6rcf
		109	metrics-server-6b88b57445-knpmb
		110	nks-metric-sender-566bc54c69-2wpvb
		111	nks-metric-sender-566bc54c69-rdsnh
		112	nks-nas-csi-controller-78fd9f7b6f-gfm6l
		113	nks-nas-csi-node-llsp6
	zookeeper-w-2lbc	114	nks-nodelocalproxy-xd6nj
		115	nodelocaldns-hkc5s
		116	snapshot-controller-0
		117	cilium-jzjzm
		118	cilium-monitor-fzrz8
		119	cilium-operator-6f856c7688-2fqmm
		120	csi-nks-controller-7476f4c7cf-kpq2f
	zookeeper-w-2lbd	121	csi-nks-node-77fkm
		122	nks-nas-csi-node-v97qp
		123	nks-nodelocalproxy-nlq8n
		124	nodelocaldns-f9pld
	zookeeper-w-2lbe	125	cilium-sb6fm
		126	csi-nks-node-64kpr
		127	nks-nas-csi-node-jdcxs
		128	nks-nodelocalproxy-p87ft
		129	nodelocaldns-krtbw
		130	cilium-d652t
		131	csi-nks-node-q2qsm
		132	nks-nas-csi-node-v7hvp
		133	nks-nodelocalproxy-8tpmn
		134	nodelocaldns-k96pr
		135	cilium-2cvcr
		136	csi-nks-node-4gst4
		137	nks-nas-csi-node-fjlj4
		138	nks-nodelocalproxy-7ssrv
		139	nodelocaldns-rtvhd

□ 남부 권(bok-now-krs)

구분	Server	POD	
chanel	services-w-2nx5	1	chanel-89bc44d78-mqp9d
	system-w-2kk1	2	chanel-89bc44d78-2j7z4
confluent	kafka-service-w-2oqb	3	confluent-operator-7556bd4b88-p9wjl
		4	schemaregistry-2
	kafka-service-w-2oqc	5	schemaregistry-1
	kafka-service-w-2oqd	6	schemaregistry-0
	kafka-w-2oq8	7	confluent-kafka-dev-0
	kafka-w-2oq9	8	confluent-kafka-dev-1
	kafka-w-2oqa	9	confluent-kafka-dev-2
	zookeeper-w-2oq5	10	zookeeper-1
	zookeeper-w-2oq6	11	zookeeper-0
	zookeeper-w-2oq7	12	zookeeper-2
default	system-w-2kk1	13	nginx-75d544d79b-cpn4p
elastic	database-w-2nx8	14	rtgs-elasticsearch-759f95cdc4-vwqtf
	database-w-2nx9	15	rtgs-kibana-9fbfc7d88-k6kll
hazelcast	database-w-2nx8	16	rtgs-logstash-6f8fbd4d56-xdfd2
	database-w-2nxa	17	rtgs-hazelcast-mc-5d84cf85f7-5zwvx
mongodb	database-w-2nx8	18	rtgs-hazelcast-6b54fcd9f9-svdff
		19	example-mongodb-2
	database-w-2nx9	20	mongodb-kubernetes-operator-75fb54bf8-b56bj
	database-w-2nxa	21	example-mongodb-0
kube-system	database-w-2nx8	22	example-mongodb-1
		23	cilium-mp7cr
		24	csi-nks-node-xmsrn
		25	nks-nas-csi-node-5b2gx
		26	nks-nodelocalproxy-45r69
	database-w-2nx9	27	nodelocaldns-mt7k2
		28	cilium-wqjmd
		29	csi-nks-node-2njgs
		30	nks-nas-csi-node-sgwzj
		31	nks-nodelocalproxy-bd7fw
	database-w-2nxa	32	nodelocaldns-7w4np
		33	cilium-wd5sg
		34	csi-nks-node-vkx7
		35	nks-nas-csi-node-cw4bc
		36	nks-nodelocalproxy-r8gzx
	kafka-service-w-2oqb	37	nodelocaldns-wz2mm
		38	cilium-2rr7r
		39	csi-nks-node-x7hfl
		40	nks-nas-csi-node-dq4fw
		41	nks-nodelocalproxy-pjlqs
	kafka-service-w-2oqc	42	nodelocaldns-l7lp6
		43	cilium-bnrz6
		44	csi-nks-node-rgqvc
		45	nks-nas-csi-node-c9wft
		46	nks-nodelocalproxy-h4f9x
	kafka-service-w-2oqd	47	nodelocaldns-5hgxy
		48	cilium-tj7wc
		49	csi-nks-node-xn5c2
		50	nks-nas-csi-node-shg5w
		51	nks-nodelocalproxy-2k6kf
	kafka-w-2oq8	52	nodelocaldns-4f4qv
		53	cilium-6kgqh
		54	csi-nks-node-lsnzx
		55	nks-nas-csi-node-kwdhs
		56	nks-nodelocalproxy-brkxj
	kafka-w-2oq9	57	nodelocaldns-629s7
		58	cilium-x8qgl
		59	csi-nks-node-9zh6n
		60	nks-nas-csi-node-zjszk
		61	nks-nodelocalproxy-6xtzh
	kafka-w-2oqa	62	nodelocaldns-zchmq
		63	cilium-r92zr
		64	csi-nks-node-qvpx
		65	nks-nas-csi-node-xbmgb
		66	nks-nodelocalproxy-tg4jj

구분	Server	POD	
	services-w-2nx5	67	nodelocaldns-mj6rv
		68	cilium-8ssqv
		69	csi-nks-node-542bm
		70	nks-nas-csi-node-fl9v8
		71	nks-nodelocalproxy-pcnxf
	services-w-2nx6	72	nodelocaldns-6pdfb
		73	cilium-fw9d7
		74	csi-nks-node-skptq
		75	nks-nas-csi-node-mt7d6
		76	nks-nodelocalproxy-9lz5d
	services-w-2nx7	77	nodelocaldns-48dzd
		78	cilium-c4cgc
		79	csi-nks-node-2smkv
		80	nks-nas-csi-node-pfjbj
		81	nks-nodelocalproxy-lkktk
	system-w-2kk1	82	nodelocaldns-rn6lq
		83	cilium-grtr8
		84	cilium-monitor-tpqcl
		85	cilium-operator-6b88b94f44-rqzrf
		86	csi-nks-node-d8n7h
		87	konnnectivity-agent-7d6fcdfdb6-7469k
		88	nks-nas-csi-node-wwqx7
	system-w-2kk2	89	nks-nodelocalproxy-fhz2r
		90	nodelocaldns-6kmdx
		91	cilium-jzqc2
		92	cilium-monitor-4c4tt
		93	coredns-6b7d7cd855-46xp7
		94	csi-nks-controller-595d6c594c-gpq45
		95	csi-nks-node-vwhsr
	system-w-2kk3	96	konnnectivity-agent-7d6fcdfdb6-hgc2s
		97	nks-nas-csi-node-4mdsr
		98	nks-nodelocalproxy-5gwc5
		99	nodelocaldns-2rw4c
		100	cilium-monitor-qwfj7
		101	cilium-msfg5
		102	cilium-operator-6b88b94f44-mm2qm
		103	coredns-6b7d7cd855-rl8q6
		104	csi-nks-node-xfrrl
		105	dns-autoscaler-6cb646c49b-wcn92
	zookeeper-w-2oq5	106	konnnectivity-agent-7d6fcdfdb6-jzd9h
		107	konnnectivity-agent-autoscaler-5878885c99-krpnn
		108	metrics-server-d9f4f7797-rvpbm
		109	nks-metric-sender-b86f89f57-54nxf
		110	nks-metric-sender-b86f89f57-7kccc
	zookeeper-w-2oq6	111	nks-nas-csi-controller-555b99b7d5-xdbcq
		112	nks-nas-csi-node-wrk2g
		113	nks-nodelocalproxy-v6mxf
		114	nodelocaldns-v9ss5
		115	snapshot-controller-0
	zookeeper-w-2oq7	116	cilium-blh7c
		117	csi-nks-node-5l8d4
		118	nks-nas-csi-node-cxb9v
		119	nks-nodelocalproxy-n2vmj
	zookeeper-w-2oq6	120	nodelocaldns-qclc2
		121	cilium-8mn4t
		122	csi-nks-node-sd6gs
		123	nks-nas-csi-node-twcdw
	zookeeper-w-2oq6	124	nks-nodelocalproxy-xxgc8
		125	nodelocaldns-kpjnb
		126	cilium-z2t4g
	zookeeper-w-2oq7	127	csi-nks-node-fkbvz
		128	nks-nas-csi-node-wqvfm
		129	nks-nodelocalproxy-ddswd
		130	nodelocaldns-h9mk2

쿠버네티스 구성 현황

Namespace		K8s Resource		
		Type	Name	설정내용
ingress-nginx		Service	ingress-nginx-controller	Load Balancer
		Deployments		replicas Ingress Controller(nginx)
		ConfigMap		Nginx 설정 정보 포함
services		Service	chanel-svc	ClusterIP(K8s 내부에서만 보임)
		Deployments	chanel	replicas2, rolling update strategy, readiness, liveness, tolerations
			dior	
			prada	replicas1, readiness, liveness, tolerations
			hermes	
	Secrets		regcred	kubernetes.io/dockerconfigjson
	ingress		chanel-ingress	전문송수신용 웹서버 설정
클러스터 (Kafka confluent)	Service		confluent-operator	ClusterIP(K8s 내부)
			zookeeper	
			confluent-kafka-dev	
	Deployments		confluent-operator	version 0.1033.33, tolerations
			confluent-operator-licensing	Opaque, 컴플루언트 라이선스 정보
	Secrets		helm.release.v1.confluent-operator	Helm Release Secret
			data-controlcenter	kafka 관리 및 모니터링 데이터 저장
	PVC		data-confluent-kafka-dev	kafka 브로커 로그 데이터 저장
			data-zookeeper	kafka 메타데이터 및 상태 정보 저장
			txnlog-zookeeper	트랜잭션 로그 저장
			controlcenter-init-config	kafka 클러스터 정보, 브로커 연결 설정
	ConfigMap		controlcenter-shared-config	kafka 클러스터 모니터링 설정
			schemaregistry-init-config	kafka 데이터 스키마 초기 설정
			schemaregistry-shared-config	kafka 데이터 스키마 공유 설정
			confluent-kafka-dev-init-config	kafka 클러스터 초기화 설정
			confluent-kafka-dev-shared-config	kafka 클러스터 개발환경 설정
			zookeeper-init-config	kafka 클러스터 메타데이터 설정
			zookeeper-shared-config	kafka 클러스터 메타데이터 설정
ELK stack	Service		rtgs-logstash	ClusterIP(K8s 내부)
			rtgs-kibana	
			rtgs-elasticsearch	
	Deployments		rtgs-logstash	replicas 1, rolling update strategy, readiness, liveness, tolerations
			rtgs-elasticsearch	
	Secrets		regcred	kubernetes.io/dockerconfigjson
IMDG (hazelcast)	Service		rtgs-hazelcast-mc	ClusterIP(K8s 내부에서만 보임)
			rtgs-hazelcast	
	Deployments		rtgs-hazelcast-mc	hazelcast/management-center:5.5.0
NoSQL DB (mongodb)			rtgs-hazelcast	hazelcast/hazelcast:5.5.0
	Service		mongodb-svc	ClusterIP(K8s 내부에서만 보임)
	Deployments		mongodb-kubernetes-operator	quay.io/mongodb/mongodb-kubernetes-operator:0.11.0
	Stateful Sets		mongodb	replicas3, readiness-pod, tolerations, mongodb-community-server: 6.0.5-ubi8
k6			data-volume-mongodb-#	storage: 10Gi, ReadWriteOnce
			logs-volume-mongodb-#	storage: 10Gi, ReadWriteOnce
	operator	Service	k6-operator-controller-manager-metrics-service	ClusterIP(K8s 내부에서만 보임)
		Deployments	k6-operator-controller-manager	ghcr.io/grafana/k6-operator:controller-v 0.0.17
	worker	Service	k6-sample-service-#	ClusterIP(K8s 내부에서만 보임)
		Job	k6-sample-#	부하 테스트 실행 설정
			load-test-script	부하 테스트 설정 및 스크립트
		ConfigMap		

개발환경 구성 리소스 파일

구분		소스코드	기능요약	
docker		docker.md	Docker 문서 설명	
		compose.yaml	컨테이너 기반의 Elasticsearch, Logstash, Kibana, MongoDB, Kafka, Hazelcast, 서비스 배포 관리	
		k6-compose.yaml	부하테스트 환경을 구축(K6를 실행하고 결과를 InfluxDB에 저장, Grafana를 통해 시각화)	
elasticsearch	elasticsearch-setup-compose.yaml		Elasticsearch, Kibana의 SSL 인증서 생성 및 초기 비밀번호 검증, 설정파일의 권한 관리	
	logstash		Dockerfile	특정 설정과 파이프라인 구성을 커스터마이징하여 Logstash 컨테이너 이미지 생성
	config	logstash-sample.conf		Beats로부터 데이터를 수집하여 Elasticsearch로 전송하는 로그처리 파이프라인을 구성
		jvm.options		JVM에서 실행되는 Elasticsearch, Logstash의 메모리, GC, 인코딩 보안, 로깅 및 성능 최적화
		startup.options		Logstash 서비스의 초기화 및 실행환경 설정(Java 경로, 메모리 설정, GC 로그, 사용자 및 그룹 권한, 파일 처리 제한 등을 정의)
		log4j2.properties		Logstash의 로깅 환경을 정의하고, 텍스트 및 JSON 형태로 로그의 콘솔 출력을 설정
		log4j2.file.properties		Logstash의 로그파일(텍스트 및 JSON 형식) 관리(롤링 및 압축 정책을 통해 효율적으로 관리)
		logstash.yml		Logstash의 HTTP API 수신 설정, Elasticsearch를 통한 Logstash의 상태 및 성능 모니터링 구성
		pipelines.yml		Logstash에서 실행할 파이프라인 정의
	pipeline	logstash.conf		Elasticsearch에 실시간으로 데이터를 수집 및 저장하여 로그 분석 및 시각화
	grafana	dashboard	2587_rev3.json	InfluxDB에 저장된 부하 테스트 데이터를 시각화하여 대시보드 형태로 모니터링
	hazelcast	migration	init.sql	Hazelcast 환경에서 계좌 정보 및 자금 이체 데이터를 저장하는 IMap을 생성하고 초기화
			sample.sql	테이블(IMap)에서 모든 데이터를 조회
	mongodb		init.script	MongoDB에 금융기관 계좌 정보를 초기화
	influxdb		influxdb.conf	시계열 DB로 K6의 부하테스트 결과를 저장
	k6	scripts	test.js	부하테스트를 실행하고 수집된 메트릭을 10초 간격으로 InfluxDB에 저장(Grafana는 InfluxDB에서 데이터를 대시보드로 시각화)
k8s(Kubernetes 클러스터)			MogoDB.yaml	클러스터의 MongoDB에 대한 접근 인증 관리
Deployments	database	Elasticsearch.yaml		Elasticsearch의 Kubernetes 클러스터 배포 관리
		Hazelcast.yaml		Hazelcast의 Kubernetes 클러스터 배포 관리
		HazelcastMC.yaml		Hazelcast Management Center(MC) 관리
		Kafka.yaml		Kafka의 Kubernetes 클러스터 배포 관리
		Kibana.yaml		Kibana(Elasticsearch 데이터의 시각화 및 대시보드 관리 도구)의 Kubernetes 클러스터 배포 관리
		Logstash.yaml		Logstash의 Kubernetes 클러스터 배포 관리
	services	Chanel.yaml		chanel의 Kubernetes 클러스터 배포 관리
		Dior.yaml		dior의 Kubernetes 클러스터 배포 관리
		Hermes.yaml		hermes의 Kubernetes 클러스터 배포 관리
		Prada.yaml		prada의 Kubernetes 클러스터 배포 관리
namespaces		sandbox.yaml	sandbox의 Kubernetes 클러스터 생성	

<참고1>

쿠버네티스(Kubernetes)

- 쿠버네티스(Kubernetes, 줄여서 K8s)는 컨테이너화된 애플리케이션의 배포, 확장, 관리 등을 자동화하기 위한 오픈소스 플랫폼으로 다수의 컨테이너*를 효율적으로 운영하고 관리할 수 있는 컨테이너 오케스트레이션 도구**

* 컨테이너(Container)는 애플리케이션과 그 실행 환경을 격리된 단위로 패키징하여 어디서든 동일하게 실행하는 가상화 기술

** 구글에서 시작한 프로젝트로, 현재는 클라우드 네이티브 컴퓨팅 재단(CNCF)에서 관리

□ 필요성

- ① 컨테이너의 수 증가 : 하나의 애플리케이션에서 수백, 수천 개의 컨테이너가 생성되며, 이를 수동으로 관리하는 것은 비효율적
- ② 애플리케이션 및 자원 확장 : 사용량에 따라 애플리케이션을 확장하거나 축소해야 하는데, 이를 자동화하지 않으면 운영 비용이 증가
- ③ 고가용성과 장애 복구 : 애플리케이션이나 서버가 다운되었을 때, 이를 자동으로 복구하고 가용성 보장이 필요
- ④ 복잡한 네트워킹 이슈 : 다수의 컨테이너 간 통신, 외부 접근, 로드 밸런싱 등의 네트워킹 문제 해결이 필요

⇒ 쿠버네티스는 대규모 컨테이너 환경에서 안정적이고 효율적인 운영을 지원

□ 특징

- ① 자동화된 배포 및 복구 : 애플리케이션 배포, 업그레이드, 롤백, 복구를 자동화
- ② 확장성(스케일링) : 컨테이너의 수를 사용량에 따라 자동으로 증가/감소
- ③ 서비스 디스커버리와 로드 밸런싱 : 서비스 간 통신을 쉽게 할 수 있도록 자동으로 DNS 이름 또는 IP 주소를 할당하고 로드 밸런싱을 지원
- ④ 셀프 힐링(Self-Healing) : 문제가 발생한 컨테이너를 자동으로 재시작하거나 교체
- ⑤ 환경 독립성 : 클라우드, 온프레미스, 하이브리드 환경 등 인프라에서 동일하게 동작
- ⑥ 구성 설정 : YAML 또는 JSON 파일로 애플리케이션과 클러스터의 상태를 정의

□ 적용 사례

- ① 마이크로서비스 아키텍처 : 여러 독립적인 서비스를 컨테이너로 분리하여 관리
- ② CI/CD 파이프라인 : 지속적인 배포를 자동화하여 SW 업데이트를 신속히 수행
- ③ 다중 클라우드 환경 : 다양한 클라우드 플랫폼에서 애플리케이션을 통합 관리
- ④ 배치 처리 작업 : 데이터 분석, 머신러닝 모델 학습 등 대규모 배치 작업 실행

컨테이너 기술 비교

기능비교	도커(Docker)	쿠버네티스(Kubernetes)
컨테이너 생성 실행	애플리케이션과 종속성을 패키징하고 컨테이너로 실행	컨테이너 생성은 Docker(또는 다른 런타임)를 통해 처리
오케스트레이션	단일 컨테이너의 배포 및 실행 관리	다수의 컨테이너를 배포, 스케일링, 로드 밸런싱, 복구 관리
네트워킹	단일 컨테이너의 네트워크 연결	여러 컨테이너 간 네트워크 설정 및 서비스 디스커버리 관리
확장성	컨테이너 단위에서 수동으로 관리	컨테이너의 자동 확장 및 축소 지원

<참고2>

InfluxDB

□ InfluxDB는 시계열 데이터베이스(TSDB: Time-Series Database)로 실시간 데이터 수집, 저장 및 분석에 특화된 데이터베이스. InfluxData 사에서 개발했으며, 오픈 소스 버전과 상업용 엔터프라이즈 버전이 있음

□ 필요성

- ① 실시간 데이터 처리(서버 상태, IoT 장치 데이터 등 대량의 실시간 데이터)
- ② 성능 모니터링(애플리케이션 및 시스템의 성능 모니터링 및 분석)
- ③ 빅데이터 분석(대규모 데이터의 트렌드 분석)

□ 장점

- ① 고성능 데이터 처리(수백만 개의 데이터 포인트를 초당 삽입 및 조회)
- ② 유연한 데이터 구조(Tag와 Field 기반의 데이터 인덱싱으로 빠르게 검색 및 집계)
- ③ 보존 정책(오래된 데이터를 자동으로 삭제 및 축약하여 효율적인 저장소 관리)
- ④ Grafana와 쉽게 연동해 시각화 및 대시보드 생성에 탁월
- ⑤ 간편한 설치 및 관리(Docker, 바이너리 등 다양한 방법으로 쉽게 배포 가능)

□ 단점 및 해결방안

- ① 데이터 모델 및 쿼리 복잡성 → InfluxQL, Flux 튜토리얼, InfluxDB University 활용
- ② 확장성 및 클러스터링 부족 → InfluxDB Enterprise, CQ/Kapacitor 사용
- ③ 데이터 정합성 부족 → 메시지 큐(RabbitMQ, Kafka) 사용, 유니크 ID 부여
- ④ 장애 복구 및 고가용성 → 클러스터링, 데이터 백업 및 Hot/Warm 스토리지 도입
- ⑤ 장기 데이터 보존 및 저장 공간 문제 → 디스크 백업

□ 활용 사례

- ① 시스템 및 애플리케이션 모니터링 : CPU 사용량, 메모리, 디스크 I/O 등 실시간 시스템 메트릭 수집
- ② IoT 데이터 처리 : IoT 센서에서 수집한 데이터의 실시간 분석 및 저장
- ③ 금융 데이터 분석 : 주식 거래 및 금융 데이터의 변동 추적 및 분석
- ④ 스마트 팩토리 : 생산 라인의 상태 및 기계 데이터 모니터링을 통한 장애 예측

TSDB 데이터베이스 비교

기능비교	InfluxDB	Prometheus	TimescaleDB
데이터처리	시계열 전용 DB	시계열 전용 DB	PostgreSQL 기반
처리 속도	매우 빠름	빠름	보통
쿼리 언어	InfluxQL, Flux	PromQL	SQL
데이터보존 정책	기본 제공	기본 제공	SQL 기반
클러스터링	Enterprise 버전 지원	Federation	PostgreSQL 확장 지원
시각화도구	Grafana, Chronograf	Grafana	Grafana, 기본 SQL

<참고3>

ELK(Elasticsearch, Logstash, Kibana)

□ ELK는 Elasticsearch, Logstash, Kibana로 구성된 오픈소스 로그 및 데이터 분석 스택. 클라우드 환경 및 대규모 애플리케이션에서 서버, 애플리케이션, 네트워크 장비 등에서 실시간으로 발생하는 로그와 이벤트 데이터를 수집, 저장, 분석 및 시각화

① Elasticsearch : 역색인(Inverted Index) 기반의 분산 검색 및 분석 엔진

- 대량의 데이터를 실시간으로 저장, 검색 및 분석하는 데 최적화
- 로그 데이터, 메트릭, 문서, 이벤트 등을 빠르게 검색
- JSON 기반으로 데이터를 저장하고, 강력한 검색 쿼리를 지원
- 확장성 : 데이터가 증가하면 클러스터를 쉽게 확장
- 고성능 검색 : 빠른 검색 속도로 대규모 데이터에서 패턴을 찾는 데 유용
- REST API를 통해 다른 애플리케이션과 쉽게 연동

② Logstash : 다양한 소스 데이터를 수집, 변환 및 전송하는 데이터 파이프라인 도구

- 수집된 데이터는 Elasticsearch에 저장되거나 다른 저장소로 전송
- 다양한 입력 지원 : 파일, 데이터베이스, 메트릭, 클라우드 서비스, 네트워크 장비 등 다양한 소스로부터 데이터 수집
- 필터링 및 변환 : 데이터 포맷을 변경하고, 필요한 필드의 추출과 변환 기능
- 확장성 및 유연성 : 파이프라인을 유연하게 설계하고 확장

③ Kibana : Elasticsearch에 저장된 데이터를 시각화 및 분석하는 도구

- 대시보드, 차트, 그래프, 맵 등으로 로그 및 메트릭 데이터를 시각적으로 표현
- 실시간 모니터링 : 실시간으로 로그 및 메트릭을 모니터링
- 쿼리 및 필터링 : Elasticsearch 데이터를 쿼리하고 필터링하여 시각화
- 경고(Alerting) : 특정 조건이 충족되면 알림을 설정하여 장애를 사전에 감지

□ ELK 스택의 데이터 흐름

- ① 로그 및 데이터 수집(Logstash) : 파일, 시스템 로그 등에서 데이터를 수집
- ② 데이터 처리 및 전송(Logstash) : 데이터를 필터링 변환 후 Elasticsearch에 전송
- ③ 데이터 저장 및 분석(Elasticsearch) : 데이터를 저장하고 실시간으로 검색 및 분석
- ④ 시각화 및 모니터링(Kibana) : Elasticsearch의 데이터를 시각화 및 분석

□ ELK의 장점

- ① 오픈 소스 : 무료로 사용할 수 있으며, 커뮤니티가 활발하게 발전
- ② 확장성 및 유연성 : 대규모 데이터에 맞게 쉽게 확장
- ③ 실시간 분석 : 데이터 수집 및 분석이 실시간으로 이루어져 빠르게 대응
- ④ 다양한 플러그인 : 다양한 플러그인을 제공하여 다양한 데이터 소스를 지원

□ ELK의 단점

- ① 복잡성 : 스택이 여러 구성 요소로 이루어져 있어 설치 및 구성 작업이 복잡
- ② 리소스 사용량 : 대량의 데이터가 수집될 경우 CPU 및 메모리 사용량 증대

<참고4>

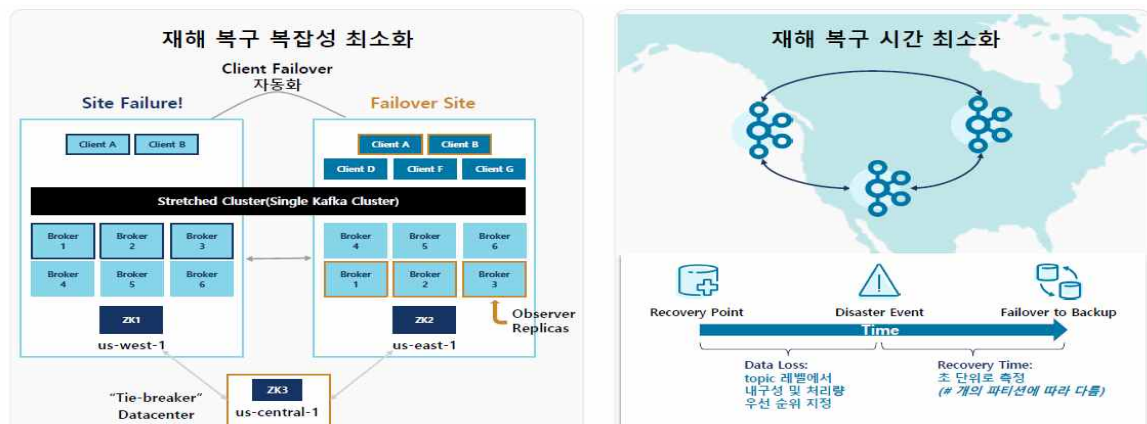
메시지 브로커(Kafka)

- 메시지 브로커(Message Broker)란 송신자(Producer)가 보낸 데이터(Message)를 수신자(Consumer)에게 전달하는 미들웨어로 메시지 전송순서와 안정적인 전달을 보장하는 기술
 - 다른 애플리케이션 간의 중개자 역할을 함으로써 수신자의 위치, 활성여부, 수신자 수 등이 불확실한 경우에도 송신자가 메시지를 발행 가능
 - 병렬적으로 수신하는 메시지의 순차적인 전송을 보장하는 메시지 전송순서 보장기능, 수신자의 메시지전달 확인신호(ack)를 활용한 메시지전달 보장기능, 메시지 재전송 기능
 - 종류로는 IBM MQ, 아파치 카프카, ActiveMQ, RabbitMQ, 솔라스 PubSub+ 등
- 링크드인(LinkedIn)에서 개발한 **카프카(Kafka)**는 대용량 실시간 스트리밍 데이터 처리를 빠르고 확장 가능하게 설계된 분산형 메시징 플랫폼(현재 ECB의 TIPS 및 금융결제원의 전자금융공동망에서 사용중)
 - 카프카를 통한 비동기 메시징 처리로 서비스를 느슨하게 결합(loosely-coupled)함으로써 아키텍처를 단순화하고 서비스 영향도를 분리 가능

□ 카프카의 주요기능

구분	내용
프로듀서와 컨슈머의 분리	메시지를 주고받는 역할을 완벽하게 분리하여 송수신 어느 한쪽의 시스템에서 문제가 발생하더라도 연쇄작용으로 문제가 발생할 확률이 낮음
멀티 프로듀서, 멀티 컨슈머	하나의 프로듀서가 여러 개의 토픽을 생성할 수 있고 하나의 컨슈머가 여러개의 토픽을 받을 수 있음
디스크에 메시지 저장 가용성	처리가 끝난 메시지를 바로 삭제하는 MQ와 달리 카프카는 메시지를 읽어 가더라도 정해져 있는 보관주기 동안 디스크에 메시지를 저장
확장성	장래 이벤트 복구 즉시 또는 과거의 이벤트 처리
높은 성능	하나의 카프카 클러스터는 3대의 브로커로 시작해 수십 대의 브로커로 확장가능하면 서비스의 중단없이 온라인 상태에서 작업이 가능
	링크드인은 2015년 기준 1조개의 메시지를 생성하고 카프카를 이용하여 일일 1페타바이트 이상의 데이터를 처리

Kafka 사용버전(Confluent)의 Multi-Region Clusters 개념



<출처:「소액RTGS시스템 구축관련 IT요소기술(메시지브로커: 카프카)」(결제시스템팀-1205, 2023.11.17)>

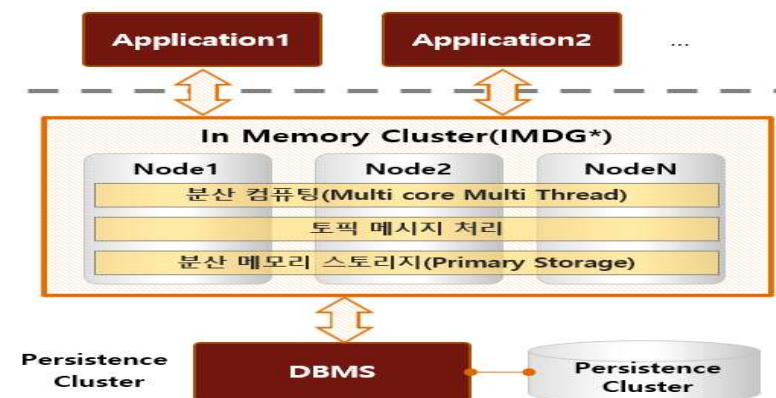
<참고5>

인메모리 컴퓨팅(Hazelcast)

- 인메모리 컴퓨팅은 디스크(HDD, SSD 등)에 데이터를 저장함으로써 발생하는 시스템 병목현상을 해결하기 위하여 메모리에 데이터를 저장하고 처리함으로써 처리속도를 개선하는 기술
 - 인메모리 컴퓨팅을 위한 메모리 데이터 저장소 In-Memory DB(IMDB)와 In-Memory Data Grid(IMDG)를 활용할 수 있으며, IMDG는 메모리를 공유하여 클러스터에서 동작하는 대규모 응용시스템들을 지원하기 위해 사용
- **헤이즐캐스트(Hazelcast)** IMDG는 여러 서버를 클러스터로 구성하여 메모리상에서 데이터를 공유하고 분산 병렬처리 및 저장, 이벤트 기반 프로세싱을 통한 분산 컴퓨팅 기능을 제공 (현재 금융결제원의 전자금융공동망에서 사용중)
- 헤이즐캐스트의 주요기능

구분	내용
인메모리 기반 데이터 처리	저장공간을 디스크가 아닌 메모리(RAM)를 활용 Key-Value 기반의 인메모리 데이터 저장소를 통해 빠른 액세스 및 쿼리 계층을 활용 High Density Memory Store 지원으로 서버의 가용 메모리를 모두 데이터 저장소로 사용할 수 있으며, GC로 인한 성능저하를 제거
클러스터 기반 분산저장	다수의 컴퓨터 메모리(RAM)을 클러스터링하여 하나의 큰 메모리 저장소로 활용함으로써 Disk I/O 병목현상 제거 여러 메모리 노드에 데이터를 분산/복제함으로써 한 노드에 장애발생시 다른 노드에서 처리하는 무중단 서비스 가능 시스템 중단없이 노드를 추가(Scale-Out) 가능
데이터 복원력 및 연속성 확보	이벤트 발생 시 Snapshot을 통해 전체 데이터에 대한 디스크 저장 및 빠른 복구(Hot Restart Store) WAN Replication을 통해 원격지 클러스터 동기화(단방향, 양방향) 가능 RDBMS 연계 저장을 통해 데이터 Persistency 확보

In-Memory Data Grid(IMDG) Cluster 개념



* IMDG: In Memory Data Grid

<출처:「소액RTGS시스템 구축관련 IT요소기술(인메모리 컴퓨팅: 헤이즐캐스트)」(결제시스템팀-1022, 2023.10.6)>

<출처:「인메모리 컴퓨팅의 개념 정리 및 사례 조사」(RTGS시스템팀-90, 2024.7.24)>

<참고6>

NoSQL(MongoDB)

□ NoSQL(Not only Structured Query Language)은 SQL만을 사용하지 않는 비관계형 및 분산 데이터 저장소 기술로 관계형 데이터베이스(RDB)를 사용하지 않는다는 의미가 아닌 여러 유형의 데이터 모델을 사용하는 DBMS를 의미

- 빅데이터, IoT, 웹2.0 환경의 등장으로 실시간 데이터와 트래픽의 양이 기하급수적으로 증가함에 따라 단일 기기에서 실행되도록 설계된 RDBMS에 큰 비용이 소요됨에 따라 클러스터 환경에 적합한 NoSQL이 등장

□ NoSQL의 주요기능

구분	내용
스키마리스 (Schema-less)	고정된 스키마가 없어 데이터 구조의 빠른 변화에 유연하게 대응, 비구조화된 다양한 형태의 데이터를 효율적으로 저장 및 관리
유연한 데이터 모델 (Flexible Data Model)	문서, 키-값, 컬럼 패밀리, 그래프 등을 지원하여, 애플리케이션의 요구사항에 따라 최적의 모델을 선택
유연한 쿼리 모델 (Flexible Querying)	SQL과 다른 쿼리 언어의 사용 없이도 복잡한 쿼리를 실행할 수 있으며, 애플리케이션 요구사항에 맞춰 쿼리를 최적화
수평적 확장 (Horizontal Scalability)	데이터 노드를 추가함으로써 선형적인 성능 향상을 달성할 수 있어, 대규모 데이터 세트를 경제적으로 처리할 수 있음
분산 처리 (Distributed Computing)	데이터와 데이터베이스 연산을 여러 노드에 걸쳐 분산 처리하여 고가용성과 부하 분산에 기여
빠른 데이터 처리 (Fast Data Access)	키-값 저장, 인덱싱, 비동기 복제 등의 기능을 통해 빠른 데이터 읽기와 쓰기를 지원
고가용성 (High Availability)	데이터 복제, 자동 장애 복구 등의 기능을 통해 시스템의 지속적인 가용성을 보장
대규모 데이터세트 처리 (Big Data Processing)	빅데이터 세트를 효율적으로 처리하고 분석할 수 있도록 설계

□ **MongoDB**는 NoSQL 기반의 문서형(Document-Oriented) 데이터베이스로, 데이터를 JSON과 유사한 BSON(Binary JSON) 형식으로 저장(현재 FRB의 FedNow에서 사용중)

- MongoDB 8.0 버전은 대량의 데이터 전환과 쿼리성능이 개선되고, LLM과 연동하여 AI처리도 가능

NoSQL 데이터베이스 비교

기능비교	MongoDB	Redis, Cassandra
데이터 모델	문서(Document) 기반	키-값(Key-Value), 열(Column) 기반
스키마 유연성	스키마리스	스키마리스
쿼리 기능	복잡한 쿼리와 집계 프레임워크 지원	일부는 간단한 키-값 조회만 지원
파일 저장	GridFS 지원	파일 저장 기능이 없는 경우가 많음
트랜잭션 지원	ACID 트랜잭션 지원	일부는 트랜잭션 미지원

<출처:「NoSQL의 기본개념 정리」(RTGS시스템팀-51, 2024.5.7)>

<출처:「부서주관직무연수결과보고(MongoDB)」(RTGS시스템팀-118, 2024.10.4)>