

1. 개요

マイクロサービス 아키텍처(MSA) 전환에서 데이터베이스 분리는 가장 중요한 면에서도 복잡한 과정 중 하나입니다.

기존 모놀리식 서비스를 통합 데이터베이스로 서비스별로 분리하는 것은 단순한 기술적 작업이 아니라 비즈니스 도메인 이해와 데이터 의존성 분석을 기반으로 한 전략적 접근이 필요합니다.

Database per Service 패턴을 중심으로 한 체계적인 접근 방법과 실제 구현 시 고려해야 할 사항들을 다룹니다.

2. 데이터베이스 현황 분석 프레임워크

2.1 분석 목표 설정

데이터베이스 현황 분석은 다음과 같은 목표를 달성하기 위해 수행됩니다:

- 현재 데이터베이스 구조와 서비스 간 의존성 파악
- MSA 분리 시 데이터 경계 및 소유권 정의
- 분리 과정에서 발생할 수 있는 리스크 식별
- 점진적 분리 전략 수립을 위한 우선순위 결정

2.2 데이터베이스 현황 분석 템플릿

데이터베이스 현황 분석 기본 템플릿			
분석 항목	상세 내용	수집 방법	비고
데이터 정보	데이터형, 용도, 데이터 블록, 상장률	DB 스키마 분석, 시스템 문서 검토	비즈니스 중요도 표시
데이터 간 관계	FK 관계, 주인 모델, 트랜잭션 범위	ERD 분석, 데이터 품질 분석	의존성 강도 평가
접근 패턴	CRUD 비율, 동사성 요구사항, 성능 특성	APM 도구, 퀘리 성능 분석	서비스별 분석
데이터 일관성	트랜잭션 관계, 일관성 요구사항	비즈니스 프로세스 분석	ACID vs BASE 판단
기술적 제약	스트리거 인腱, 인腱, 뷰, 프로시저	DB 객체 분석	분리 시 영향도 평가

2.3 도메인별 데이터 분류

DDD(Domain-Driven Design) 원칙에 따라 데이터를 비즈니스 도메인별로 분류하는 것이 MSA 분리의 첫 번째 단계입니다.

도메인별 데이터 분류 템플릿				
도메인	핵심 엔티티	관련 테이블	비즈니스 기능	외부 의존성
사용자 관리	User, Role, Permission	users, roles, permissions, user_roles	인증, 인가, 프로필 관리	없음
주문 관리	Order, OrderItem	orders, order_items, payment_info	주문 생성, 결제 처리	사용자, 상품, 결제
상품 관리	Product, Category	products, categories, inventory	상품 등록, 재고 관리	없음
결제 관리	Payment, Invoice	payments, invoices, refunds	결제 처리, 환불 관리	주문, 사용자

3. 데이터베이스 분리 전략

3.1 Database per Service 패턴

Database per Service 패턴은 각 마이크로서비스가 자체 데이터베이스를 소유하고 관리하는 MSA의 핵심 원칙입니다. 이 패턴은 다음과 같은 이점을 제공합니다:

- **논소유 절감:** 서비스 간 데이터베이스 의존성 제거
- **기술적 자유도:** 서비스별 최적화된 데이터베이스 선택 가능
- **독립적 확장:** 서비스별 개발 확장 및 성능 최적화
- **장애 격리:** 한 서비스의 DB 장애에 다른 서비스에 미치는 영향 최소화

주의사항: Database per Service 패턴 적용 시 분산 트랜잭션, 데이터 일관성, 복잡한 조인 위기 등의 문제가 발생할 수 있습니다. 이를 해결하기 위해 SAGA 패턴, CQRS, API Composition 등의 패턴을 함께 고려해야 합니다.

3.2 분리 접근 방법

데이터베이스 분리는 다음 세 가지 접근 방법 중 하나를 선택할 수 있습니다:

3.2.1 데이터베이스 우선 분리

애플리케이션 코드는 유지 가능한 채 데이터베이스를 먼저 분리하는 방식입니다.

항목	단점	적용 시나리오
• 소비자 역할 최소화	• 단기적인 데이터 처리 필요	• 가능한 경우 최종 일관성 후회
• 반복 풀이 일관성	• 복잡한 데이터 처리 필요	• 신규 기간 연장 필요
• 성능 영향 최소화	• 개별 기간 연장	• 학개 시스템 유지

3.2.2 애플리케이션 코드 우선 분리

데이터베이스와 애플리케이션은 공유한 채 애플리케이션 먼저 분리하는 방식입니다.

항목	단점	적용 시나리오
• 빠른 서비스 분리	• 데이터 출판도 유지	• 빠른 서비스 분리 필요
• 독립적 확장 가능	• 전통적인 헤리티지 부족	• 신경한 헤리티지 고려
• 개별 생산성 향상	• 장애 전파 위험	• 단기적 동기화 필요

3.2.3 동시에 분리

데이터베이스와 애플리케이션을 동시에 분리하는 방식입니다.

권장하지 않음: 동시에 분리하는 높은 리스크를 수반하며 예상보다 오랜 시간이 소요됩니다. 가능한 한 단계적 접근을 권장합니다.

3.3 분리 패턴별 구현 방법

3.3.1 논리적 분리 (Schema per Service)

단일 데이터베이스 인스턴스 내에서 서비스별로 스키마를 분리하는 방법입니다.

서비스별 스키마 분리 예시 CREATE SCHEMA user_service; CREATE SCHEMA order_service; CREATE SCHEMA payment_service; CREATE USER 'user_service_user'@'%' IDENTIFIED BY 'password'; GRANT ALL PRIVILEGES ON user_service.* TO 'user_service_user'@'%'; GRANT ALL PRIVILEGES ON order_service.* TO 'order_service_user'@'%'; GRANT ALL PRIVILEGES ON payment_service.* TO 'product_service_user'@'%';

3.3.2 물리적 분리 (Database per Service)

각 서비스가 독립된 데이터베이스 인스턴스를 갖는 방법입니다.

항목	단점	적용 시나리오
• 빠른 서비스 분리	• 데이터 출판도 유지	• 빠른 서비스 분리 필요
• 독립적 확장 가능	• 전통적인 헤리티지 부족	• 신경한 헤리티지 고려
• 개별 생산성 향상	• 장애 전파 위험	• 단기적 동기화 필요

4. 데이터베이스 의존성 분석

4.1 의존성 분석 체크리스트

데이터 분석			
• 외래 키(FK) 관계 매핑	• MSA 분리 시 데이터 경계 및 소유권 정의	• 데이터 품질 분석	• DB 구조 분석
• 주인 퀘리 패턴 분석	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 분석
• 트랜잭션 경계 설정	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 품질 분석
• 데이터 일관성 체크	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 품질 분석
• 단기적인 헤리티지 고려	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 품질 분석
• 단기적인 헤리티지 고려	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 품질 분석
• 단기적인 헤리티지 고려	• 데이터 품질 분석	• 서비스별 품질 분석	• 서비스별 품질 분석

4.2 의존성 해결 방안

4.2.1 강한 의존성이 있는 경우

서비스 간 강한 의존성이 있는 경우 다음과 같은 해결 방안을 고려할 수 있습니다:

의존성 유형	해결 방안	구현 방법
트랜잭션 경계	SAGA 패턴	이번트 기반 보상 트랜잭션
서비스 간 데이터 조회	API Composition	애플리케이션 계층 조인
복잡한 퀘리	CQRS	일기 관리 분리 구조
결조 데이터	데이터 복제	이번트 기반 동기화

4.2.2 얕은 의존성 해결

약한 의존성은 다음과 같은 방법으로 해결할 수 있습니다:

- 데이터베이스 내부에서 필요한 데이터를 중복 저장
- 캐시 활용: 자주 참조하는 데이터를 캐시로 관리
- 배치 동기화: 비슷한 시간대에 데이터를 배치로 동기화
- 이벤트 발행: 데이터 변경 시 이벤트를 발행하여 관련 서비스에 알림

5. 실무 적용 가이드

5.1 분리 순서 결정

데이터베이스 분리 시 다음 기준에 따라 우선순위를 결정합니다:

분리 우선순위 척도 매트릭스
평가 기준
• 높음 (2점)
• 중간 (1점)
• 낮음 (0점)

5.2 단계별 분리 프로젝트

1. 후보 선정: 독립성이 높고 복잡도가 낮은 서비스 선택
2. 프로토 타입 개발: 분리 패턴 검증
3. 성능 테스트: 분리 후 성능 영향 측정
4. 교환 도구: 경험을 바탕으로 프로세스 개선

5.3 분리 후 운영 고려사항

5.3.1 보안 체계

- 서비스별 성능 모니터링: 응답시간, 처리량, 리소스 사용률
- 데이터베이스 모니터링: 연결 수, 퀘리 성능, 리소스 사용량
- 분산 트랜잭션 모니터링: SAGA 패턴 실행 상태
- 데이터 일관성 모니터링: 서비스 간 데이터 동기화 확장

5.3.2 백업 및 복원

- 서비스별 백업 테이블 분리
- Point-in-time 복구: 분산 환경에서 일관성 있는 복구 방안
- 재해 복구 계획: 서비스 별 우선순위를 고려한 복구 구조

5.3.3 보안 관리

- 접근 제어: 서비스별 데이터베이스 접근 관리
- 암호화: 데이터 전송 및 저장 시 암호화
- 강화: 데이터 접근 및 변경 이력 추적

5.4 주요 위험 요소

<table