마이크로 서비스 성숙도 레벨, 체크포인트, 역량, 액티비티

작성자: 유엔진솔루션즈 장진영

구현 관점	Early (Lv1)	Inception (Lv2)	Expanding (Lv3)	Mature (Lv4)
기능 분해	기능과 유즈케이스 단위로 분리 - 비즈니스 역량은 도출한 비즈니스 기능과 유즈케이스 단위로 분리가능	서비스별 인터페이스 정의 - 추출 된 각 유스케이스와 인터페이스를 통해 액세스 할 데이터에 대해 잘 정의 된 인터페이스를 가짐.	도메인 **Context** 분리 **- Ubiquitous language** 가 다른 **bounded context**간의 커뮤니케이션 시, **Anti-corruption layer**를 통해 수행	도메인 기반 이벤트 - 구체화된 보기, 읽기 쓰기를 위한 별도의 모델**(CQRS)** 구축
데이터	**2PC** 사용 가능 **- ACID** 기반의 트랜잭션을 유지합니다. **Canonical Data Model** 를 지향	**Scheme per Service** - 각각의 서비스는 자신만의 **Scheme**를 가짐 - 서비스들과 다중 엔터프라이즈 데이터 저장소 간의 트랜잭션이 적은 조정으로 이루어짐	**DBMS per Service** - 완전히 분산된 데이터 관리 - 서비스별 다른 유형의 **DBMS** 사용이 가능한 폴리글랏 퍼시스턴스를 지향	**Event-Driven Archi.** - 이벤트 기반 데이터 관리, 이벤트 소싱 및 커멘드 쿼리 - 일시적으로 데이터에 일관성이 없는 상태가 존재할 수 있으나, 일정 시간이 지나면, 데이터가 도착하여 다시 **Consistency**를 충족
SW 아키텍처	**UI/UX : Server Side rendering, Session based** 보안, 단일 언어**(eg. Spring framework)**	**UI/UX : Server Side rendering, Session cluster** 적용, **MSA** 지향 언어**(eg. Spring Boot)**	**UI/UX : Client Side rendering, Token based** 보안, **OAuth2,** 폴리글랏 **Language**	**UI/UX : Client Side rendering+MVVM, Token based** 보안, **OAuth2,** 폴리글랏 **Language + Service Mesh**
Infra 스트럭처	지속적인 빌드, 지속적인 통합 운용	지속적인 딜리버리와 배포, 로그의 중앙 집중화	컨테이너 사용**(도커),** 컨테이너 오케스트레이터**(k8s),** 외부 구성**(유레카,** 주키퍼**)**	자동 프로비저닝을 갖춘 **PaaS**기반 솔루션

배포	설치 스크립트 구동, 호스트 당 멀티 서비스 인스턴스	**VM** 당 하나의 서비스 인스턴스 클라이언트 사이드 로드 벨런싱 서버사이드 로드 벨런싱	**Immutable** 서버, 컨테이너 당 하나의 서비스 인스턴스, **blue/green** 배포	멀티 클라우드 및 멀티 데이터 센터 지원
팀구조	개발, **QA,**릴리즈, 운영이 분리된 하나의 기능 팀	공유된 서비스 모델로 팀 공동 작업 내부 소스 공개	서비스별 프로덕트 팀**(PO, UI/UX** 디자이너, 개발자**) Cross Functional**한 플랫폼 팀	업무 기능별 혹은 도메인별 팀들이 모든 관점에서 책임을 수반. **"**네가 구축한 것은 네가 운영합니다**."**

기능분해관점 체크포인트

Level 1: 기능과 유즈케이스 단위로 분리

- [ ] 비즈니스 역량 도출: 비즈니스 도메인을 분석하여 주요 기능과 유즈케이스를 도출하였는가?
- [ ] 기능 정의: 각 기능을 명확하게 정의하고 해당 기능의 범위를 결정하였는가?
- [ ] 유즈케이스 식별: 각 기능에 대한 유즈케이스를 식별하고 이를 단위로 분리하였는가?
- [ ] 의존성 분석: 각 기능과 유즈케이스 간의 의존성을 분석하여 적절한 분리 수준을 결정하였는가?

Level 2: 서비스별 인터페이스 정의

- [ ] 인터페이스 설계: 각 유즈케이스와 관련된 데이터에 대한 인터페이스를 정의하고 필요한 메소드와 매개변수를 명시하였는가?
- [ ] 데이터 액세스 계획: 각 유즈케이스에서 필요한 데이터 액세스 방법을 계획하고 이를 인터페이스에 반영하였는가?
- [ ] 보안 및 인증 고려: 서비스 간의 인터페이스에서 보안 및 인증 요구사항을 고려하여 액세스 제어 방법을 정의하였는가?
- [ ] 에러 핸들링 정의: 인터페이스에서 발생할 수 있는 예외 상황에 대한 처리 방법과 오류 핸들링 전략을 정의하였는가?

Level 3: 도메인 Context 분리

- [ ] Ubiquitous language 정의: 각 bounded context에서 사용될 공통 언어와 용어를 정의하고 이를 문서화하였는가?
- [ ] Boundary 정의: 각 bounded context의 경계를 명확하게 정의하고 외부에서의 커뮤니케이션 방식을 결정하였는가?
- [ ] Anti-corruption layer 구현: bounded context 간의 통신을 관리하기 위해 Anti-corruption layer를 구현하고 외부 인터페이스 변환 및 데이터 변환을 수행하였는가?
- [ ] 도메인 이벤트 식별: bounded context 간에 필요한 도메인 이벤트를 식별하고 이를 통해 비동기적인 상호작용을 구현하였는가?

Level 4: 도메인 기반 이벤트

- [ ] CQRS 모델 설계: 읽기와 쓰기를 위한 별도의 모델을 설계하고 도메인 이벤트를 활용하였는가?
- [ ] 이벤트 스토리지 구현: 도메인 이벤트를 저장하고 관리하기 위한 이벤트 스토리지를 구현하였는가?
- [ ] 이벤트 드리븐 아키텍처 구축: 도메인 이벤트를 기반으로 비즈니스 로직을 처리하는 이벤트 드리븐 아키텍처를 구축하였는가?
- [ ] 이벤트 버전 관리: 도메인 이벤트의 버전을 관리하고 업데이트할 수 있는 메커니즘을 도입하였는가?
- [ ] 이벤트 소싱 및 재생: 이벤트 소싱을 통해 시스템 상태를 재생하고 이벤트 스트림을 통해 상태 변경을 추적하였는가?

각 레벨에서의 개선:

1. Level 1에서는 마이크로서비스의 독립성과 재사용성이 향상되고, 비즈니스 역량을 명확하게 분리하여 책임과 범위를 부여할 수 있습니다.
2. Level 2에서는 마이크로서비스 간의 결합도가 낮아지고 독립적으로 개발하고 배포할 수 있게 됩니다.
3. Level 3에서는 도메인 간의 경계가 명확해지고 공통 언어를 정의하여 도메인 간의 의사소통을 원활하게 할 수 있습니다.
4. Level 4에서는 도메인 기반 이벤트를 통해 비즈니스 도메인의 상태 변경을 추적하고 비동기적인 상호작용을 구현할 수 있으며, CQRS 모델을 도입하여 성능과 확장성을 향상시킬 수 있습니다.

5. 각 레벨별로 개선되는 내용은 해당 레벨 이전에 비해 더 높은 수준의 성숙도와 유연성을 제공하며, 이전 수준에서의 한계를 극복하기 위한 다양한 개선 방법을 제시합니다.

데이터 관점 체크포인트

Level 1: 2PC 사용 가능

- [ ] 트랜잭션 관리: 각 마이크로서비스에서 ACID 기반의 트랜잭션을 유지하기 위한 방법과 프레임워크를 도입하였는가?
- [ ] 데이터 일관성 유지: Canonical Data Model을 도입하여 데이터의 일관성을 유지하고 데이터의 중복 및 모순을 방지하였는가?

Level 2: Scheme per Service

- [ ] 개별 스키마 정의: 각 마이크로서비스는 자체적인 스키마를 가지고 있으며, 각 서비스의 데이터 구조와 필드 정의가 명확하게 이루어졌는가?
- [ ] 다중 엔터프라이즈 데이터 조정: 서비스 간의 다중 엔터프라이즈 데이터 저장소와의 트랜잭션 조정 방식을 정의하고 이를 구현하였는가?
- [ ] 데이터 일관성 유지: 데이터 조정 및 동기화 과정에서 데이터 일관성을 유지하기 위한 메커니즘을 도입하였는가?

Level 3: DBMS per Service

- [ ] 분산 데이터 관리: 각 마이크로서비스는 완전히 분산된 데이터 관리를 수행하고 있으며, 필요에 따라 데이터의 파티셔닝, 샤딩, 복제 등을 구현하였는가?

- [ ] 폴리글랏 퍼시스턴스: 각 마이크로서비스는 자체적으로 다른 유형의 DBMS를 사용할 수 있는 폴리글랏 퍼시스턴스를 구현하였는가?
- [ ] 데이터 접근 및 관리: 각 마이크로서비스에서 필요한 데이터 액세스 방법을 선택하고 이를 효율적으로 관리하고 있으며, 성능과 확장성을 고려하였는가?

Level 4: Event-Driven Architecture

- [ ] 이벤트 기반 데이터 관리: 이벤트 중심 아키텍처를 구축하여 각 마이크로서비스 간의 데이터 흐름을 이벤트 기반으로 관리하고 있는가?
- [ ] 이벤트 소싱 및 커맨드-쿼리 분리(CQRS): 이벤트 소싱을 통해 시스템의 상태를 재생하고 커맨드와 쿼리를 분리하여 비동기적인 데이터 처리를 구현하였는가?
- [ ] 데이터 일관성 관리: 일시적으로 데이터에 일관성이 없는 상태가 존재할 수 있으며, 이를 일정 시간 내에 해소하여 다시 일관성을 충족시키는 메커니즘을 구현하였는가?

소프트웨어 아키텍처 관점 체크포인트

Level 1:

- [ ] UI/UX: Server Side Rendering (SSR): 클라이언트 요청 시 서버에서 페이지를 완전히 렌더링하여 전송하는 방식을 사용합니다. 이를 통해 초기 로딩 속도를 개선하고 SEO에 더 유리한 환경을 제공할 수 있습니다.
- [ ] Session-based 보안: 세션을 사용하여 사용자 인증 및 권한 부여를 처리합니다. 세션은 서버에 저장되어 클라이언트와의 상태를 유지하고 보안을 강화합니다.
- [ ] 단일 언어(예: Spring Framework): 특정 언어 또는 프레임워크에 의존하여 애플리케이션을 개발합니다. 예를 들어, Spring Framework를 사용하여 모노리틱 서비스를 구축하고 운영합니다.

Level 2:

- [ ] UI/UX: Server Side Rendering (SSR): Level 1과 동일하게 서버에서 페이지를 렌더링하여 전송합니다.
- [ ] Session Cluster 적용: 세션 클러스터링을 통해 여러 서버 간에 세션 정보를 공유하여 확장성과 가용성을 개선합니다.
- [ ] MSA 지향 언어(예: Spring Boot): 마이크로서비스 아키텍처에 적합한 언어 또는 프레임워크를 선택하여 개발합니다. Spring Boot와 같은 경량화된 프레임워크를 사용하여 서비스를 독립적으로 구축하고 배포합니다.

Level 3:

- [ ] UI/UX: Client Side Rendering (CSR): 클라이언트에서 페이지 렌더링을 수행하고 필요한 데이터만 서버로부터 요청하여 가져옵니다. 이를 통해 초기 로딩 시간을 줄이고 사용자 경험을 향상시킬 수 있습니다.
- [ ] Token-based 보안, OAuth2: 세션 대신 토큰을 사용하여 사용자 인증과 권한 부여를 처리합니다. OAuth2와 같은 표준 프로토콜을 사용하여 안전한 인증 및 인가를 구현합니다.
- [ ] 폴리글랏 언어: 다양한 언어를 선택하여 각 마이크로서비스에 가장 적합한 언어를 사용합니다. 이를 통해 개발자들은 자신이 가장 잘 알고 있는 언어로 서비스를 개발할 수 있으며, 시스템 전체의 다양성과 유연성을 높일 수 있습니다.

Level 4:

- [ ] UI/UX: Client Side Rendering (CSR) + MVVM: 클라이언트에서 페이지 렌더링 및 상태 관리를 담당하는 MVVM(Model-View-ViewModel) 아키텍처를 사용합니다. 이를 통해 사용자 경험을 향상시키고 복잡한 상태 관리를 용이하게 합니다.
- [ ] Token-based 보안, OAuth2: Level 3과 동일하게 토큰 기반의 보안 메커니즘을 사용합니다.
- [ ] 폴리글랏 언어 + Service Mesh: 다양한 언어를 사용하며, 마이크로서비스 간 통신과 네트워크 기능을 관리하기 위해 Service Mesh를 도입합니다. Service Mesh는 서비스 간의 통신, 로드 밸런싱, 모니터링 등을 추상화하여 관리합니다. 이를 통해 시스템의 확장성과 안정성을 향상시킬 수 있습니다.

인프라 아키텍처 관점 체크포인트

Level 1:

- [ ] 지속적인 빌드와 지속적인 통합 운영을 수행하고 있는가?
- [ ] 개발 및 운영 사이의 원활한 협업을 위한 도구와 프로세스를 도입하였는가?

Level 2:

- [ ] 지속적인 딜리버리와 배포를 수행하고 있는가?
- [ ] 로그를 중앙 집중화하여 모니터링과 분석을 용이하게 하였는가?

Level 3:

- [ ] 컨테이너 사용 (도커)을 적용하였는가?
- [ ] 컨테이너 오케스트레이터 (Kubernetes)를 사용하여 마이크로서비스를 관리하고 스케일링하였는가?
- [ ] 외부 구성 요소 (예: 유레카, 주키퍼)를 활용하여 마이크로서비스 간의 통신과 구성 관리를 개선했는가?

Level 4:

- [ ] 자동 프로비저닝을 갖춘 PaaS(Pass-as-a-Service) 기반 솔루션을 사용하였는가?
- [ ] 인프라 자원의 프로비저닝, 확장 및 관리를 자동화하고 스케일링 용이성을 향상시켰는가?
- [ ] PaaS를 통해 개발팀은 인프라 구성과 관리에 대해 걱정하지 않고 비즈니스 로직에 집중할 수 있는 환경을 제공받았는가?

배포 관점 체크포인트:

Level 1:

- [ ] 설치 스크립트 구동 및 호스트 당 멀티 서비스 인스턴스를 적용하였는가?
- [ ] 설치 스크립트: 서비스 인스턴스를 설치하고 구성하기 위한 자동화된 스크립트를 사용하였는가?
- [ ] 호스트 당 멀티 서비스 인스턴스: 단일 호스트에서 여러 개의 서비스 인스턴스를 실행하여 확장성을 향상시켰는가?

Level 2:

- [ ] VM 당 하나의 서비스 인스턴스, 클라이언트 사이드 로드 밸런싱, 서버 사이드 로드 밸런싱을 적용하였는가?
- [ ] VM 당 하나의 서비스 인스턴스: 가상 머신(VM) 당 하나의 서비스 인스턴스를 실행하여 격리와 확장성을 개선했는가?
- [ ] 클라이언트 사이드 로드 밸런싱: 클라이언트 측에서 요청을 여러 서비스 인스턴스로 분산시키는 로드 밸런싱을 구현하였는가?
- [ ] 서버 사이드 로드 밸런싱: 로드 밸런서를 사용하여 서비스 인스턴스 간에 요청을 분산시키는 로드 밸런싱을 구현하였는가?

Level 3:

- [ ] Immutable 서버, 컨테이너 당 하나의 서비스 인스턴스, blue/green 배포를 포함한 다양한 배포 전략을 구사하였는가?
- [ ] Immutable 서버: 변경 불가능한 서버 이미지를 사용하여 배포와 롤백의 안정성과 일관성을 확보하였는가?
- [ ] 컨테이너 당 하나의 서비스 인스턴스: 컨테이너 당 하나의 서비스 인스턴스를 실행하여 격리와 확장성을 개선했는가?
- [ ] Blue/Green 배포: 새로운 버전의 서비스를 기존 버전과 동시에 배포하고 트래픽을 전환하는 Blue/Green 배포 전략을 구현하였는가?

Level 4:

- [ ] 멀티 클라우드 및 멀티 데이터 센터 지원을 포함한 배포 관점의 성숙도를 갖췄는가?
- [ ] 멀티 클라우드: 여러 클라우드 제공업체를 사용하여 애플리케이션을 배포 및 실행하는 환경을 구축하였는가?
- [ ] 멀티 데이터 센터: 여러 데이터 센터에 애플리케이션을 분산 배포하여 가용성과 복원력을 향상시켰는가?

각 레벨별로 배포 관점에서의 성숙도가 높아짐에 따라 배포 과정이 자동화되고, 가용성과 확장성이 향상됩니다. 레벨이 높아질수록 서비스 인스턴스의 관리와 배포 전략이 더욱 다양해지며, 클라우드 및 데이터 센터의 다중화가 가능해집니다. 이를 통해 애플리케이션의 신뢰성과 유연성을 높일 수 있습니다.

팀 구조와 문화 관점 체크포인트

Level 1:

- [ ] 개발, QA, 릴리즈, 운영이 분리된 하나의 기능 팀을 적용하였는가?
- [ ] 개발, QA, 릴리즈, 운영이 분리된 하나의 기능 팀: 기능별로 분리된 팀이 각자의 역할에 집중하여 작업하고, 개발부터 운영까지 전체 생명주기를 담당하였는가?

역할 및 역량

- [ ] 개발자: 주어진 기능에 대한 개발 및 테스트 수행
- [ ] QA 엔지니어: 품질 보증을 위한 테스트 수행
- [ ] 릴리즈 엔지니어: 배포 및 릴리즈 관련 작업 수행
- [ ] 운영 엔지니어: 서비스 운영 및 유지보수 담당

액티비티와 프로세스

- 개발 프로세스: 요구사항 분석, 개발, 테스트, 릴리즈, 운영 단계로 이루어진 전통적인 워터폴 모델

Level 2:

- [ ] 공유된 서비스 모델로 팀 공동 작업, 내부 소스 공개를 적용하였는가?
- [ ] 공유된 서비스 모델: 여러 팀이 공통적으로 사용하는 서비스 모델을 개발하고 유지보수하여 효율성을 향상시켰는가?
- [ ] 팀 공동 작업: 여러 팀이 협업하여 기능을 개발하고 통합하는 작업을 수행하였는가?
- [ ] 내부 소스 공개: 팀 내에서 개발된 소스 코드와 지식을 공유하고 문서화하여 효율적인 협업을 도모하였는가?

역할 및 역량

- [ ] 개발자: 공동 작업 및 서비스 모델 공유를 위한 협업
- [ ] QA 엔지니어: 공동 작업에 따른 테스트 및 품질 보증 수행
- [ ] 릴리즈 엔지니어: 공동 작업에 따른 배포 및 릴리즈 관련 작업 수행
- [ ] 운영 엔지니어: 공동 작업에 따른 서비스 운영 및 유지보수 담당

액티비티와 프로세스

- 공동 작업 프로세스: 다수의 팀이 함께 작업하여 기능을 개발하고, 테스트 및 운영을 수행하는 공동 작업 프로세스를 따릅니다.
- 내부 소스 공개: 팀 간 지식 공유와 협업을 위해 내부 소스 코드를 공개하고 사용합니다.

Level 3:

- [ ] 서비스별 프로덕트 팀(PO, UI/UX 디자이너, 개발자) Cross Functional한 플랫폼 팀을 적용하였는가?
- [ ] 서비스별 프로덕트 팀: 각 서비스에 대한 전체적인 책임을 갖는 프로덕트 팀을 구성하였는가?

- ☐ PO, UI/UX 디자이너, 개발자: 다양한 역할을 수행하는 멤버들로 구성되어 프로덕트를 개발하고 개선하는데 참여하였는가?
- ☐ Cross Functional한 플랫폼 팀: 다양한 기술과 역할을 갖춘 팀이 플랫폼 관련 업무를 수행하고, 서비스 팀을 지원하였는가?

역할 및 역량

- ☐ 제품 오너 (PO): 서비스별 제품 관리 및 비전 수립
- ☐ UI/UX 디자이너: 사용자 경험과 인터페이스 디자인 담당
- ☐ 개발자: 프로덕트 팀과 협업하여 서비스 개발 및 유지보수 수행
- ☐ Cross Functional한 플랫폼 팀: 서비스 플랫폼 관리 및 개선을 위한 다양한 역할 수행

액티비티와 프로세스

- 프로덕트 팀 프로세스: 서비스별로 제품 팀(Product Team)이 구성되어 제품 관리, 개발, 테스트, 운영 등을 담당하는 프로세스를 따릅니다.
- Cross Functional한 플랫폼 팀: 서비스 플랫폼을 관리하고 개선하기 위해 다양한 역할과 전문성을 갖춘 팀이 협업합니다.

Level 4:

- ☐ 업무 기능별 혹은 도메인별 팀들이 모든 관점에서 책임을 수반하였으며, "네가 구축한 것은 네가 운영합니다"를 적용하였는가?
- ☐ 업무 기능별 혹은 도메인별 팀: 특정 업무 기능이나 도메인을 담당하는 팀이 모든 측면에서 책임을 진다.
- ☐ "네가 구축한 것은 네가 운영합니다": 각 팀이 자체적으로 구축한 서비스나 기능을 직접 운영하고 유지보수하는 책임을 가지며, 자율성을 강조하였는가?

역할 및 역량

- 업무 기능별 혹은 도메인별 팀:
  - 기능 팀: 해당 기능의 개발, 테스트, 배포, 운영 및 유지보수 담당
  - 도메인 팀: 해당 도메인의 비즈니스 요구사항을 이해하고 개발, 운영, 유지보수 수행
- 필요한 역량:
  - 전문성: 자신의 역할에 필요한 기술 및 지식을 보유하고 활용할 수 있는 능력
  - 협업과 커뮤니케이션: 팀원들과의 원활한 협업과 의사소통을 위한 능력
  - 책임감: 자체 운영 및 책임을 수행하고 결과에 대한 책임감을 가지는 자세
  - 문제 해결 능력: 복잡한 문제를 해결하고 시스템을 개선하기 위한 능력
  - 성장 마인드셋: 지속적인 학습과 개선에 대한 열린 자세를 가지고 있는 팀원들

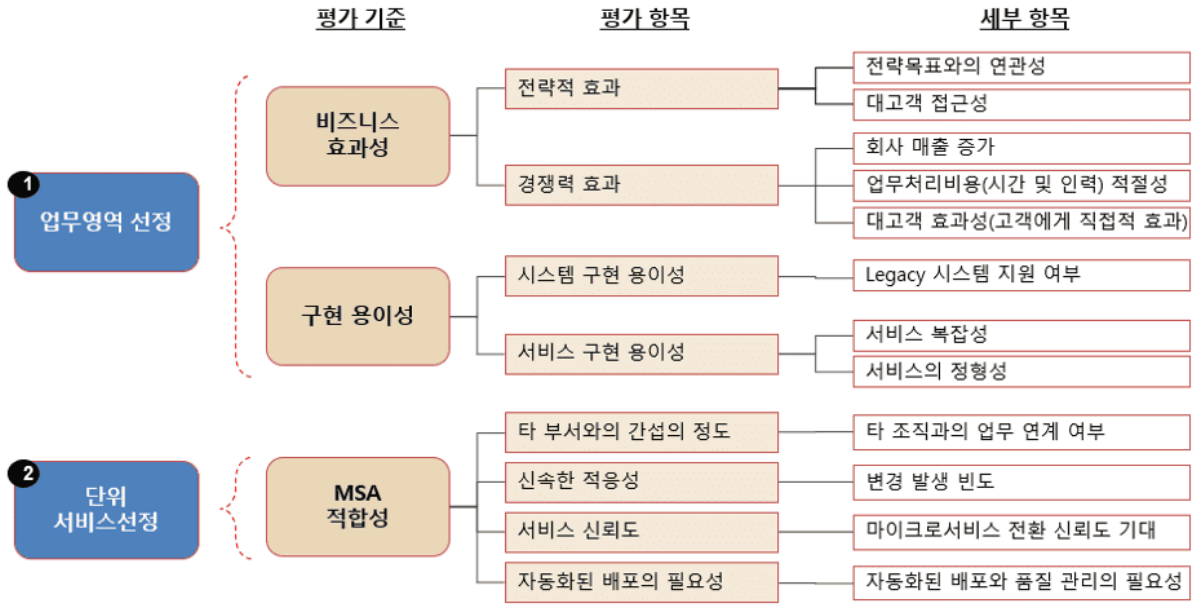
액티비티와 프로세스

- 업무 기능별 혹은 도메인별 팀 프로세스: 각 업무 기능이나 도메인에 따라 팀이 구성되어 모든 관점에서 책임을 수행하는 프로세스를 따릅니다.
- "네가 구축한 것은 네가 운영합니다": 각 팀은 자체 구축한 서비스나 기능에 대해 완전한 책임을 갖고 운영합니다.

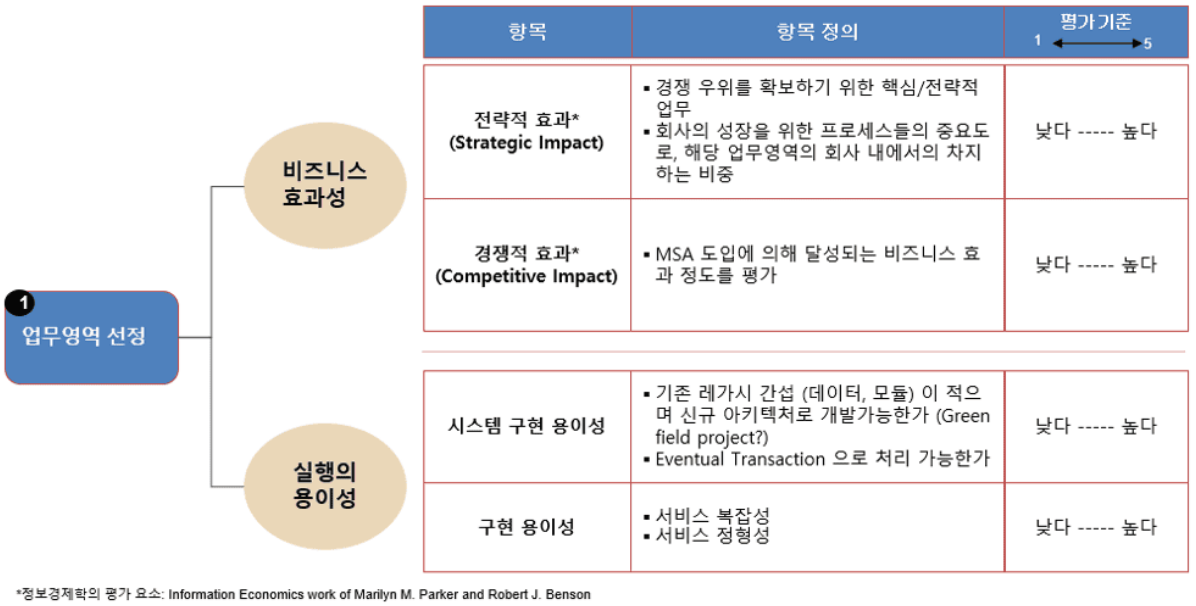
각 레벨별로 팀 구조와 문화, 프로세스 관점에서의 성숙도가 높아짐에 따라 팀 간 협업과 책임 분담이 더욱 강화되며, 자율성과 효율성이 증가합니다. 이를 통해 더욱 빠르고 안정적인 개발 및 운영 프로세스를 구축할 수 있음.

Appendix: MSA 전환 대상 식별 기준

전환 대상 업무의 식별 기준은 업무영역 및 단위 서비스 관점으로 분류할 수 있습니다.

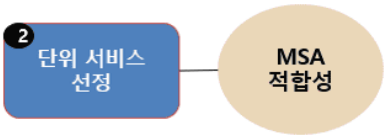


업무영역은 비즈니스 효과성 및 실행의 용이성 측면에서, 전략적으로 경쟁우위에 있으며, ERP와 같은 레거시로부터 상대적으로 간섭이 적고 구현이 쉬운 시스템을 우선 대상으로 선정하여야 합니다.



*정보경제학의 평가 요소: Information Economics work of Marilyn M. Parker and Robert J. Benson

단위 서비스 관점에선 관련조직이 2개 부서 이상이며, 변경빈도가 잦을수록 MSA에 적합하나, 데이터 처리와 트랜잭션이 밀접한 즉, 데이터 인텐시브한 경우에는 모노리식이 더 적합한 모형일 수 있습니다.



항목	항목 정의	평가 기준
배포시 타 부서와의 간섭	▪ 유지보수시 타 부서와 시스템과의 간섭이 심한 경우 → 대상 서비스 추가 선정 기준	모듈 커플링 여부 (대상선정 : 관련조직 2개 부서 이상)
신속한 적응성	▪ 자주 혹은 적어도 정기적으로 변화가 발생하는 서비스여야 함 → 대상 서비스 filtering 기준	변경 빈도 (대상선정 : 1개월 이하)
서비스 신뢰성을 MSA 에 의해 기대	▪ 트랜잭션 처리방식과 정보참조 모형을 MSA 기반으로 할 수 있는가? ▪ 서비스의 신뢰성을 탄력적 배포자동화 및 테스트 자동화에 의해서 기대할 수 있는가? → 대상 서비스 filtering 기준	모놀로씩 아키텍처가 더 적합한 모형인가 (제외 대상 : 데이터-인텐시브한 업무)
서비스의 모니터링 관리가 필요한가	▪ 의사결정, 통제 및 관리를 위해 서비스 진행상황에 대한 실시간 추적, 분석 및 성과 측정이 필요 정도 ▪ 회사 차원에서 전략적으로 관리하고자 하는 서비스의 경우 → 대상 서비스 추가 선정 기준	모니터링 필요 여부 (대상선정 : 모니터링 필요할 경우)

Cloud Native Application 전환 목적에 따른 적용 기술(패턴) 매핑도

관점	Painpoint 개선점	적용 기술과 CNM Level Goal	전환 전략
Availability (방어적-내부적)	최근 마이너한 신규 기능의 추가로 인하여 핵심 기능의 장애가 발생한 적이 있는가 요구되는 uptime 이 가장 높은 업무 영역과 낮은 영역간의 격차가 얼마나 되는가. 단일 데이터베이스를 여러업무가 사용하면서 lock 이 걸리거나 사용자가 대기시간이 오래걸리는 일들이 있는가 사용자 과부하시에 특정 기능에 대기시간이 급격히 늘어나거나 사용자 불만이 존재하는가 시스템 확장에 한계로 작용하는 취약한 지점이 존재하는가 특정 시간대에 업무가 몰려들어 충분한 자원이 있음에도 사용자 이탈 혹은 요청 처리가 불가능한 경유가 발생하여 장	Infra per svc Database per svc Event driven architecture Replica/Auto scale/Infra per svc/Container	
업무 간섭 / 이해의 문제 (내부 프로세스 혁신)	— 업무 수정을 위하여 하나 이상의 부서의 프로세스 합의나 데이터베이스 구조 등의 합의가 필요한가 — 던젼 마스터가 존재하는가 — 던쟌 마스터의 일정에 얼마나 종속적인가 — 타 시스템간 통합을 위하여 해당 시스템이 죽게 되면 내 시스템도 장애가 발생하는가 - 데이터베이스 영향도 때문에 스키마가 영향을 받아 시스템 장애가 생긴작이 있는가. 팀간에 동일한 시스템 공유로 인한 테스트시 야근 등으로 개발자 만족도가 떨어지는가 개발된 제품이 실제로 운영에 반영되는데 여러가지 정서적 기술적 이유로 지체되는 일이 이쓴가 시스템은 분리되어있으나 이들간의 api 가 잘 준수되지 못하여 api spec 연동이슈 및 버전의 혼란이 존재하는가 서버들은 잘 분리되었으나 프론트엔드에서도 팀간	DDD Database Per Svc Infra per svc DevOps Contract Test Microfrontends	

	분리 사용기술 다양화 등의 이슈가 붉어지기 시작하는가		
Agility (공격적-대외적)	업무 배포 시간 — 신규 업무를 개발하여 배포하는데 타 서비스들간의 테스트를 위하여 대기해야하는 시간이 얼마나 걸리는가 (코드 머지, 통합 테스트, 간섭에 의한 오류 수정).	DevOps Pipeline per svc Infra per svc	
다양한 최신기술 수용성 (공격적-대외적)	- 레가시 기술들(버전,언어, os) 발목이 잡혀 신규 기술들을 적용하지 못함 - 업무 영역중 사서 써도 되는 시스템이지만 통합의 이슈로 직접 구현해서 쓰는 시스템 영역은 어떤 것들이 있는가	Container DDD 전략적 설계/Generic Domain (Prioritization)	
성능 (공격적-대외적)	대시보드 혹은 결산 등 데이터 조회에 동원되는 테이블 개수가 막대하여 성능 문제와 join sql 등의 관리가 어려운가 금액 등 중요한 정보를 다루기 때문에 데이터 변경이력을 꼭 관리해야하고 필요시 복구해야 하는가	CQRS Event Sourcing	
관리자동화/ 효율화 (공격적)	관리하는 서버들을 수작업으로 관리하는가 아니면 자동화된 시스템으로 확장 축소 장애시 재시작 등이 이루어지는가 서버들간의 연동에 있어 직접적인 ip 어드레스 등이 관리어려운가 서비스 사용자의 폭주등을 관리할 수 있도록 접속 제한, 공격방어 등이 필요한가 시스템 복구나 테스트용도의 시스템을 최대한 빠르게 재구성할 수 있어야 하는가	DevOps Container Orchestration Service Mesh Infra As A Code / gitOps	
데이터 분석 및 혁신 (매우 공격적)	각 업무 분석의 데이터가 너무 통합되어 도메인별 관리가 어려운가 시계열 데이터의 수집과 분석이 필요한가	Data Mesh EDA	
지속적 혁신 (매우 공격적)	기존 시스템에 대한 개선 지표와 서비스 수준이 관리될 필요가 있는가 기존 사용자가 새로운 시도에 얼마나 반응하는지 지속적인 개선을 위한 실 사용자 피드백이 절실한가	Observability Growth Hack / AB Testing (Service Mesh)	

참고자료:

1. www.msaschool.io
2. https://www.linkedin.com/pulse/microservice-maturity-model-enterprise-adoption-rishi-singh/
3. CMM 4.8 – Open Alliance for Cloud Adoption (oaca-project.org)

참고: 자가진단 시스템: 적합성 평가 / 수준 평가

기능:

- 각 체크포인트의 (가중치 존재) 선택에 따라 역할을 시각화하여 보여줌
- 가고자 하는 목표 레벨에 필요한 역량과 액티비티들을 추출하여 가이드해줌