가치 기술 기반의 인간중심적 서비스를 향한

마이크로서비스아키텍처 기반의 HIS 고도화 전환 사례

2018.11.06

건국대학교병원/대한병원정보협회

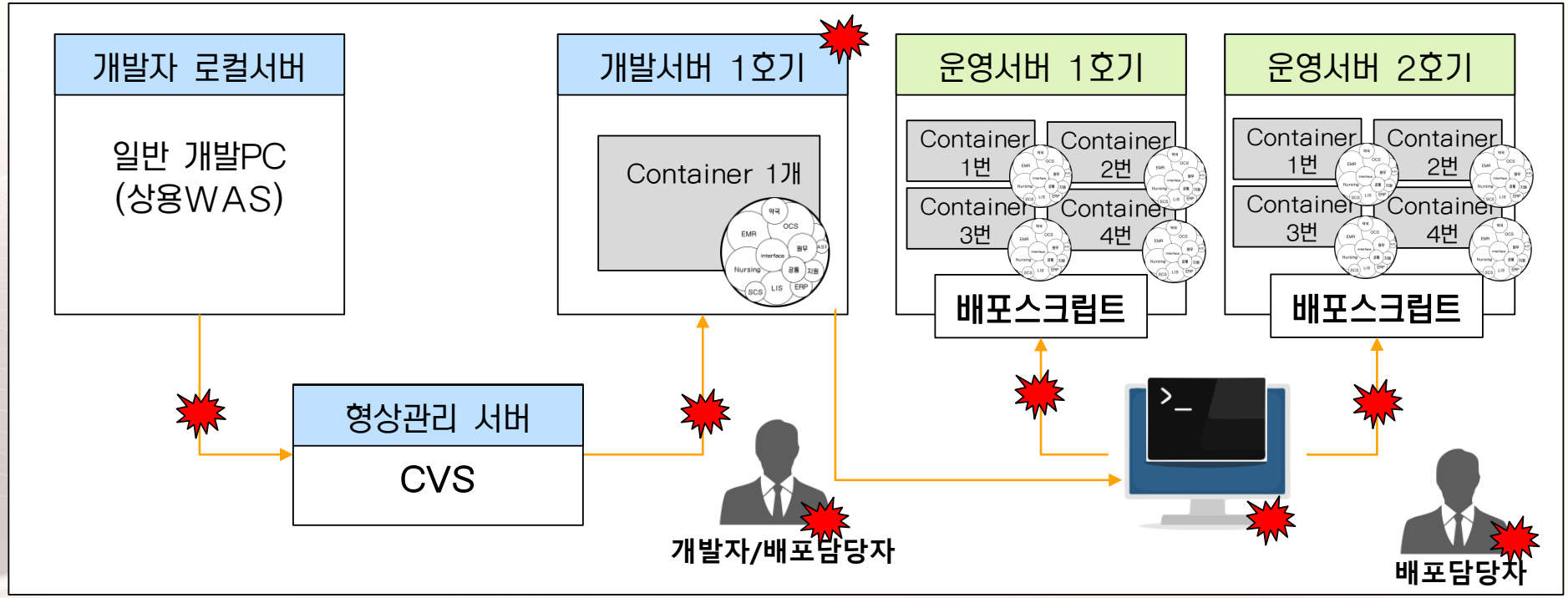
이제관 기술사

newhope@kuh.ac.kr

HIS: Hospital Information System

사업배경: 기존 HIS 운영방식에 따른 이슈

- 우리는 왜 이 사업을 했는가?
  - 이전엔 어떻게 일하고 있었나?



  - 기존 운영방식, 문제가 무엇인가?

컨테이너 느린반영	개발중소스 핫픽스이슈	운영로그 분석불가	배포과정 휴먼에러	타서비스 부하영향	WAS 라이센스관리	운영형상 식별불가

기타 등등

사업배경 : 원하는 것

- **우리는 왜 이 사업을 했는가?**
  - 기존 시스템의 구조를 쉽게 개선하지 못한 이유

▪ CPU JDK 종속버전으로 인한 방해 (Intel Itanium JDK 종속 Build) ✓ 특정기업 Framework 코어소스 부재 및 특정버전 Java, 구성정보의 블랙박스 ✓ 사용 WAS 특정버전에 종속적인 API 강 결합된 구조	밴더종속 코드
▪ 상용 WAS CPU 칩 코어기반 라이선스의 이슈	장인 수준 수작업 숙련도
▪ UNIX 기반의 개발/운영환경으로 관련 서버 프로세스 재정비 부담 (인증점검모듈 등) ▪ 10년 이상의 반복된 행위, 사람에 익숙해진 스크립트 배포 환경	서버 시스템 개발접근 부재
▪ 14년 이상의 오래된 히스토리를 보유한, 문제 많은 형상관리시스템 (CVS) 종속 ▪ 개발자 중심이 아닌 배포 및 시스템 담당자의 막강한 권한행사 ▪ 수동화된 배포환경에 대한 구조 이해자 부재 ▪ DevOps의 이해부족	단일화된 거대SW구조

- **사업을 통해 어떻게 개선하고, 무엇을 하고 싶었나?**

빠른 반영	범용적 환경	밴더종속탈피	최신개발환경	고성능서비스	서비스 분리
오픈소스환경	운영비용절감	자동화 CI/CD	이기종FW구동	Scale Out	Linux/x86

과연 변화가 가능할 것인가, 검증 전환의 필요성

필요성 01

낮은 자바 버전으로 인한
확장성과 호환성 문제

- 웹 애플리케이션 서버의 낮은 **Java 버전(1.4)의 사용 (2002년~2008년/EOL)**
- **기술간 상호기술 연동이 어려움**
- **기술 고립화로** 인한 고도화 확장이 절대적으로 불가
- 인터페이스 모듈들의 업그레이드 가능여부를 보장하지 못함

필요성 02

시스템 버전업그레이드를
위한 **구조진단**

- 낮은 버전의 스펙으로 서버구동환경을 운영하여 관련된 많은 인터페이스 **모듈들의 업그레이드를 보장하지 못함**
- 시스템의 버전 업그레이드를 위해서는 기존시스템에서 사용구조의 정확한 구조분석과 사전 전환검증을 통해 리스크를 감소하기 위한 **전환 구조 진단 단계가 필요**

필요성 03

개방형 표준 기술인
공개 S/W 전환

- **정부의 공개 S/W 육성 및 사용 정책**
- 모바일헬스, 진료정보교류, 정밀의료 등 급변하게 변화하는 의료IT산업의 **기술 환경에 유연한 대응을 IT 신기술 도입**
- 비영리기관인 의료기관이 **벤더 종속적 환경을 탈피**
- 전 세계가 함께 개발하는 오픈 소스 기반의 환경으로 기술기반을 전환

필요성 04

메인 시스템
안정화 계획

- WAS 기반 구조의 성능 개선과 안정화
- **Java 업그레이드** 및 WAS 마이그레이션 **가능여부를 사전에 검증**
- 효율적 전환 체계를 마련

필요성 05

**확장성/호환성/신뢰성/
안정성 확보**

- 기존 Java개발환경의 구조진단 및 전수전환을 통해 변화에 따른 **고 신뢰성 및 안정성 확보**

기술중심의 사업 성공을 위한 사업접근방안

- 사업의 분할과 정복 (Divide and Conquer)

1차 사업 (16.12~17.3)	결과 검증 버퍼	2차 사업 (17.9~17.12)
U2L 전환검증 사업 (개발전환 가능성검증)	추가 해결과제 및 방안정리 HW/SW 운영환경 모의설계 고도화 방안 모의구상 설계	WAS 아키텍처 변경사업 (HW변경/운영환경 적용사업)

- 사업 발주자 공동참여 & 사업 직접리딩

발주처 (건국대학교병원)		사업 수행사
사업 기획 설계 / 아키텍처 구조 설계 지원 기존시스템 연동검증 / 품질검증 Matrix 점검 전환결과 전수 테스트 지원 / AS-IS 분석 지원 데일리 스크럼미팅 / 부분개발지원 / PMO	 상호협력	적정기술검토 / 전환자동 툴 개발 / TO-BE 구현 개발환경 설계 / 품질지표 달성 점검 연동환경 개발 / 인터페이스 Mismatching 해결 데일리 스크럼미팅 / 소스전환 / 기술지원

1단계 U2L 사업 : 개방형 표준 기술로의 인프라 환경 전환

AS-IS

TO-BE

AS-IS	TO-BE
JEUS 4.2	JBoss EAP 6/7
J2EE 1.3	Java EE 1.7
JDK 1.4	JDK 1.8
UNIX : HP-UX(11.23)	Red Hat Linux(7.3)
물리서버	RHEV 4.0 가상화
SPARC/PA-RISC/Power	표준 X86

적용방안

- UNIX TO LINUX (U2L 전환 검증)
- 인프라 (서버/OS/미들웨어) 를 단일 벤더 지원
- 각 부분별로 **개방형 표준 환경**으로 전환
- **오픈 소스 프레임워크**를 통한 개발
- 라이선스 비용 없기 때문에 초기 도입 비용 절감

기대효과

- 특정 서버 벤더와 기술에 얽매이지 않는 환경
- Cost Performance 가 좋은 인프라를 선택
- 서비스 품질이 높은 개발/운영 업체 선택
- 운영 효율성 향상
- 저비용 고효율 시스템 구축

1단계 사업 : U2L 전환 타겟 KIS 시스템 애플리케이션 현황

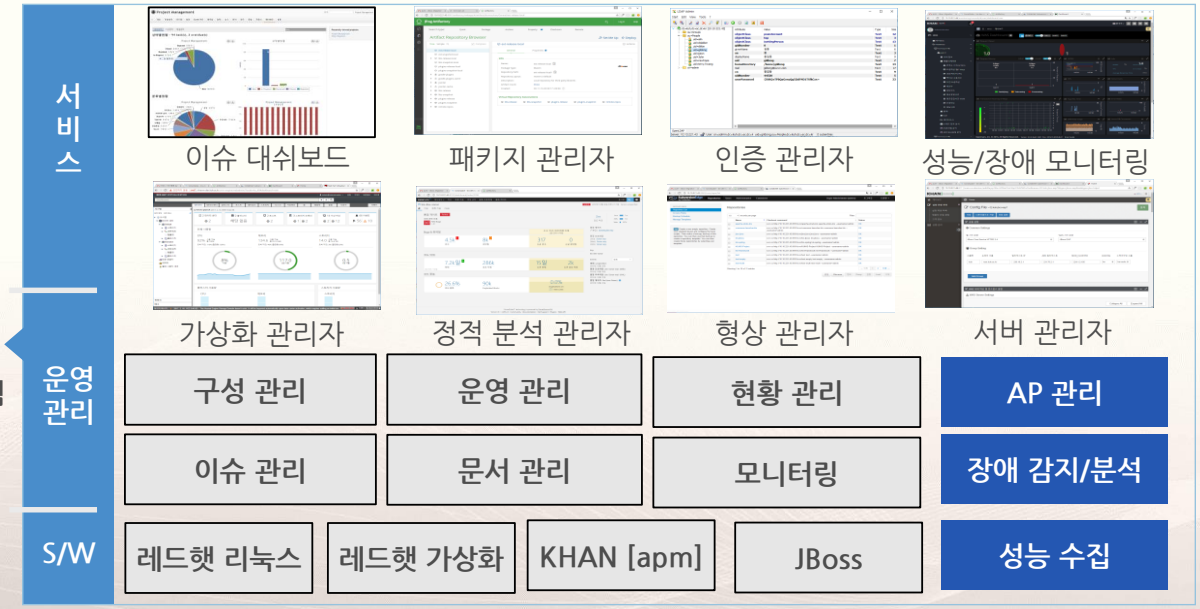
- 건국대학교병원 업무 시스템의 전체 애플리케이션 사이즈가 상당히 큰 대규모

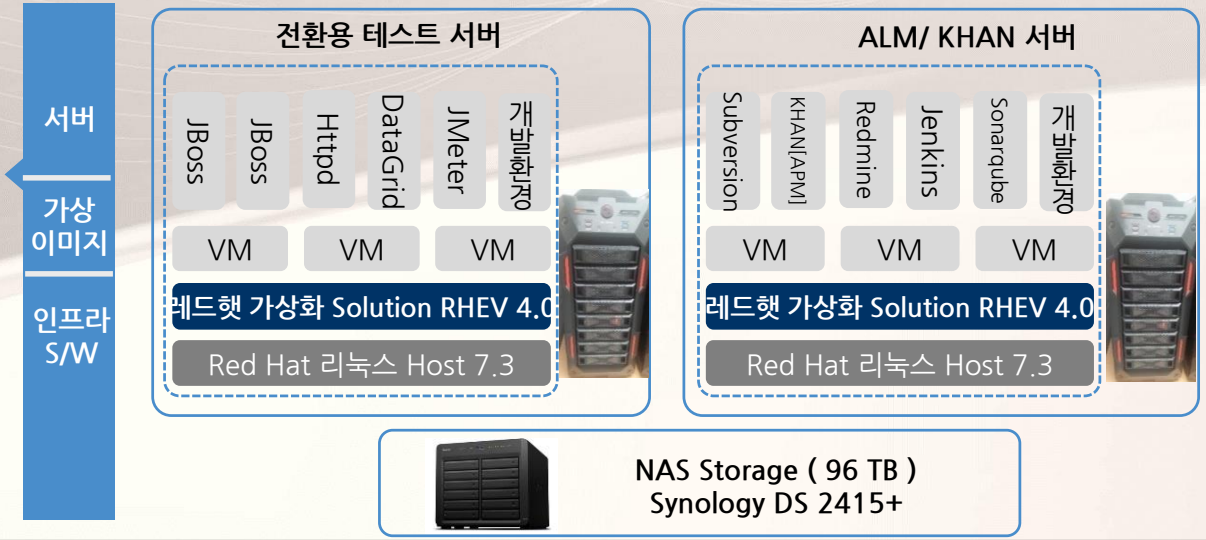
구분	현황	분석

소스 및 설정파일	• **화면 9,000본**
	• EJB JAR 패키지 190개
	  • **691개**의 Stateless **Session EJB**
	• WEB : 44개
	• 배치 : 17개
	• XFM 화면 Layout 4,488개
	• JSP 6,464개
	• 애플리케이션 소스 크기 : 2.1GB
	• UNIX Shell 프로그램 : 82개
	• **전체 파일 수 : 약 62, 000 개**
	  • java/jsp – 17,000 개
	  • image – 16,000 개
	  • js/css/ui – 13,000 개
	  • xml – 6,000 개
	  • 기타 – 10,000 개
		• 엄청난 큰 프로젝트 사이즈
		• 의존도 관계 및 **복잡도가 높아 관리 어려움**

개발환경	• 일반 에디터가 주를 이루고, 일부 Eclipse, Intellij 사용 • 소스가 너무 커서 **통합개발환경 사용 시 어려움**	• 표준화 구조가 아니다 보니 통합개발환경 설정이 어려워 **일반 에디터를 주로 활용** • 개발 생산성이 낮음

1단계 U2L사업 : 마이그레이션 시스템 구성 아키텍처





OpenSource SW 도입검증 제품

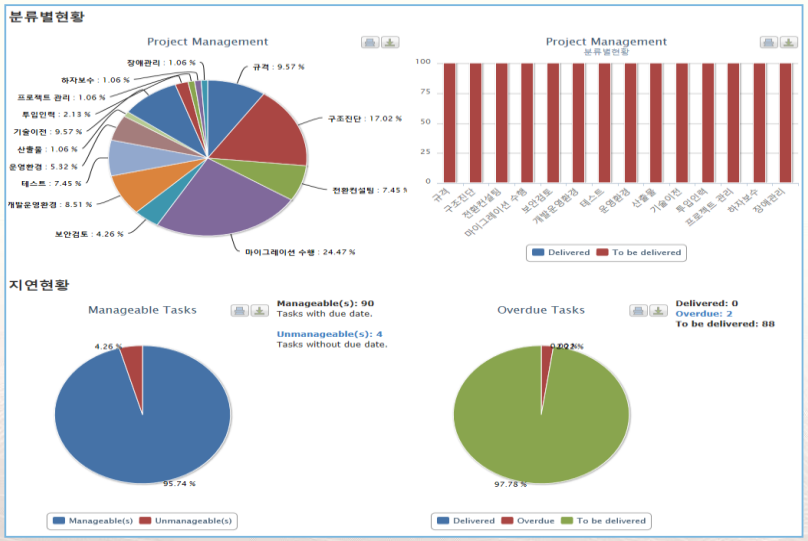
- (Opennaru) KHAN [apm] (어플리케이션 모니터링)
- (Opennaru) KHAN [session manager]
- OpenSource Redmine : 이슈관리
- OpenSource Jenkins : 배포관리
- OpenSource Subversion : 형상관리
- (Red Hat) JBoss EAP 6.4 / 7.0 : 오픈WAS
- (Red Hat) Enterprise Linux 7.3
- (Red Hat) Enterprise Virtualization 4.0
- (Red Hat) DataGrid (메모리 캐쉬디스크)

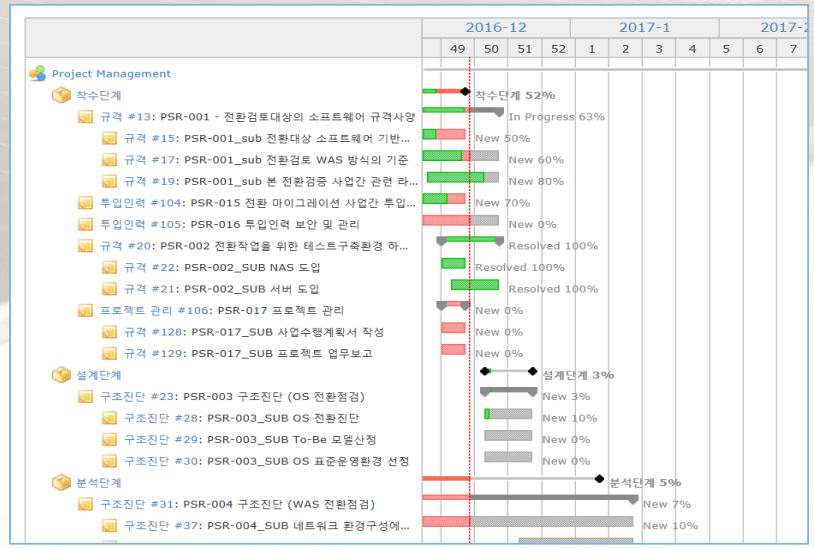
표준 X86 구동 검증용 장비 (PC급, 용산 조립)

- 제온 CPU 2.6GHz 데카(10)코어이상 * 2개
- Cooler 1CPU (2EA)
- 메모리 128GB (DDR4 32GB * 4)
- SSD 512GB (삼성전자 850 PRO, 2EA)
- HDD 4TB (7200RPM 2개이상 장착 2EA)
- PowerSupply 1000W 이상 정격
- ATI Video Card
- 4K-UHD 모니터 27인치 구성(LG)
- 호환 키보드/마우스(1000DPI 이상)
- PC용 VM환경 Linux 구동 최적 스팩으로 구성

- 2.4 GHz로 실행하는 쿼드 코어 CPU
- 2GB의 DDR3 RAM(6GB까지 확장 가능)
- NAS 전용 하드 96 TB (RAID 1 +0, 8TB * 12 EA)

U2L사업간 오픈소스 ALM 기반 온라인 프로젝트 관리



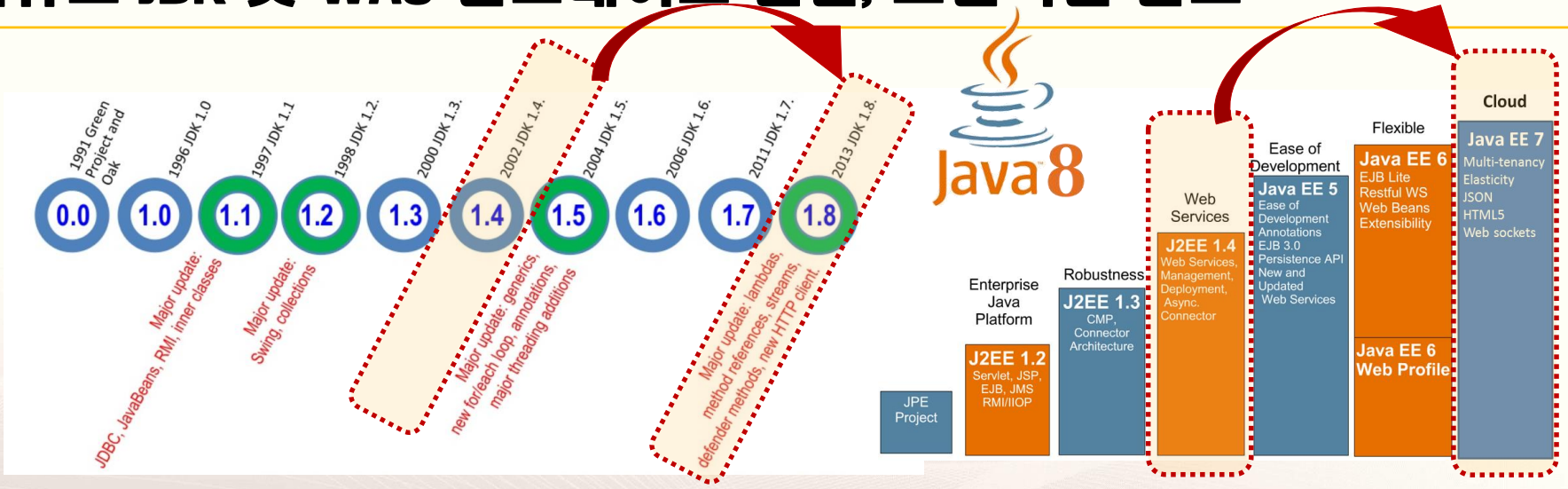


프로젝트 목표달성	ALM 기반 프로젝트 관리
• 고품질 시스템 구축 • 요구사항을 반영한 시스템 구축 • 사용자 중심 시스템 구축 • 프로젝트 성과 목표 달성	• **ALM 기술에 대한 내재화** • **효율적인 프로젝트 관리** • 프로젝트 성과 목표 달성

프로젝트 관리 투명화 | 이슈와 위험관리 강화 | 품질관리 강화 | 체계적인 변화관리

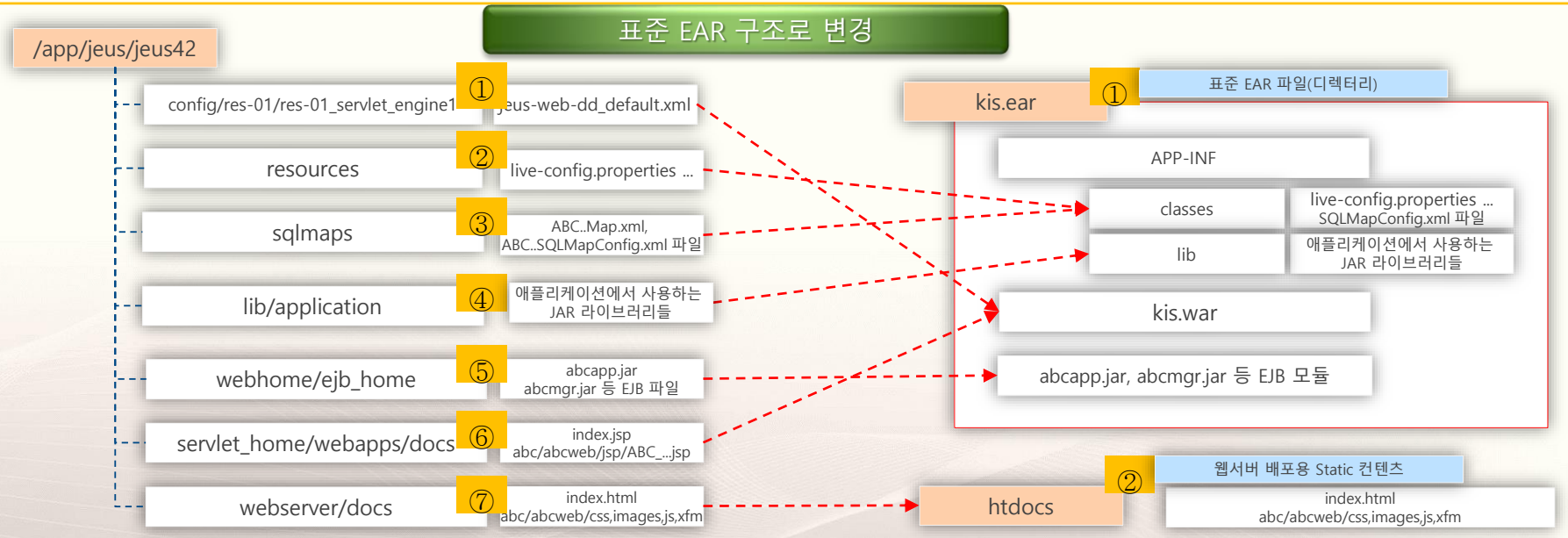
ALM : Application Life-cycle Management / 개발 전 주기에 대한 가시화

대규모 JDK 및 WAS 업그레이드 전환, 호환작업 완료



항목	AS-IS	TO-BE
생산성	• **낮음** • 활용도가 높은 **최신 오픈소스 모듈 사용 불가** – 대부분 JDK 1.6 이상 지원	• **높음** • 대부분의 **최신 오픈소스 활용 가능** • 람다와 같은 최신 문법으로 코딩량 감소 등 **생산성 증가**
성능 및 보안	• **낮음**	• **높음**
최신기술	• **활용 불가**	• 전자정부 프레임워크, Spring Framework 등 **활용 가능**
최신	• JDK 1.4 의 치명적인 문제 **EOL**	• **최신 JDK 1.8 환경**으로 전환
최신 소프트웨어 지원	• JEUS 최신 버전 **지원 불가** • JEUS 4.x EOL • 제니퍼, KHAN 등 **최신 솔루션 지원 불가**	• 최신 JBoss EAP 7.0(6.4) 버전으로 전환 • 향후 **기술지원 가능한 최신 버전** • **최신 솔루션 지원 가능**
보안	• **SSL 도입 불가**	• **SSL 환경 적용 가능**

특정 밴더종속 구조 탈피, WAS 표준 구조의 배포 모듈화



항목	AS-IS	TO-BE
구조	• 비 표준	• 표준
동작 서버	• **JEUS 4.2 만 가능한 배포 구조**	• **표준을 지원하는 모든 WAS에서 동작 가능**
관리 인력	• **JEUS 전문가**와 구성에 참여 했던 사람만 알 수 있는 구조 • 어디에도 없는 구조	• **표준만 알면** 어떤 엔지니어나 중급 개발자면 **누구나 쉽게 가능**
교육비용	• 증가	• 교육비용 거의 없음
적응기간	• 수개월	• **표준 기술만 익히면 즉시 적응**
개발 환경	• **셋팅이 어려움**(1~2주 소요) • IDE 툴의 기능을 100% 사용하기 어려움 • IDE 툴 따로, WAS 구동 따로, • 소스 수정 따로, 메타 설정 파일 수정 따로 • 여러 툴 및 에디터를 사용해야 개발 가능	• IDE툴을 100% 활용 가능 – IDE 툴 안에서 개발 및 WAS 컨트롤까지 모두 가능 • 개발자가 모든 설정을 관리 • 모든 설정이 형상관리 대상이 됨 • WAS의 설정은 데이터소스 밖에 없음

마이크로서비스아키텍처 기반의 HIS 고도화 전환 사례, 건국대학교병원 이제관 기술사

밴더종속 소스제거, 오픈소스 라이브러리 전환 업그레이드

번호	라이브러리 이름	기존 버전	업그레이드 버전
1	live-core (수정)	1.01	1.02
2	live-ejb (수정)	1.01	1.02
3	live-web (수정)	1.01	1.02
4	service	1.01	1.02
5	medical	1.01	1.02
6	struts	1.2.7	1.3.10
7	**commons-launcher**	**1.1**	**1.1.kis**
8	commons-logging	1.0.4	1.1.1
9	commons-validator	1.1.4	1.3.1
10	commons-beanutils	1.7.0	1.8.0
11	commons-digester	1.6	1.8
12	ibatis	2.0.9	2.3.4
13	**log4j**	**1.2.9**	**logback-1.1.8**
14	httpclient	추가	4.5.2

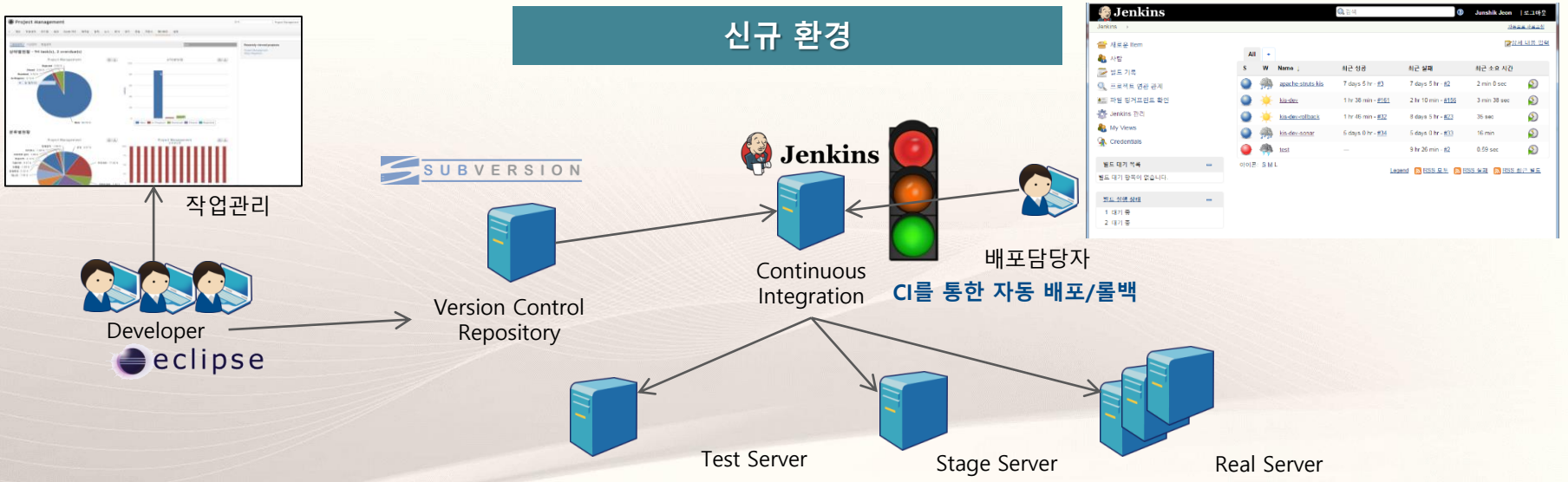
구분	갯수
추가	2
수정	12
삭제	1

항목	기존 Live Framework (특정 SI회사 개발)	신규 OpenSource Lib (공개모듈)
변경내용	• Live Framework(core, service, medical, batch, web) 모듈을 **Java 1.8 환경에 맞게 튜닝 및 수정 배포** • 개발 중단 된 commons-launcher를 **JBoss, Java 1.8 환경에서 동작하도록 수정하여 배포** • Java 1.4 이상에서 사용 할 수 없었던 **오픈소스 유틸리티 라이브러리를 최신 버전으로 업그레이드** • 라이브러리에 따라서 **업그레이드, 추가, 폐기, 교체**	• 개발 공통으로 사용되는 유틸리티 최신 버전으로 업그레이드를 통한 **기능 강화 및 성능 및 보안성 강화** • DB 통신을 위한 iBatis 최신 버전으로 **업그레이드하여 성능 및 보안성 강화, 로깅 기능 추가** – 호출된 SQL 로그에 대한 정렬 및 파라미터 자동 맵핑되어 출력 • log4j → **logback 변경으로 성능 향상**

오픈소스 ALM 기반의 신규 개발/빌드 환경으로의 유연성 검증

Continuous Integration의 주요 특징

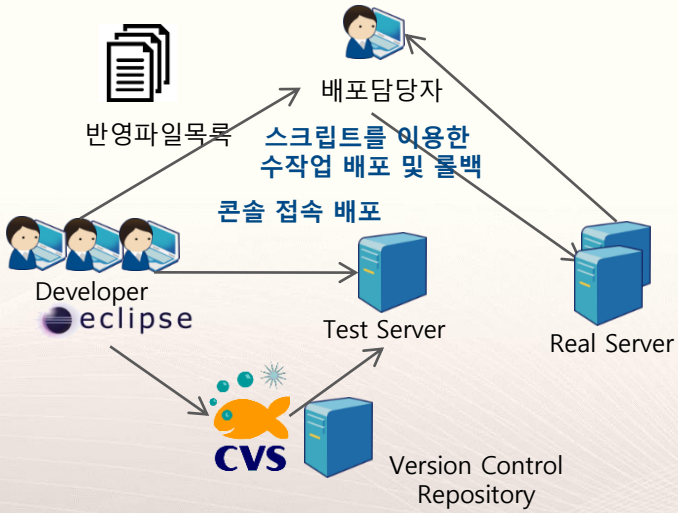
- 소스코드 일관성 유지
- 자동 빌드(스케줄링에 의한 시간 간격이나 소스 커밋이 되면 빌드)
- 자동화된 테스팅, 일일 체크아웃 및 빌드
- 소스 코드에 대한 주기적인 정적 분석 작업도 가능
- 자동 배포 및 롤백, JBoss 인스턴스 순차 재 기동
- GUI를 통한 관리
- 다양한 플러그인 활용



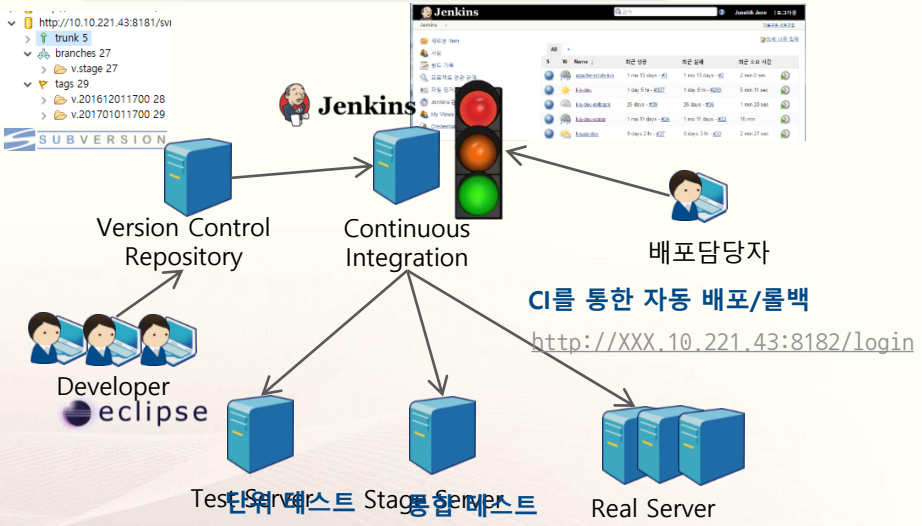
항목	AS-IS	TO-BE
장점	• 없음	• **프로세스 전반적인 부분에 대한 가시화** • **요구 단계 → 운영 반영까지 투명하게 관리** • **변경 사항에 대한 세부적인 내용까지 추적 가능** : 누가 요청을 해서, 누가 개발을 하고, 언제, 어느 부분을, 어떻게 수정, 언제 개발에 반영, 언제 운영에 반영
단점	• 투명하지 않은 프로세스 • **수정 내역에 대한 추적이 어려움** • **보이지 않으면 관리하기 힘들다**	• 데이터 오류 : 개별 일감에 대한 난이도와는 별개로 일감 수만 늘려 실적을 올리려는 상황 발생 • 신기술 도입에 대한 부담

U2L 전환간, 모의 운영 반영 프로세스 개선

기존 배포 프로세스



CI를 통한 배포 프로세스



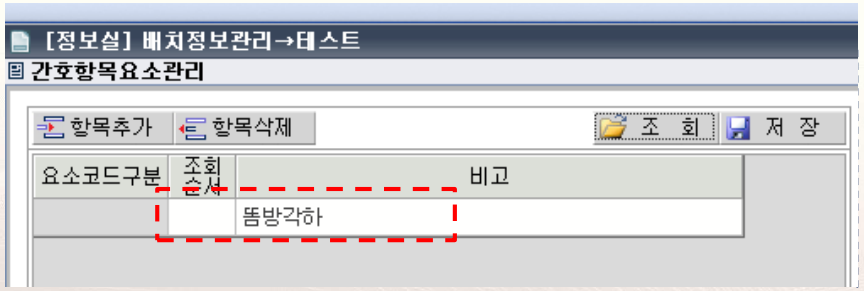
항목	AS-IS	TO-BE
자동화 여부	• 전담 인력에 의한 **수작업**	• CI도구(오픈소스)를 이용한 **자동 빌드 및 자동 배포**
패키징	• **배포 담당자**가 **수작업 취합**하여 정리	• 형상관리 기준으로 **자동 취합**
배포 시간	• **약 20분 (1대 서버 기준)**	• **약 7분 (1차 작업) → 개선대상!!!! (목표 운영 3분이내)**
롤백	• **일부 파일 및 반 자동화** 된 롤백	• **자동화**(배포 시 해당 파일 자동 저장)
롤백 시간	• **약 20분**(배포 시간과 동일)	• 패키징 시간이 없으므로 **약 3분**
배포 인원	• **배포 담당자, 시스템 관리자**	• 배포 담당자가 웹기반 UI에서 **버튼 클릭 한번**으로 가능
휴먼 장애	• 사람이 하는 일이라 파일 누락 등 **실수 가능**	• **없음**
소스 및 설정 파일	• 서버와 **형상 관리와 설정파일 다중 관리**	• **형상관리**로만 관리
배포검증	• **자동화 불가**	• 테스트 케이스, 정적분석(**시큐어 코딩**), 코드 커버리지 등 기술 적용 가능
현행	• **자동으로 빌드 후 클러스터링된 환경으로 순차적으로 자동 배포하여 반영(테스트 담당자는 재시작을 인지하지 못함)**	

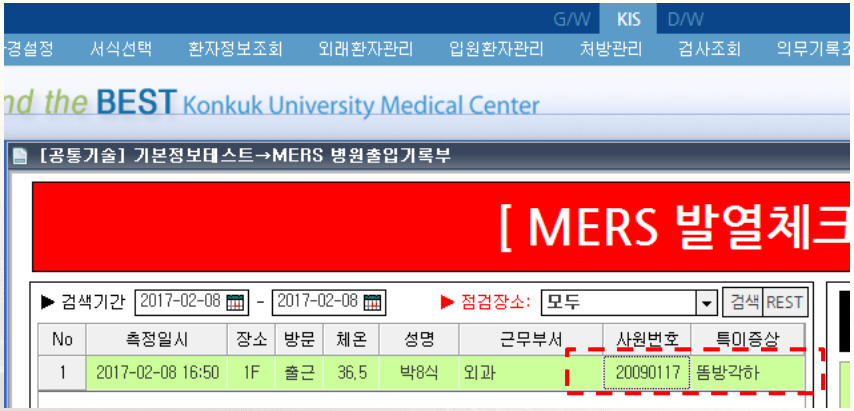
U2L 전환간, 인코딩 개선, 한글 문자 입력 범위 확대

외국인 환자 이름 한글 문자 입력 가능

- WAS 서버 시스템 MS949를 지원하도록 환경 설정
- JSP 파일/ Servlet필터이 MS949 확장 문자셋을 지원하도록 설정

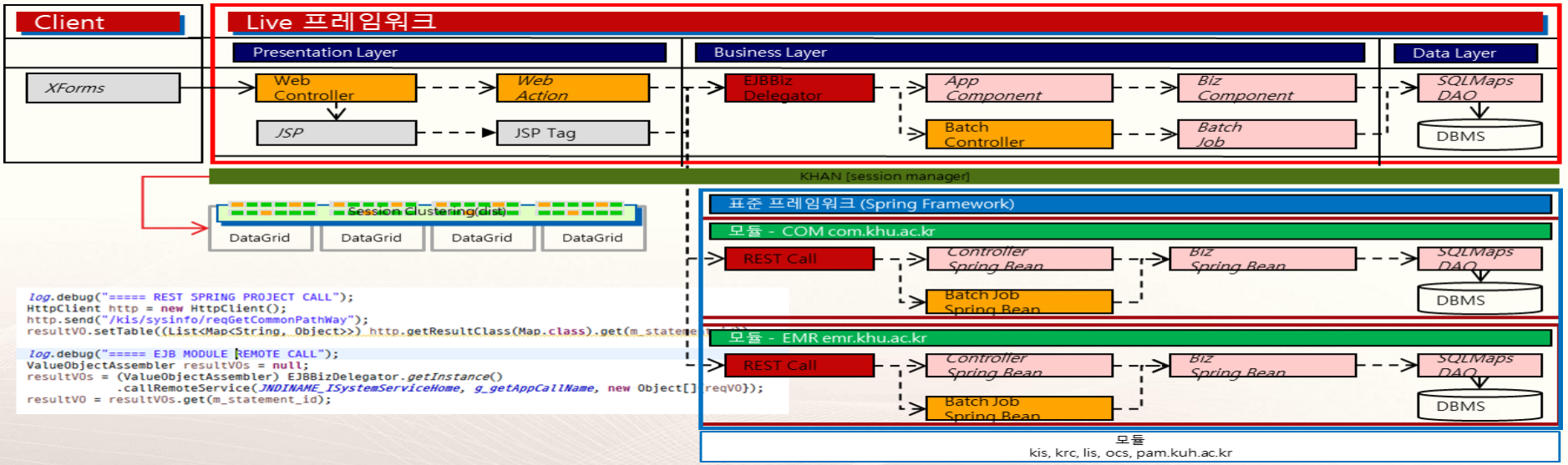
- 데이터셋을 생성하는 JSP파일의 MS949 지원하도록 변경
- 테스트 페이지와 MERS 체크 페이지를 통해 '똠' 문자가 입력/표현됨을 확인





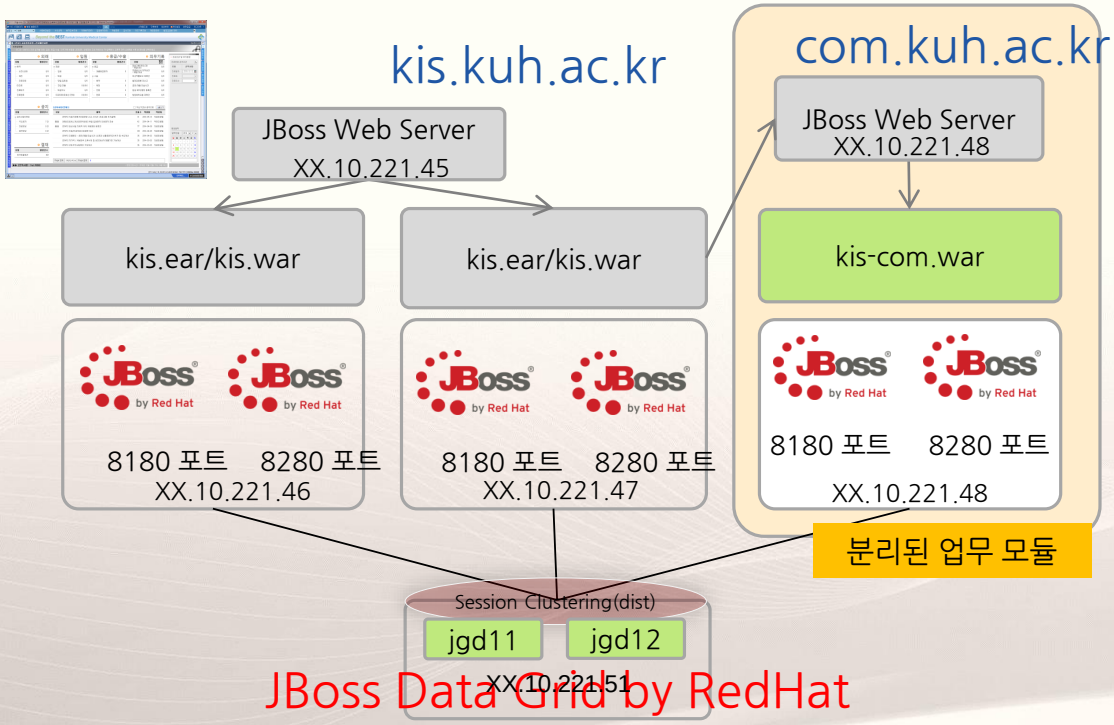
항목	AS-IS	TO-BE
인코딩	• EUC-KR 사용	• MS949 로 전환
차이	• 외국인 환자 이름 한글입력등 외래어 표기 불가 • 똠, 뷀 등 표기 불가	• **외국인 환자 이름 한글입력등 외래어 표기 가능** • 똠, 뷀 등 표기 가능
단계별 설정	• OS : ko_KR.eucKR • WAS JEUS : EUC-KR • XML, JSP : EUC-KR • DB : MS949	• OS : ko_KR.eucKR • WAS JBoss EAP : MS949 • XML, JSP : MS949 • DB : MS949
테스트 상황		• 레포팅 ActiveX 상황 테스트, 프린터 – OK • 외래어 등 인코딩과 관련된 이슈들이 해결되었음 • 데이터베이스 저장, 화면 출력, 프린트, 레포팅 등에 대해서 테스트 완료

주요 진행 사항 – 전자정부프레임워크 연동 검증 완료



항목	BigOne Project : Servlet → (remote)EJB -> DAO	Module Project : Servlet → (rest)Spring → DAO
장점	• **단일 거대한 어플리케이션 구조**(소스만 60,000개)	• **업무별로 분리하여 프로젝트의 경량화 가능** • **IDE 툴 사용** 시에도 가벼워서 능률 향상 • 업무별 의존도 감소 – 장애가 발생 할 경우 해당 업무만 장애 발생 • 배포 속도 향상 • 국가가 공인한 전자정부 프레임워크 기반으로 개발 가능 • 최신 기술 및 기능을 활용하고, 추가 기술 도입이 쉬움 • 기존 EJB 개발보다 복잡도가 낮아 개발 생산성 향상 – 최근에 EJB 를 다뤄본 개발자도 거의 없음 • EJB Remote Call 보다 성능 향상 • 기존 비즈니스 로직은 같으므로, COPY & PASTE 수준으로 개선 가능 – 또는 신규 개발 요건에 대해서만 순차적으로 적용
단점	• 프로젝트가 무거움(소스만 약 40,000개) • IDE 툴을 사용하기가 버거움(엄청 느림) • 복잡도가 높아 생산성 낮음	• 업무별 프로젝트를 관리
비고	• 아키텍처 그림	• 마이크로 서비스로 가기 위한 첫 걸음

오픈소스 메모리 캐쉬기반의 Data Grid 활용한 사용자 세션 처리



Data Grid 방식의 장점

- WAS 장애 발생시 세션 데이터 보호
- WAS의 세션을 별도 보관하여 WAS 분할 가능 구조 설계
- 세션 정보를 분산하여 세션 저장영역에 대한 스케일 아웃
- 대용량으로 Http Session 데이터를 사용하는 웹 애플리케이션
- High Volume 웹 애플리케이션 혹은 부적절하게 설계된 Http Session을 사용하는 웹 애플리케이션에 적용
- Heap 사이즈를 크게 하는 것이 아니라 JVM 의 프로세스 수를 늘리는 것으로 Full GC 의 악영향을 회피하면서 시스템을 확장
  - 유휴 H/W 메모리 활용
  - 이 기종의 AP 서버 간의 세션 정보 공유
  - 웹 애플리케이션간 세션 정보 공유 (none SSO)

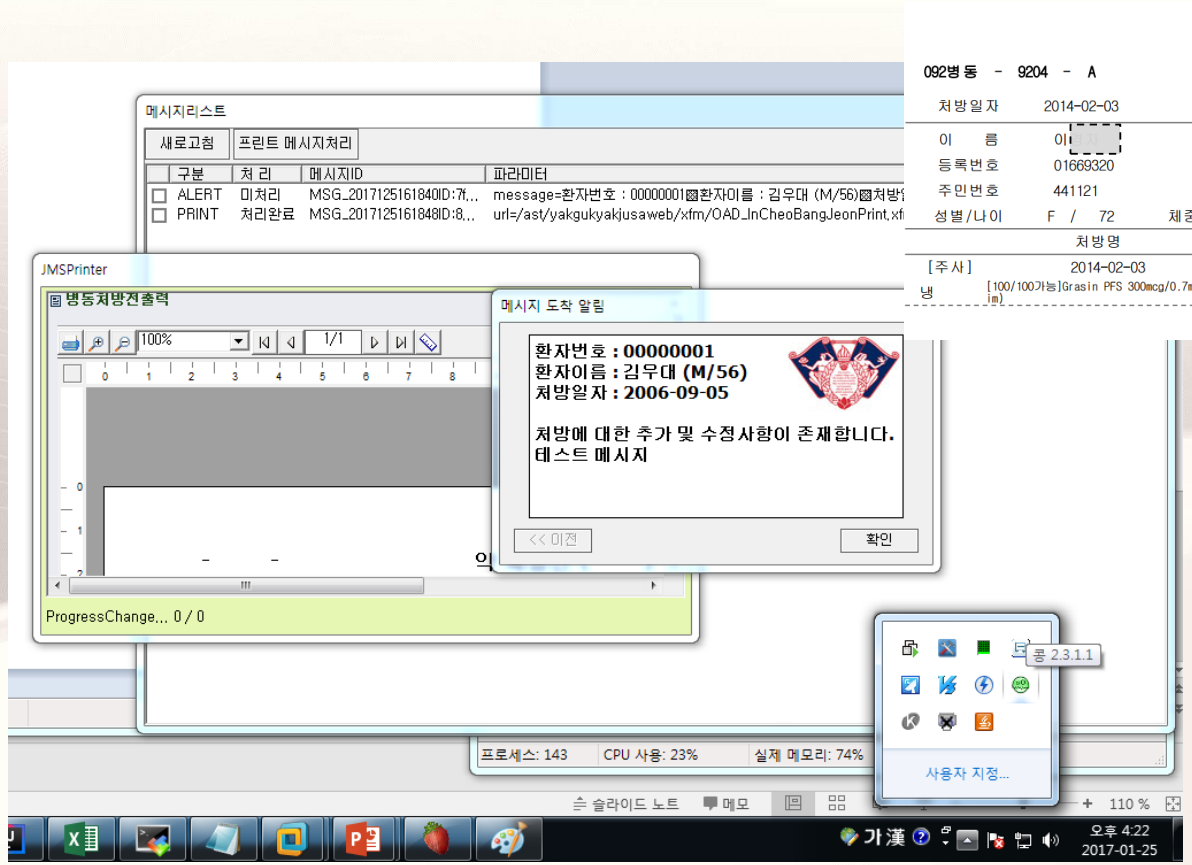
항목	AS-IS	TO-BE
장점	• WAS 기능만으로 가능	• 많은 서버로 운영할 경우 세션 공유(동기화)에 대한 오버헤드가 적음 • **업무별 모듈화 및 마이크로 서비스**시 SSO 연동 가능 • kis.kuh.ac.kr • emr.kuh.ac.kr • com.kuh.ac.kr • pam.kuk.ac.kr 와 같이 업무별로 서비스가 나뉘어도 **세션 공유(SSO) 가능**
단점	• SSO구현 어려움	• 비용 발생

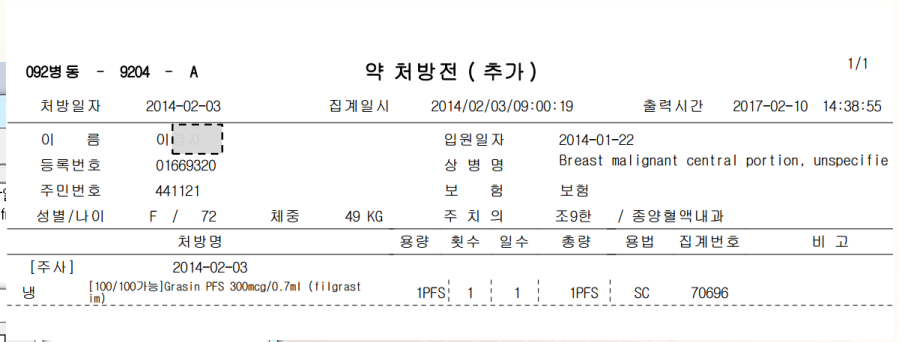
메세지큐 기반 WAS독립형 JMS & Batch 전환: 약국 처방전 / 원격라벨 프린트 업무

Windows C++ 클라이언트 작업

- JBoss JMS Queue를 이중화하여 설치함
- Windows 애플리케이션에서 JEUS 사용 부분을 JBoss로 변경
- 기존 클라이언트의 JRE 환경 변경 : Java 1.4 → Java 1.8

- 실제로 대부분의 JMS관련 로직은 Java로 구현되어 있음
- exe4j를 사용하여 Java 애플리케이션을 윈도우 exe 파일로 변환함
- 배치를 이용한 약국 처방전과 라벨 프린트 작업 테스트 완료



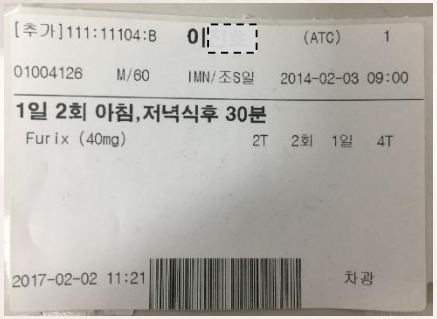

092병동 - 9204 - A

약 처방전 (추가)

1/1

처방일자	2014-02-03	집계일시	2014/02/03/09:00:19	출력시간	2017-02-10 14:38:55
이 름	이		입원일자	2014-01-22	
등록번호	01669320		상 병 명	Breast malignant central portion, unspecifie	
주민번호	441121		보 험	보험	
성별/나이	F / 72	체중 49 KG	주 치 의	조9한 / 종양혈액내과	

처방명	용량	횟수	일수	총량	용법	집계번호	비 고
[주사] 2014-02-03							
냉 [100/100가능]Grasin PFS 300mcg/0.7ml (filgrast im)	1PFS	1	1	1PFS	SC	70696	


[추가]111:11104:B 이 (ATC) 1

01004126 M/60 IMN/조S일 2014-02-03 09:00

1일 2회 아침,저녁식후 30분

Furix (40mg) 2T 2회 1일 4T

2017-02-02 11:21 차광

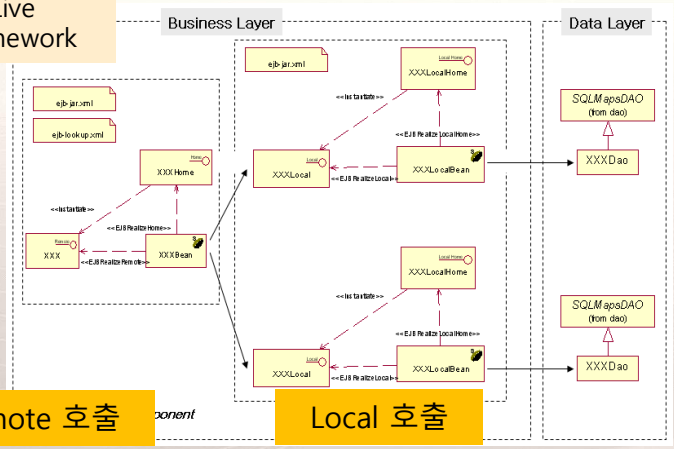
마이크로서비스아키텍처 기반의 HIS 고도화 전환 사례, 건국대학교병원 이제관 기술사

기존 EJB 컴포넌트 재활용, 호출변경으로 아키텍처 품질 성능 개선

① 코드변경없이 EJB Remote Call을 Local Call로 변경

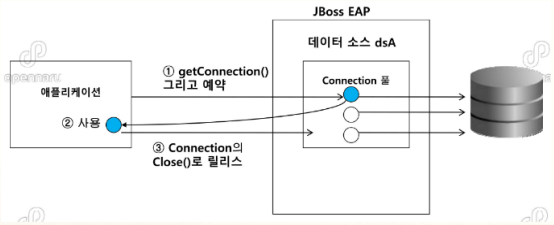
- JBoss EAP의 기능을 통해 Remote 호출 코드가 Local 호출(Call-By-Reference)로 실행되도록 설정함(표준 EAR 구조로 변경되었기 때문에 가능함)

Live Framework



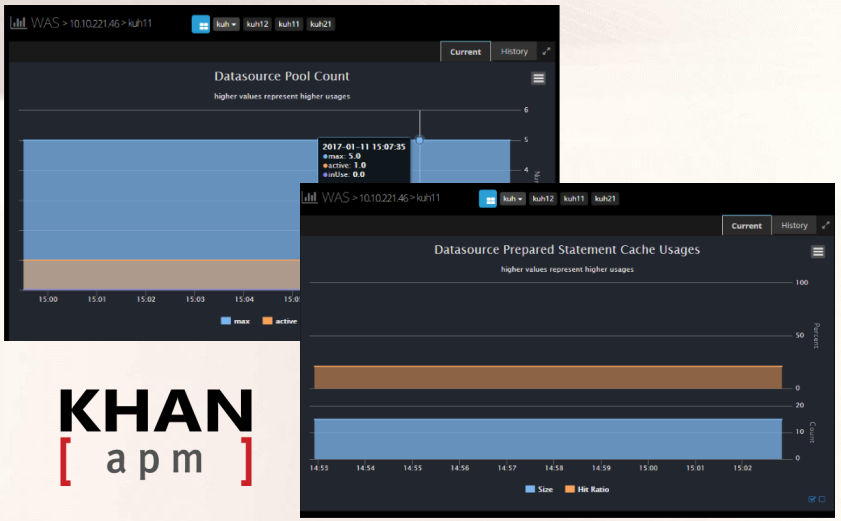
호출 방법	속도
Local EJB Call	1 (기준)
Remote EJB(같은 EAR)	2.7배 느림
Remote EJB(다른 EAR)	27배 느림

② 데이터베이스 연결 풀 Tuning



- DB 연결 풀 체크 로직 개선
- Prepared Statement Cache를 적용(200개)
- Oracle JDBC ResultSet Prefetch를 튜닝 : 10 → 100개

DB 연결풀 및 Prepared Statement Cache 모니터링

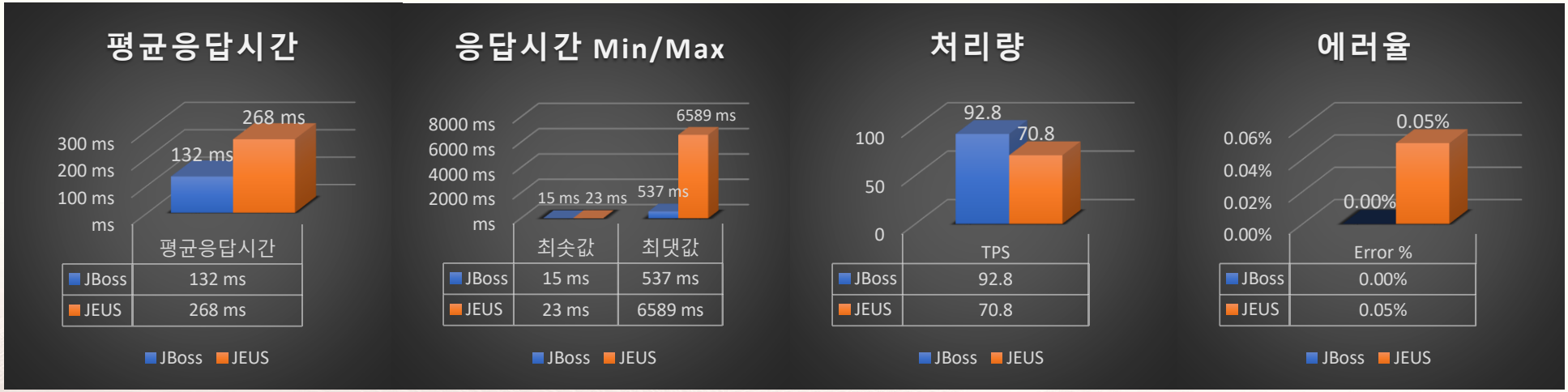


KHAN [apm]

③ JVM 메모리 튜닝, OS 튜닝, JBoss Web/WAS 튜닝

- KHAN [apm] Provisioning의 기능의 자동 튜닝 적용
- 최적화된 OS 커널, JVM 메모리, Web/WAS 튜닝

기존 WAS (슈퍼돔1) VS 신규 WAS (x86 PC급) 아키텍처 성능 비교



- 단순 부하테스트
- Bottleneck 구간에 대한 튜닝이 필요할 수 있음
- PC서버의 경우 가상화 환경임

부하 생성

기존 KIS App
JEUS 4.2
HP-UX 11.23
Java 1.4
고사양 UNIX 서버

WAS 계층

신규 KIS App
JBoss 7.0
RHEL 7.3/RHEV 4.0
Java 1.8
PC급 Linux 서버

DB 계층

ISMS를 위한 WAS 아키텍처, 보안 요구 사항 조치 내역

항목	ISMS 요구사항	조치 사항
웹 취약성	• 웹 취약성 조치	• KHAN [apm]을 통한 • WEB/WAS 설치 시에 웹 취약성 보완
시큐어 코딩	• 시큐어 코딩 정적 분석 적용하여 관리할 것을 요구	• SonarQube를 이용한 시큐어 코딩 정적 분석 적용
SSL 적용	• SSL(HTTPS) 기반으로 운영 될 수 있도록 개선	• 2048 비트 공개키 SSL 적용
사용자 증적 자료	• 사용자의 행위에 대한 모든 증적 자료를 남기도록 요구	• 전수로그 수집 • LiveFramework 에 사용자 이용 증적 자료 - 사용자 요청에 대한 모든 정보를 기록 할 수 있는 방안 마련 (Servlet Filter 를 활용하여 샘플링 가이드)

마이크로서비스아키텍처 기반의 HIS 고도화 전환 사례, 건국대학교병원 이제관 기술사

보안 관련, 웹 취약성 조치 사항 (자동 프로비저닝 & 튜닝)

GUI 기반 웹시스템 설치/구성/튜닝 (웹시스템 프로비저닝)

- 웹서버/WAS/Datagrid 에 대한 자동 설치/구성/튜닝 GUI 도구 제공 (Red Hat Linux 계열)
- 입력한 정보를 기준으로 시스템 자동 구성

Apache /Tomcat/ JBoss 등 오픈 소스에 최적화된 APM

- 리눅스 종류와 버전을 식별하여 해당 OS에 맞는 웹서버/WAS 제품으로 설치
- 리눅스 OS/ Apache /Tomcat / Jboss 에 최적화된 모니터링

Troubleshooting 을 위한 다양한 분석 도구 제공

- 웹시스템 장애 시 즉각적인 원인 분석을 위한 다양한 도구 제공
  (Thread Dump / Netstat Dump / lsof dump)

전문가 수준의 튜닝과 정보 제공

- OS 커널 파라미터나 WAS Thread Pool 이나 데이터소스 Pool을 설정
- JDK, mod_jk, mod_cluster , 운영 스크립트 파일 등을 구성 요청에 맞추어 설정하여 설치

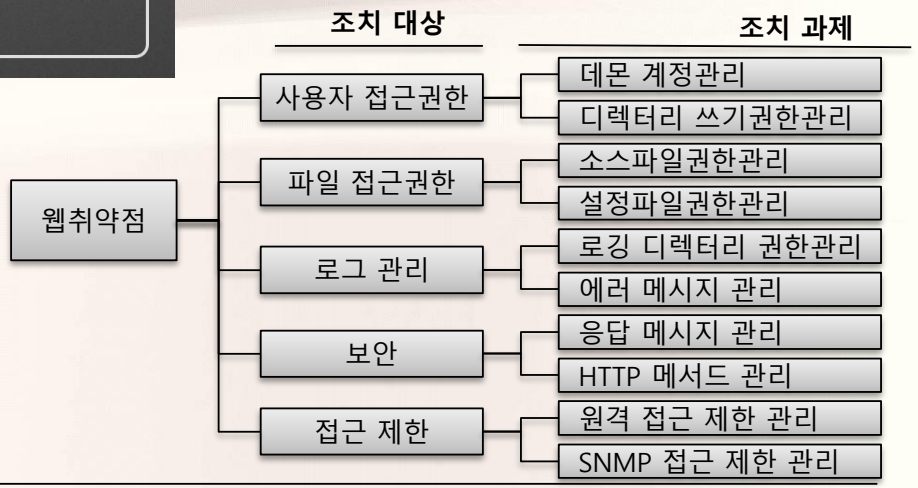
HTML5 인터페이스와 보고서 제공

- 웹브라우저 상에 별도의 Launch 없이 다양한 모니터링 기능 제공
- 웹관리 콘솔 접속 정보 등 웹/WAS 환경 운영시 참조할 수 있는 가이드 제공

KHAN [apm]

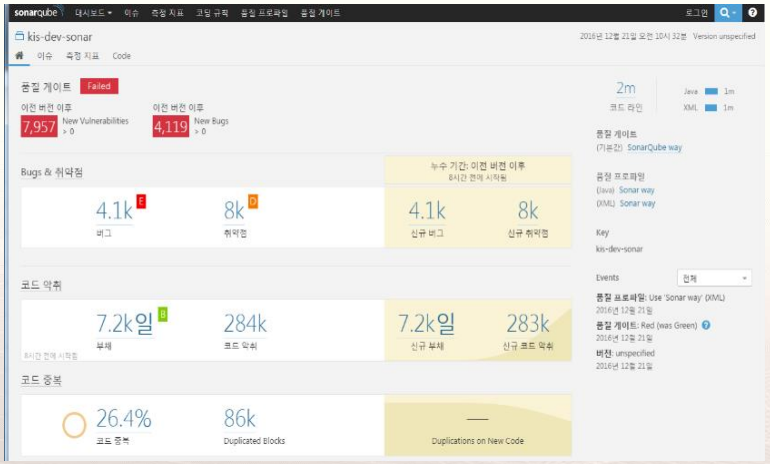
opennaru

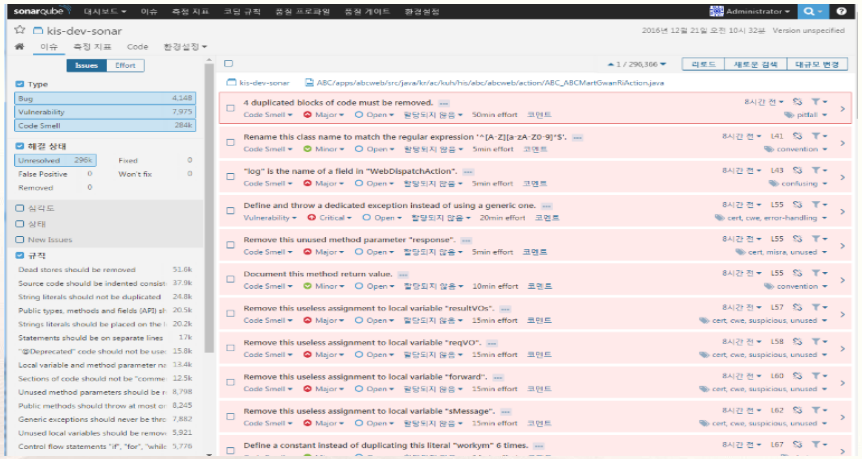
KHAN [apm]을 이용한
자동 설치시 웹 보안 취약성관련
설정을 자동 진행

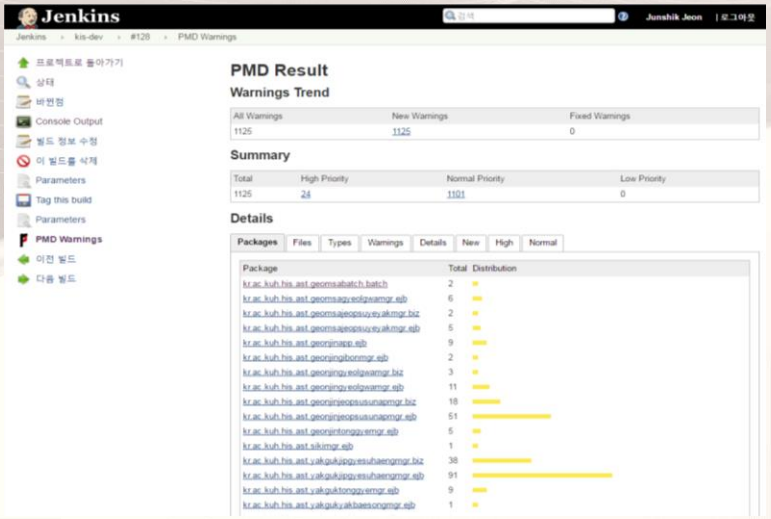


지속적인 개발환경, 시큐어 코딩 (정적 소스 분석) 점검 환경 구축

- 오픈소스 **소나큐브**를 이용한 기반의 **개발소스 보안 품질점검** 환경 구축







"92%의 보안 취약점이 네트워크가 아닌 어플리케이션에서 발견" – *NIST*

"릴리즈 이전에 소프트웨어 취약점을 50% 줄이면, 침해사고 대응 비용이 75% 감소" – *Gartner*

"릴리즈 이후 오류를 수정하고자 할 경우 설계 단계보다 100배 증가 " – *IBM*

"릴리즈 이후 오류를 수정하기 위해서는 약 $30,000 의 비용 소요, 하지만 개발 중 오류를 수정하기 위해서는 약 $5,000 면 충분" – *NIST*

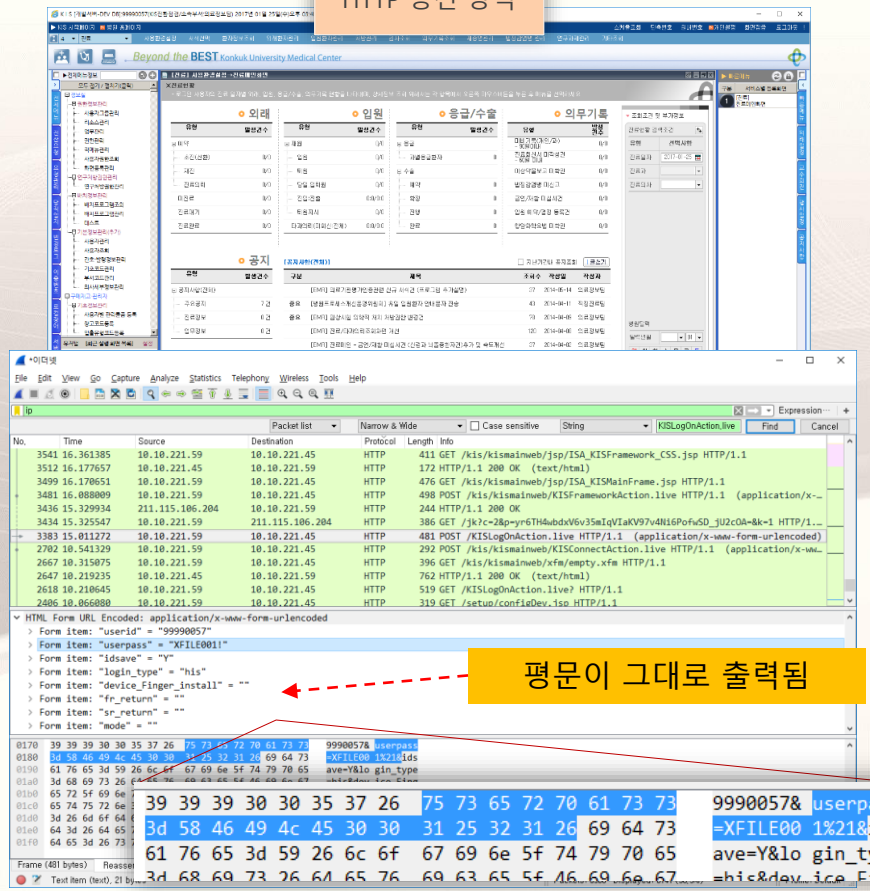
HIS, 통신구간 암호화 SSL 적용

SSL 인증서 적용

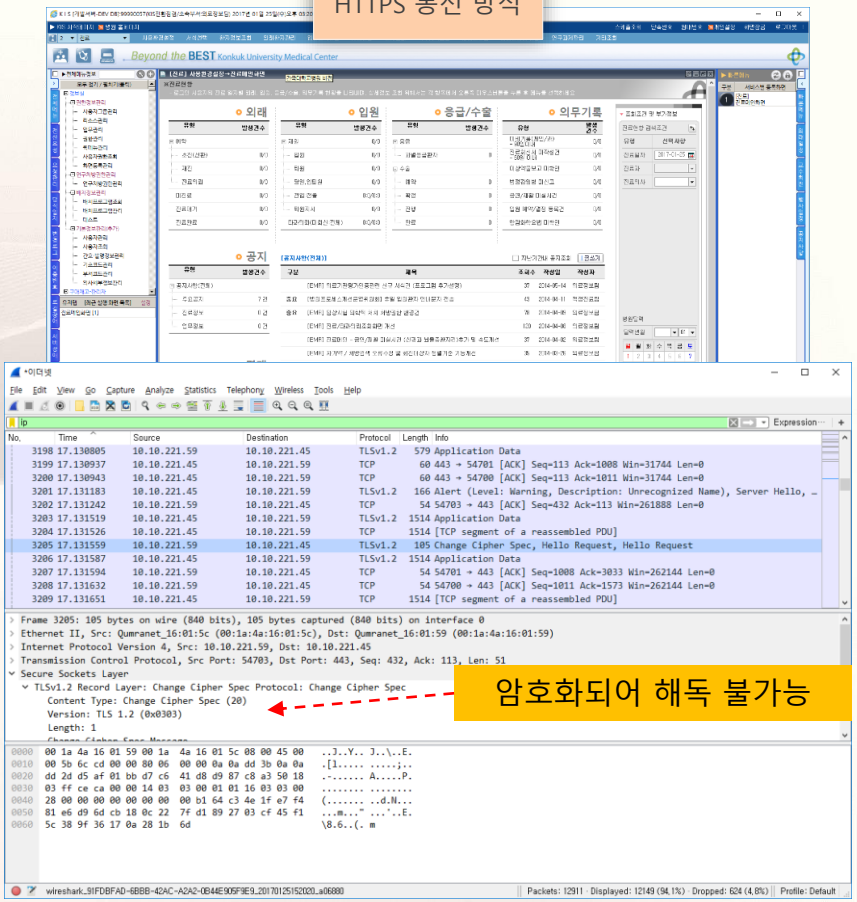
- Apache 웹서버에 SSL 인증서 적용
- HTTP 통신 구간이 암호화되도록 설정

- HTTP와 HTTPS를 함께 운영할 수 있도록 구성
- SSL 2048 bit 암호화 적용

HTTP 통신 방식



HTTPS 통신 방식



ISMS, WAS 사용자 전수 행위로그 수집, 수집결과 Splunk 연동

ISMS 사용자 모든 이용 증적 남기기

- KIS 시스템에서 공통적으로 사용하는 Framework의 LogOnFilter를 확장
- 사용자 페이지 호출 증적에 필요한 정보를 별도의 파일로 저장
- 기존에 사용 중인 로그 분석 시스템에서 파일을 주기적으로 읽어 감
- 기존 로그 분석시스템에서 사용자 증적 검색 / 분석

수집목표사항 :
"어떤 사용자가 언제, 어떤 자료에 접근했는지에 대한 로깅 생성"

ISMS 사용자 이용 증적 확인 절차

LiveFramework LogOnFilter

모든 요청이 통과해야만 되는 공통 모듈

호출시각, 호출URL, 메소드, 클라이언트 IP주소, 요청파라미터, 사용자 ID

CSV 파일 형태로 별도의 파일을 남김

```
2017-02-08 09:00:51,025 [INFO ] [http-thread-pool-threads - 144] SERVICE-LOG[serviceLogProcess:324] -
/kis/kismainweb/KISMenuContentsAction.live,GET,10.10.221.64,10.10.221.45,:80,99990057,defaultroleflag=1&userid=99990057&occukind=10&userdeptcd=000010&Mode=reqGetMenuC
ontents&svrccode=SVRC00040000000&
2017-02-08 09:00:54,348 [INFO ] [http-thread-pool-threads - 148] SERVICE-LOG[serviceLogProcess:324] -
/com/commonweb/GCCodeLoader.live,POST,10.10.221.64,10.10.221.45,:80,99990057,Data6=&Data7=&Data8=&Data9=&Data2=&Data3=&Data4=&Mode=reqGetGCCodeList&Mode=&Data5=&Data1
=&codeflag=1459,2003&Data11=&Data12=&Data10=&
2017-02-08 09:00:54,377 [INFO ] [http-thread-pool-threads - 149] SERVICE-LOG[serviceLogProcess:324] -
/ocs/cheobangweb/GetOCSOPMBSCALM01.live,POST,10.10.221.64,10.10.221.45,:80,99990057,Data6=&Data7=&Data8=&Data9=&Data2=&Data3=&Data4=&Mode=reqGetOCSOPMBSCALM01&Data5=&
Data1=201702&Data11=&Data12=&Data10=&
```

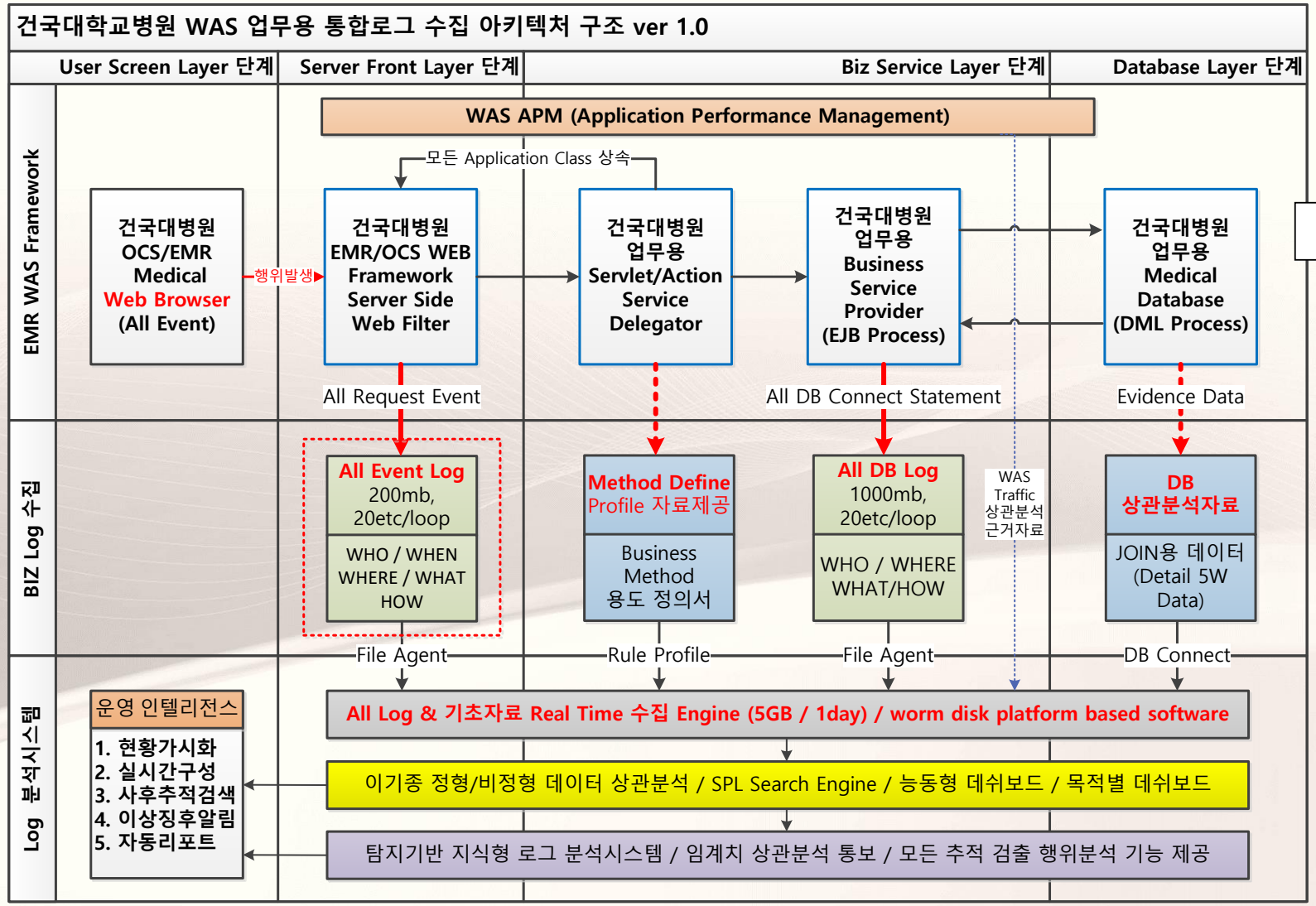
TO DO

Server

병원 로그분석 시스템

사용자 증적 검색/분석

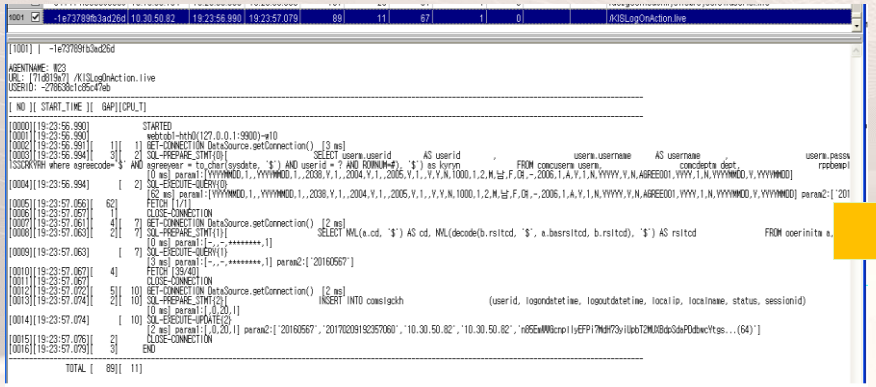
ISMS, WAS 사용자 전수 행위로그 수집, 수집결과 Splunk 연동

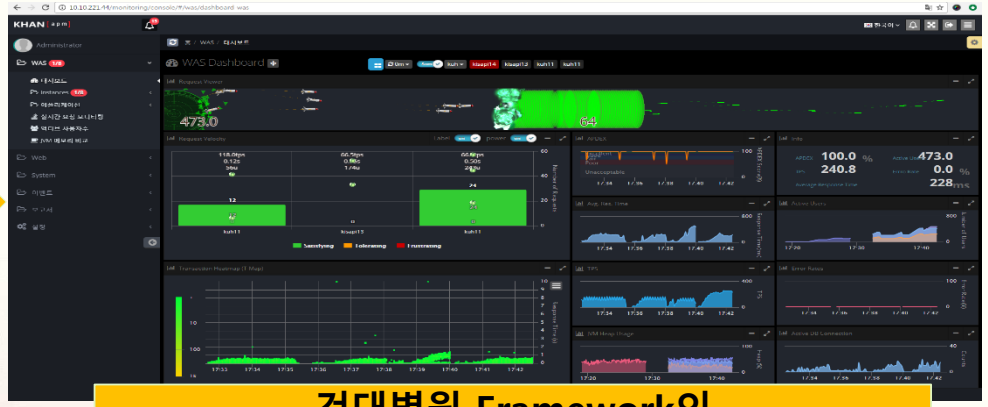


비즈니스 행위로그분석

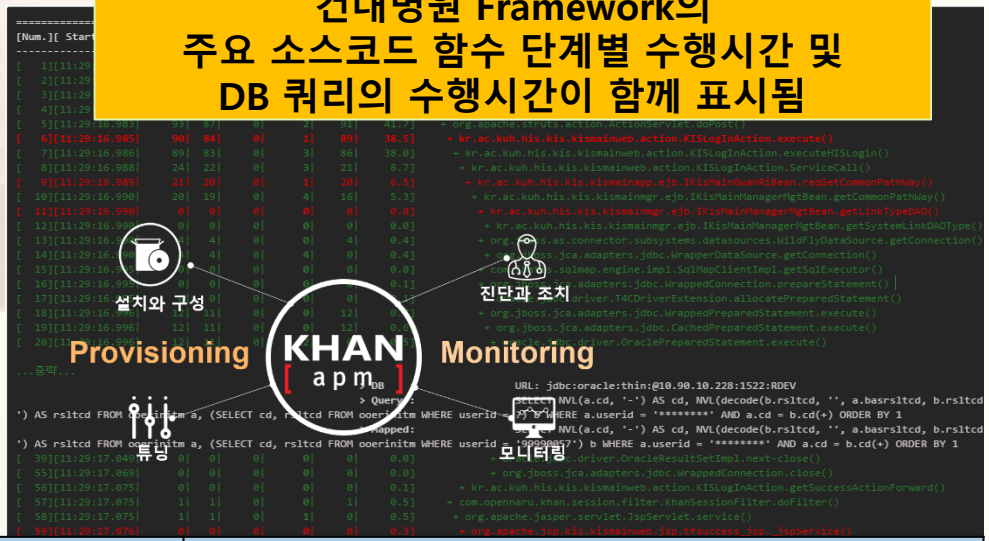
APM, HIS 운영자 중심 → 개발자 성능분석 중심의 모니터링 개선

DB 쿼리의 주요작업 수행시간만 표시됨





건대병원 Framework의
주요 소스코드 함수 단계별 수행시간 및
DB 쿼리의 수행시간이 함께 표시됨



항목	AS-IS	TO-BE
주요 차이점	• 관리자를 위한 **대시보드 위주**	• **개발자/운영자**를 위한 **실질적인 장애 분석 기능 위주**
Trace 레벨	• DB 쿼리 위주의 수행시간 측정	• DB 쿼리 수행시간 및 건대병원 Framework(Action/EJB/JSP) 주요수행시간 상세측정

서비스 고도화를 위한 MSA 기반의 WAS 고도화 설계 원칙

- 1차 U2L 검증사업의 결과

- 전환검증 결과가 만족할 만한 성과존재, 시나리오와 일치되게 전환간 우려된 이슈 대다수 해결
- 기존 시스템의 구성을 Linux기반의 OpenSource 표준구조 기반 x86 환경에서 구동하도록 제작함
- 전환 검증완료된 구성은 클라우드 플랫폼 서비스 구성환경과 궁합이 맞고, 표준클라우드로 전환에도 수월한 방식
- 전환검증에 따라 개발관리, 운영관리 환경의 불편요소 해소방안 정리
- 최신의 기술개발 환경 접목에 유연하고, 수월한 구성으로 업무 편의별 컨테이너 방식(MSA) 설계 가능 판단
- 자동 CI/CD 관리환경으로 ALM 고도화 방안 마련 (자동빌드, 자동배포)

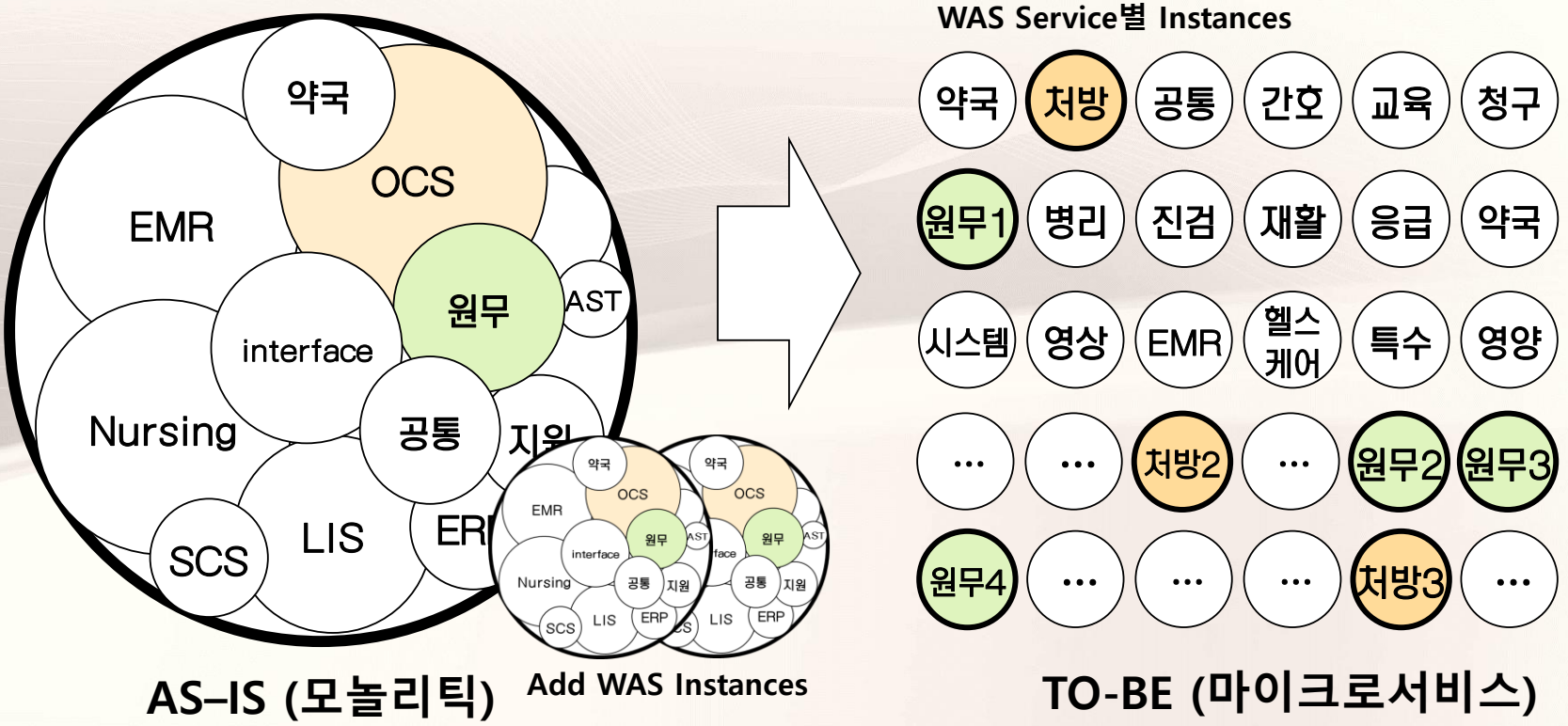
- 2차 전환작업시 이것을 어떻게 해결하고 싶었는가? TO-BE

- 해결의 핵심은 무엇인가? TO-BE의 KEY
  - MSA 업무별 아키텍처 : 업무별 서비스 목적에 따라 인스턴스를 제공하고, 필요시 증가/감소 시키는 구조
  - 최고 성능의 WAS 활용: WAS Booting 고성능 부팅 (2~3초 부팅, 소스 로드시 10초 이내 부팅)
  - 소스코드의 재활용 : 기존 EJB 재설계로 효율적 트랜젝션 응용 구성
  - Source 코드 관리: 브랜치가 자유로운 git 적용, hotfix 브랜치 모델 설계, 개발자 자유도 존중
  - Application 수명관리: 자동 원칙 / DevOps 구현, ALM CI/CD, 품질QA, 개발환경
  - 응용 최신기술 검토: Docker/Openshift, 다종류 FW공존 (Struts/Spring/전자정부 등), TCO

마이크로서비스 아키텍처 (Micro-Service Architecture) ?

- 의료정보시스템에서의 MSA 구성사업? 건대병원은 왜?

마이크로서비스 아키텍처 기반의 의료정보시스템 고도화 전환 고려

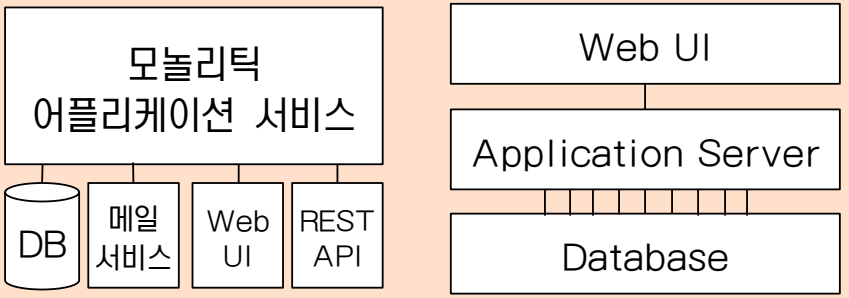


마이크로서비스 아키텍처 (MSA) 기본이해

- 마이크로서비스 아키텍처 개념

  - 하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 변경과 조합이 가능하도록 만든 아키텍처

- 마이크로서비스 아키텍처 vs 모놀리틱 아키텍처 비교를 통한 이해

MICROSERVICE ARCHITECTURE	MONOLITHIC ARCHITECTURE
	
- 시스템 복잡할 수록 서비스 단순화 구현용이 - 필요 서비스만 스케일 아웃 - 저비용 확장구조	- 하나의 프로젝트 관리 및 단일패키지 배포 - 서비스 확장시 동일서버의 구조적 증설 필요 - 고비용 확장구조
- 개발조직문화 업무별 유연한 변화유도 가능 - 서비스 단위 독립배포가능, 집중 성능 개선 - 상호 영향도 최소화, 이기종 F/W서비스 구동	- 서비스 지속성장 및 규모증가시 한계 - 대규모 개발시 서비스 복잡, 파생된 버그 증가 - 재활용 저하, 중복코드 증가

마이크로서비스 아키텍처 특징

- **일반적인 특징 (마틴 파울러)**
  - 마이크로 서비스(MSA)는 어플리케이션 **아키텍처 스타일** 임
  - MSA 서비스들은 **작은 서비스들의 집합**으로 어플리케이션을 개발하는 방법임
  - MSA 서비스들은 각각이 프로세스 임
  - MSA 서비스들은 서로 HTTP, JMX, JMS, AMQP, STOMP, REST API 같은 **가벼운 통신 메커니즘**을 사용함
  - MSA 서비스들은 비즈니스를 구현하고, 각각은 **완전히 자동화된 방법으로 독립적으로 배포**함
  - MSA 서비스들은 **중앙 관리는 최소화** 함
  - MSA 서비스들은 **다양한 프로그래밍 언어로 개발** 될 수 있음
  - MSA 서비스들은 **다양한 데이터 저장 기술을 사용**할 수 있음

- **MSA 왜 필요한가?**
  - 애플리케이션을 특화된 기능별로 나누게 되면 자연스럽게 애플리케이션의 추상화(abstraction)가 가능해 짐
  - 해당 API를 유지한 상태에서 세부적인 구현 내용을 언제든지 손쉽게 개선하고 변경 가능

마이크로서비스 아키텍처 (MSA) 기본이해

- **MSA 언제 사용하면 좋은가?**
  - 애플리케이션의 배포에 긴 시간이 소요될 때 (ex: 과거 35~40분 소요)
  - 단순한 기능 하나를 수정해도 전체 기능에 대한 QA가 필요 시 (ex: 서비스영향도 분석이 안되어 통 서비스)
  - 단순한 버그 수정이 더 중대한 버그를 생산하는 일이 많을 시 (ex: 소스의 상호연계가 Tightly Coupled)
  - 현재의 애플리케이션을 기능별로 나눈다고 가정했을 때 수십개의 마이크로서비스가 가능시 (원무, 간호, 진료…)

- **MSA 단점 극복을 위해 필요한 사항 (설계조건)**
  - 서비스 많아져서 전체적으로 관리포인트가 많아짐
  - 서비스 빌드 및 배포 자동화등 ALM의 구성이 필요함
  - WAS 부팅시간이 오래 걸리지 않도록 빠른부팅의 WAS사용과 부팅 모듈 최적화가 필요함
  - 많은 WAS의 종합적인 모니터링이 필요함

- **Monolithic 시스템에서 MSA로 전환설계 (검토대상)**
  - 인프라 측면 : 확장성, 가용성 확보여부, 시스템 자원의 신속한 지원 여부
  - 미들웨어 측면 : 미들웨어 가용성, 확장성 위한 Scale Out, 자동화 포인트
  - 어플리케이션 측면 : Source간 Dependency
  - 데이터베이스 측면 : Datatbase간 Depenncy

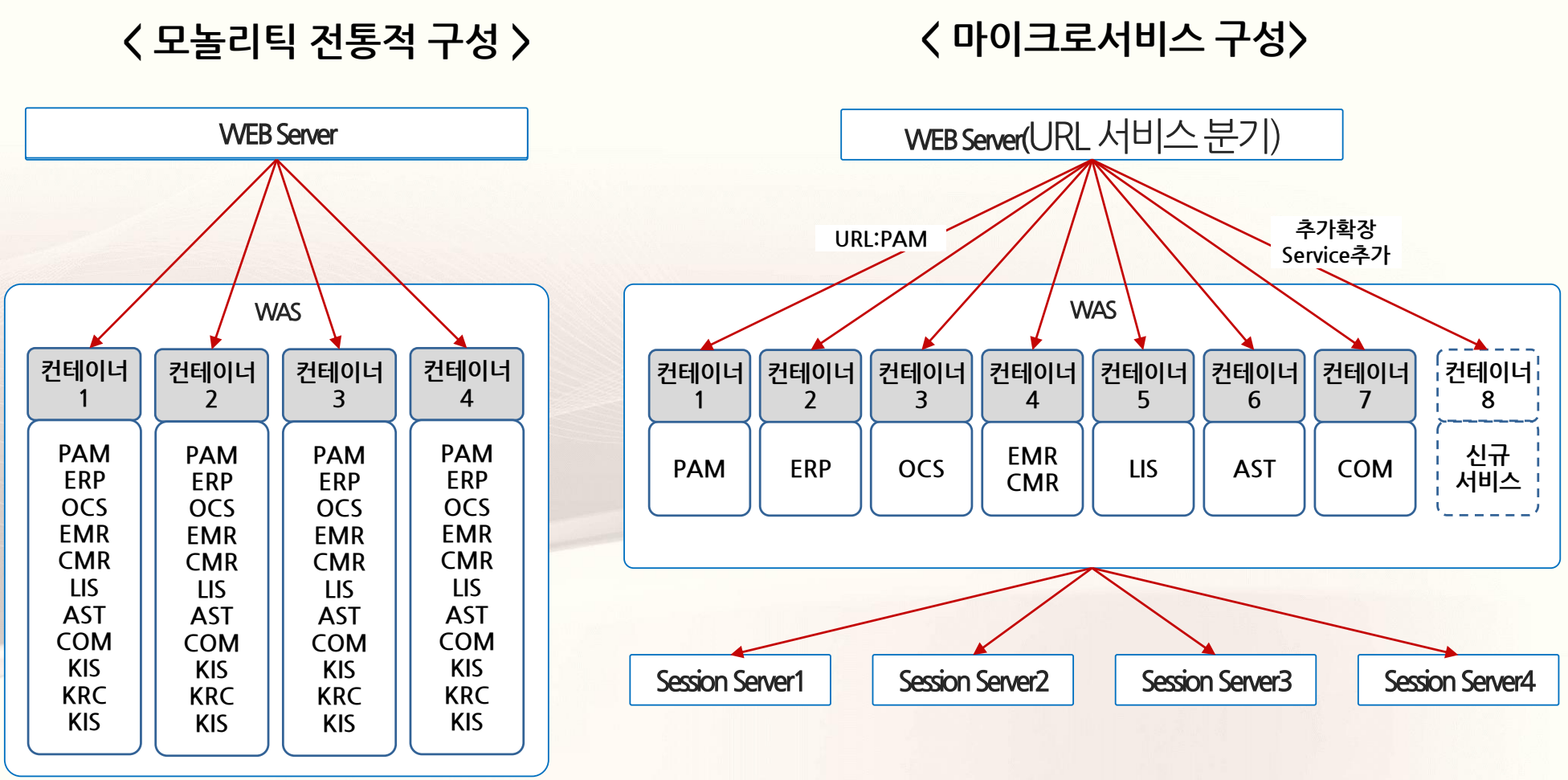
의료정보시스템의 업무현황 분석

- 통합 의료정보 시스템의 서비스 업무 구분

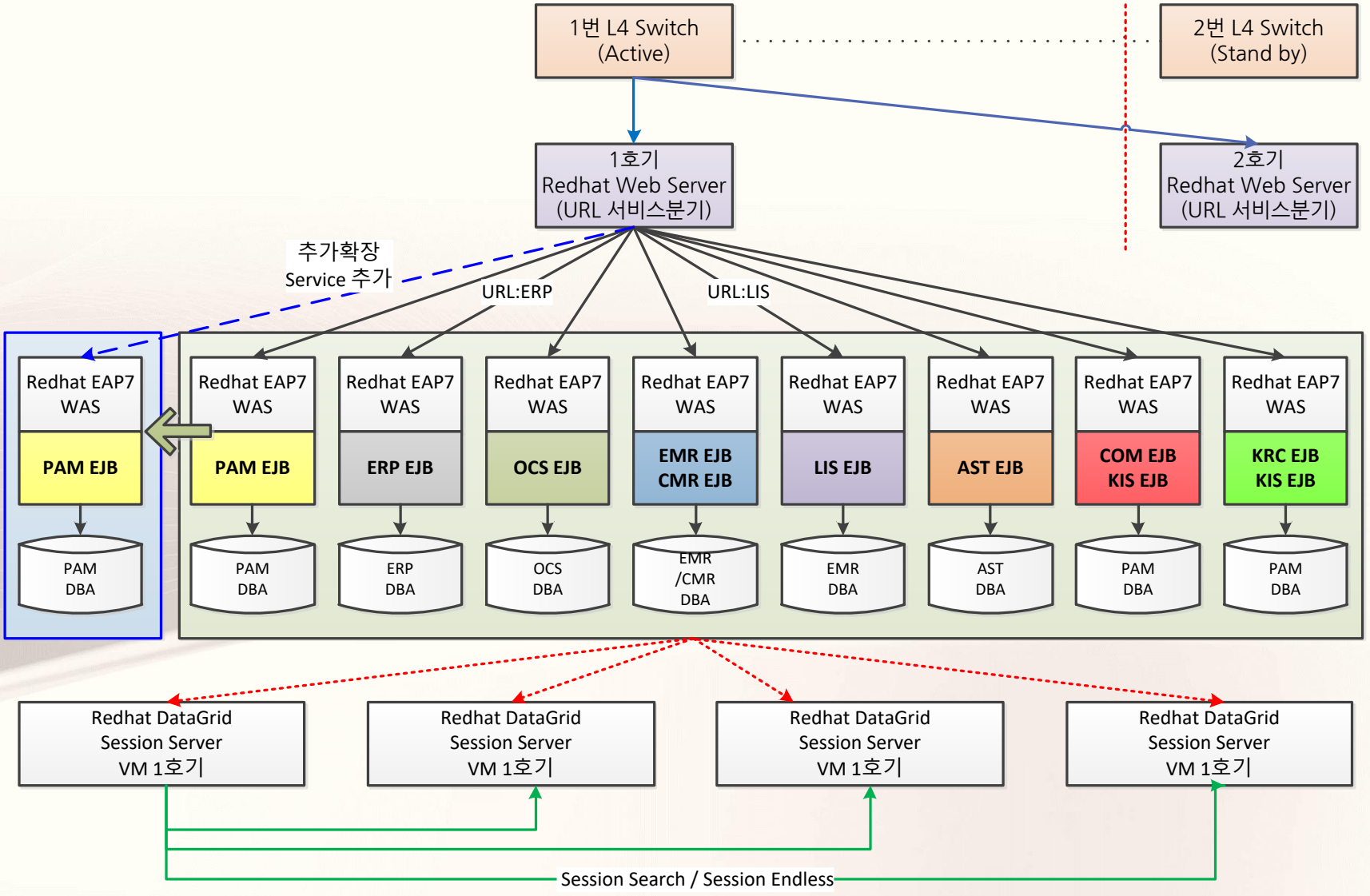
업무	ABC	AST	CRM	EMR	ERP	LIS	PAM	OCS	KRC	KIS	COM	epaper
	원가관리	진료지원	광스캔	차트작성	ERP	진단검사	원무, 수납, 접수, 청구	처방, 간호처방	병원간 인터페이스	메인시스템	공통모듈	전자동의서
내용	원가관리 DB Link 사용	검사, 처방, 진단 검사 결과 도출 검사 장비 인터페이스	종이 스캔 시스템 이미지 처리 로직	의사, 간호사 차트작성 동적생성 알고리즘 X-Ray 렌더링	SAP ERP와 유사 자체구축 인사, 회계, 재무	AST처럼 질병 원인 파악 (업무적)	키보드 모듈 심사 평가원 연동	환자의 접근/의사의 오더 병원의 커뮤니케이션 역할 수행	진료의뢰 리퍼센터 다른기관과의 연계	메일전송 로그수집 메뉴 권한관리	화면에 대한 공동 모듈 새주소, SMS, Message 처리	전자동의서 모바일 문서관리 시스템
소스 본수 (전체: 38,149)	213	3,938	227	3,963	7,479	5,764	6,184	4,266	709	1,638	3,768	-

- 기존 거대한 단일화된 종합업무 구성 서비스 구조를 목적별 서비스로 분리
- 서비스 EJB 별로 그룹핑 처리 및 Dependency 분석
- 서비스 배포시점(빌드/배포)에 EJB 설정으로 서비스 분류 구분 구성

2단계 사업 : 마이크로 서비스 아키텍처 미들웨어 구조방안

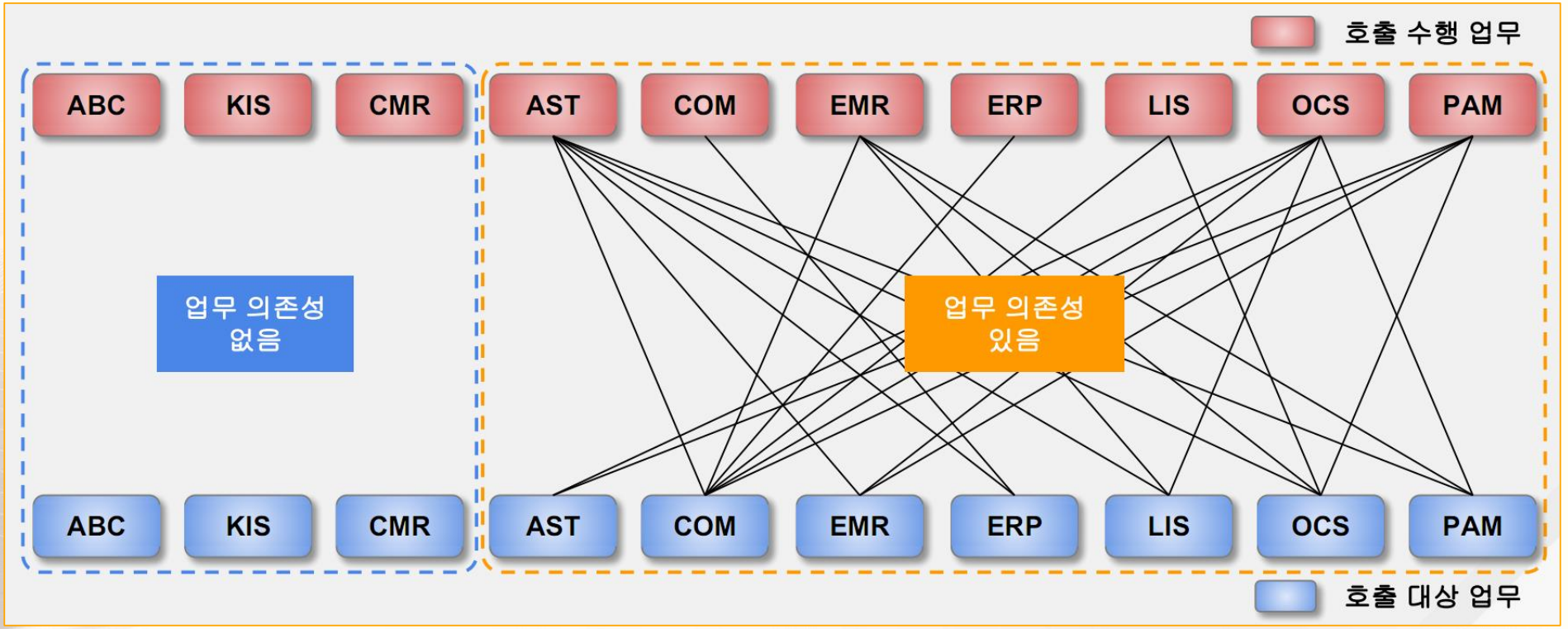


HIS 마이크로 서비스 아키텍처와 DB Pool, 세션 SET 구성



HIS 마이크로 서비스 아키텍처 서비스의 타 업무 의존성 분석

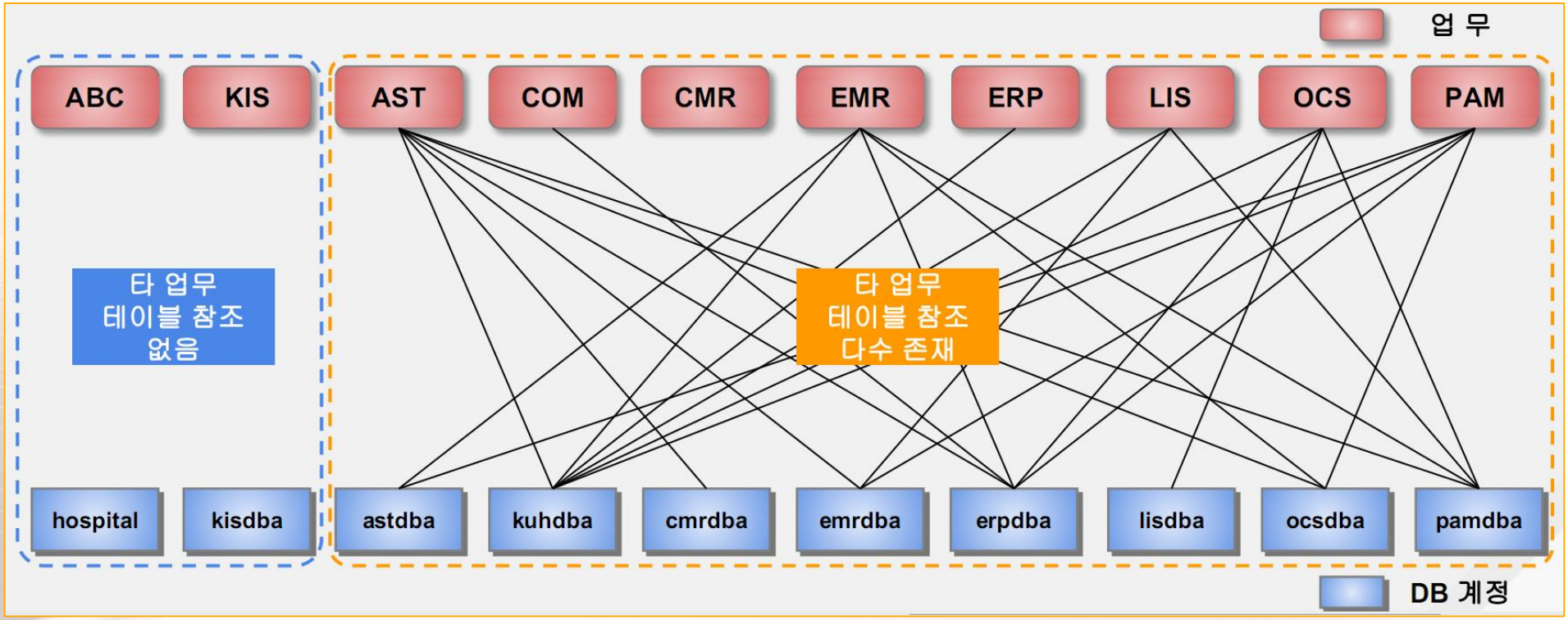
- **서비스 관점,** 업무 의존성 분석(타 업무의 EJB 호출) 결과



- 업무의 의존성이 있는 비즈니스 로직을 파악
- 의존성의 강도를 업무적으로 파악하고 분리 가능여부를 진단
- 함께 반드시 구성되어야 하는 서비스는 실질적으로 하나의 서비스 성격으로 처리
- 특정 케이스 시점에만 처리되는 로직은 EJB의 특성상 서비스 호출 Alive 방식으로 내부 호출 처리

HIS 마이크로 서비스 아키텍처 서비스의 타 업무 의존성 분석

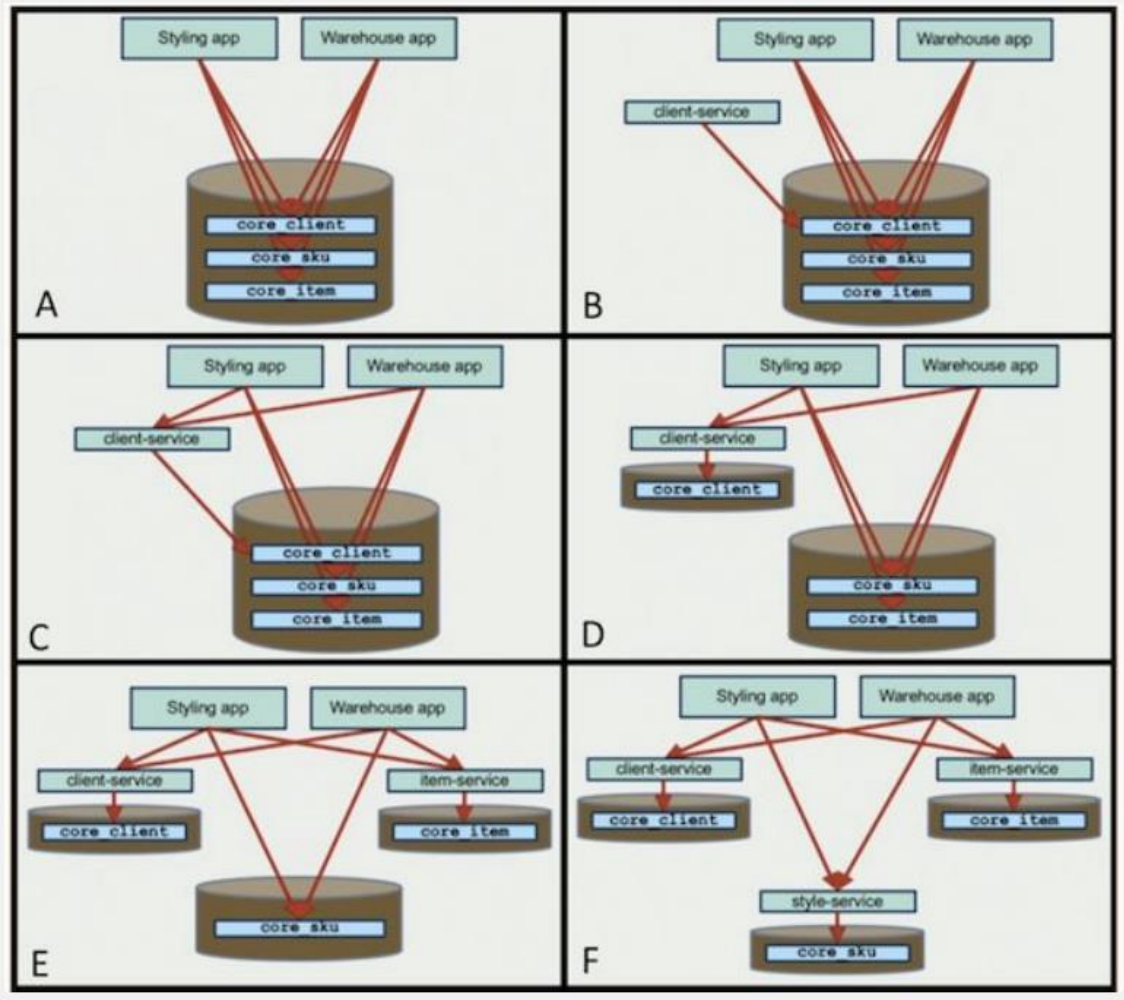
- DB 관점, 서비스 업무테이블 참조 분석결과



- 쿼리로 비즈니스 로직을 구성하는 시스템은 단위 업무별 MSA구성이 현실적으로 어려움
- 테이블을 직접 참조하는 패턴도 단순 조회성 질의가 아닌 JOIN을 포함한 복합 질의 형태 구성이 다수 존재
- SQL 쿼리에 업무간 상호작용이 많을 시 MSA 구성에 어려움이 존재
- MSA에서 JOIN을 내부업무간 JOIN과 타 어플리케이션 JOIN으로 구분하여 판단 필요

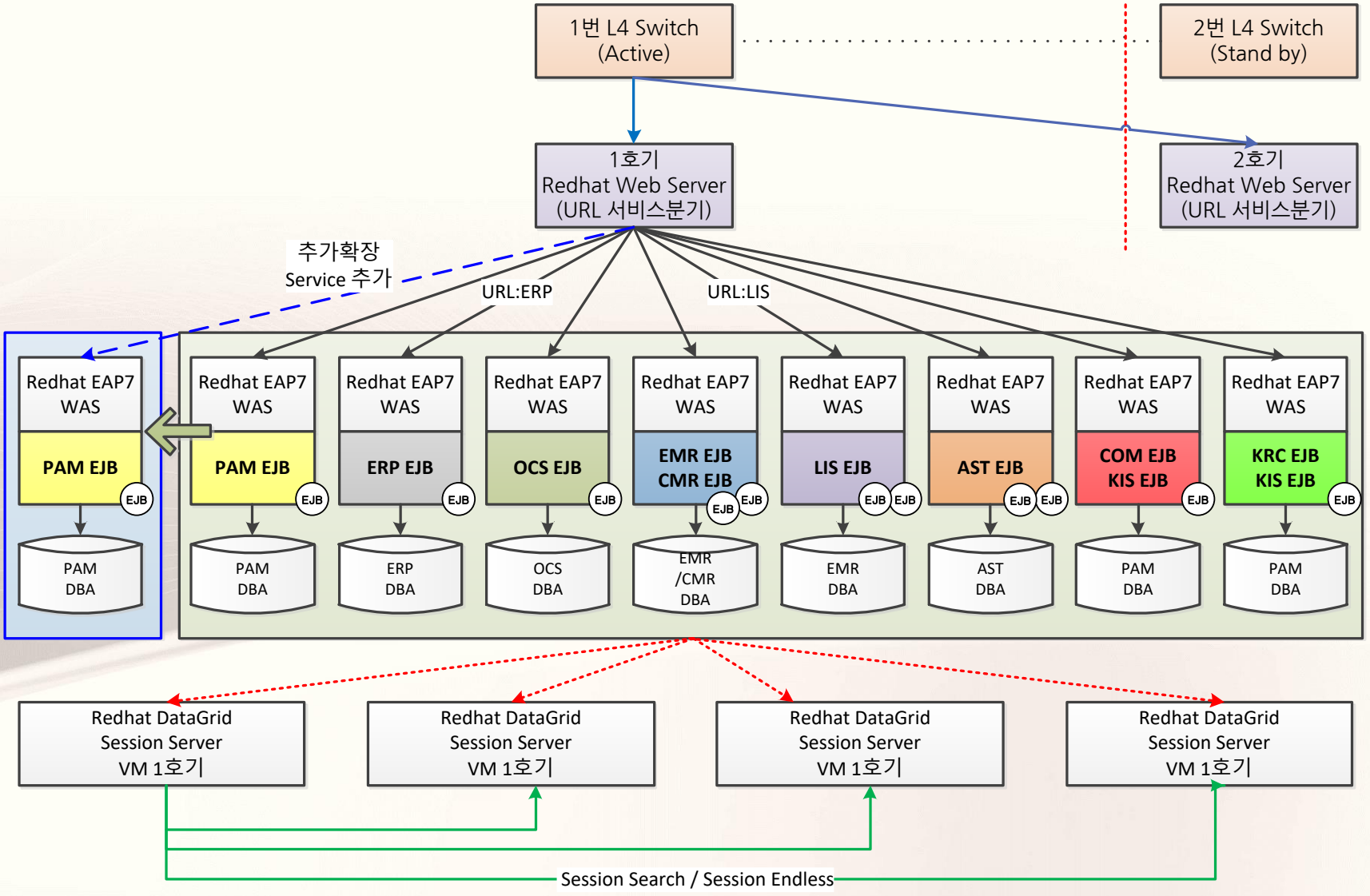
HIS 마이크로 서비스 아키텍처 서비스의 타 업무 의존성 해결

- DB 관점, 업무 테이블을 Rest API 서비스 로 전환하는 절차



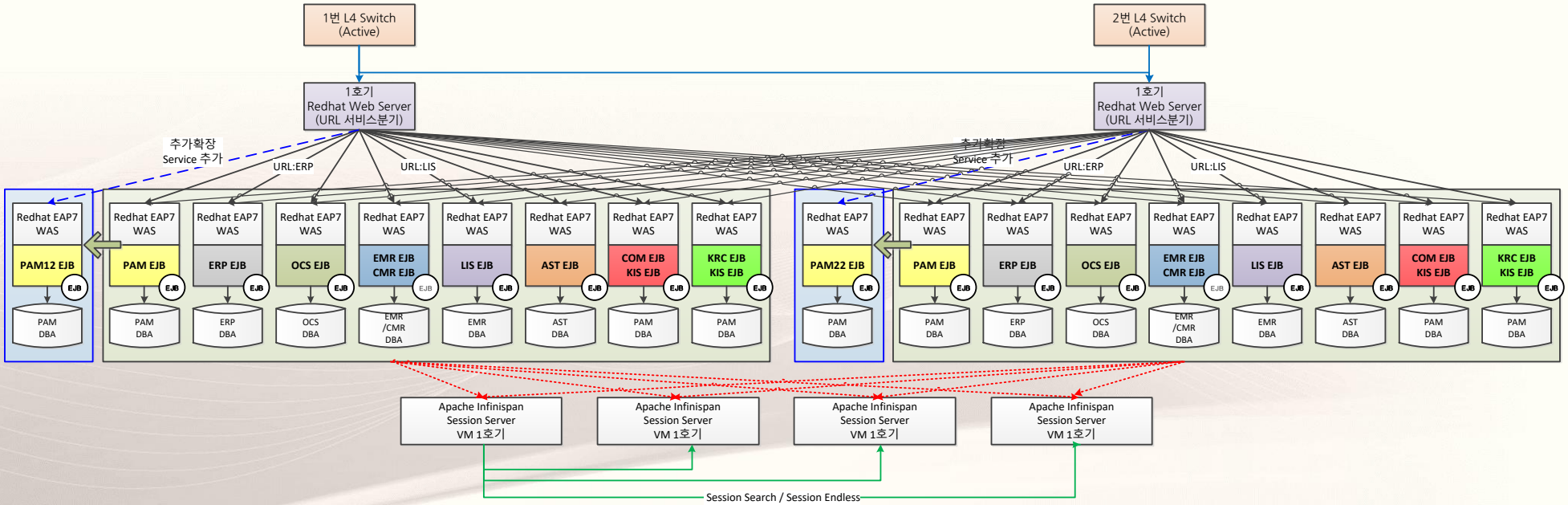
1. MSA 전환 대상 테이블 선정
   새로운 Rest API 서비스 개발
   (A,B)
2. 기존 MSA 대상 테이블 참조 업무
   새로운 Rest API 참조하도록 수정
   (C, D)
3. 1~2과정 지속, 점진적 MSA 전환
   (E, F)
4. 업무의 데이터 이용시 Rest API 이용
   타 업무의 데이터를 참조하는 MSA완성
   (F)

HIS 마이크로 서비스 아키텍처와 Dependency EJB 서비스 조합



마이크로 서비스 아키텍처 기반의 업무적 다중화 구성

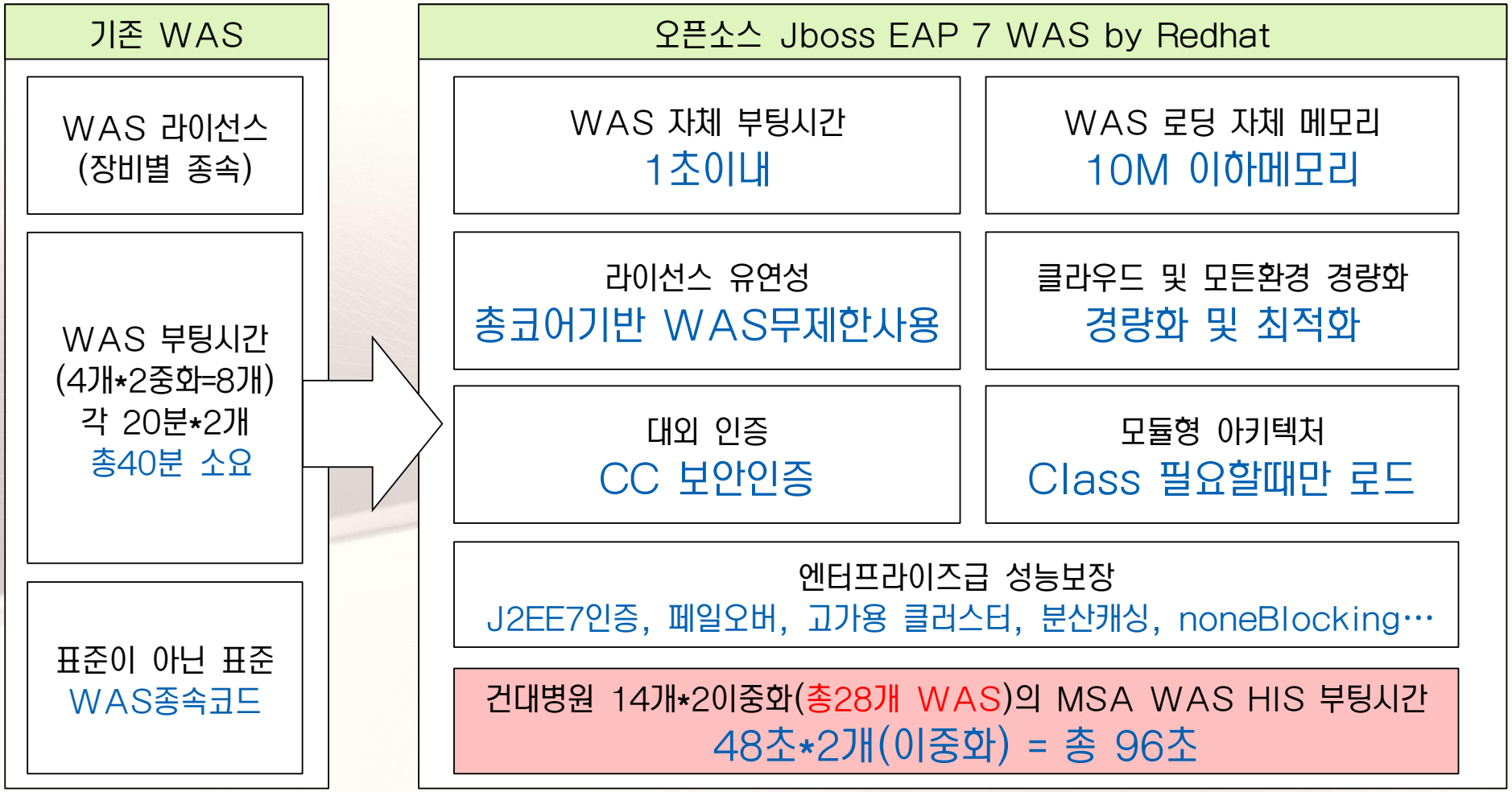
- 전체 서비스 업무적 EJB 종속성(서비스/DB)을 고려한 논리적 다중화 MSA 구성도



- 세션서버의 분리로 인해 기존 WAS 인스턴스가 같은 물리머신에 존재하지 않아도 됨
- WAS의 인스턴스는 HIS의 업무 부하가 있는 특정서비스만 필요시 증가해도 됨
  - 예) 진료/간호 EMR → 진료 EMR, 간호 EMR 분리가능 → 진료EMR 영역에서도 세분화 가능함
  - 예) 진료/간호 OCS → 진료OCS, 간호OCS 분리가능
- 개발 프레임워크가 다른 인스턴스도 MSA로 연결 구성가능함

인스턴스가 많은 WAS, WAS 로드/부팅시간 최소화 이슈

- WAS 인스턴스가 많으면 로드성능의 저하가 기술적 이슈가 될 수 있음

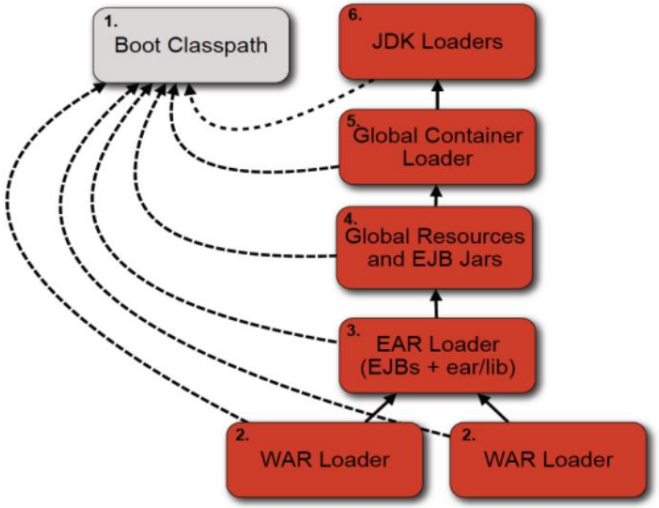


HIS를 MSA하기에 최적화된 오픈소스 Jboss WAS

- 선진화된 클래스로더: 모듈형 클래스로더 효과 (초고속 부트)

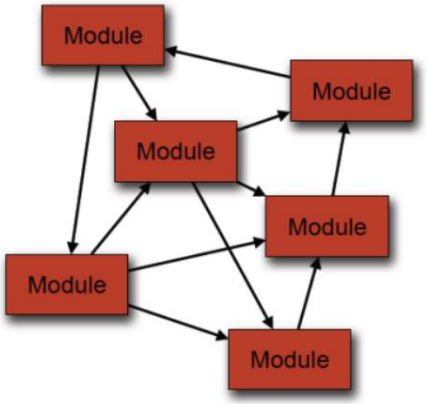
계층형 클래스로더

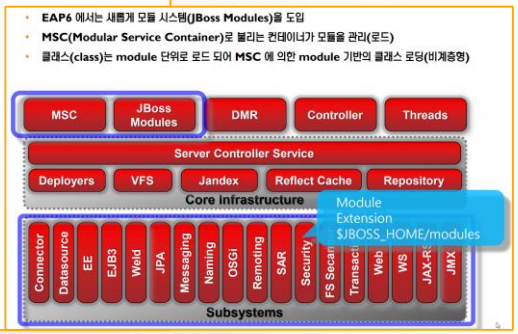
- 중복 배포
- 로드 순서에 의한 교착 상태 발생
- 복잡/ 클래스 검색이 늦음
- 중복 배포에 의한 오류 발생
- 클래스 공유
- 문제를 회피하기 위한 구조가 더 복잡도를 높여 악순환



모듈형 클래스로더

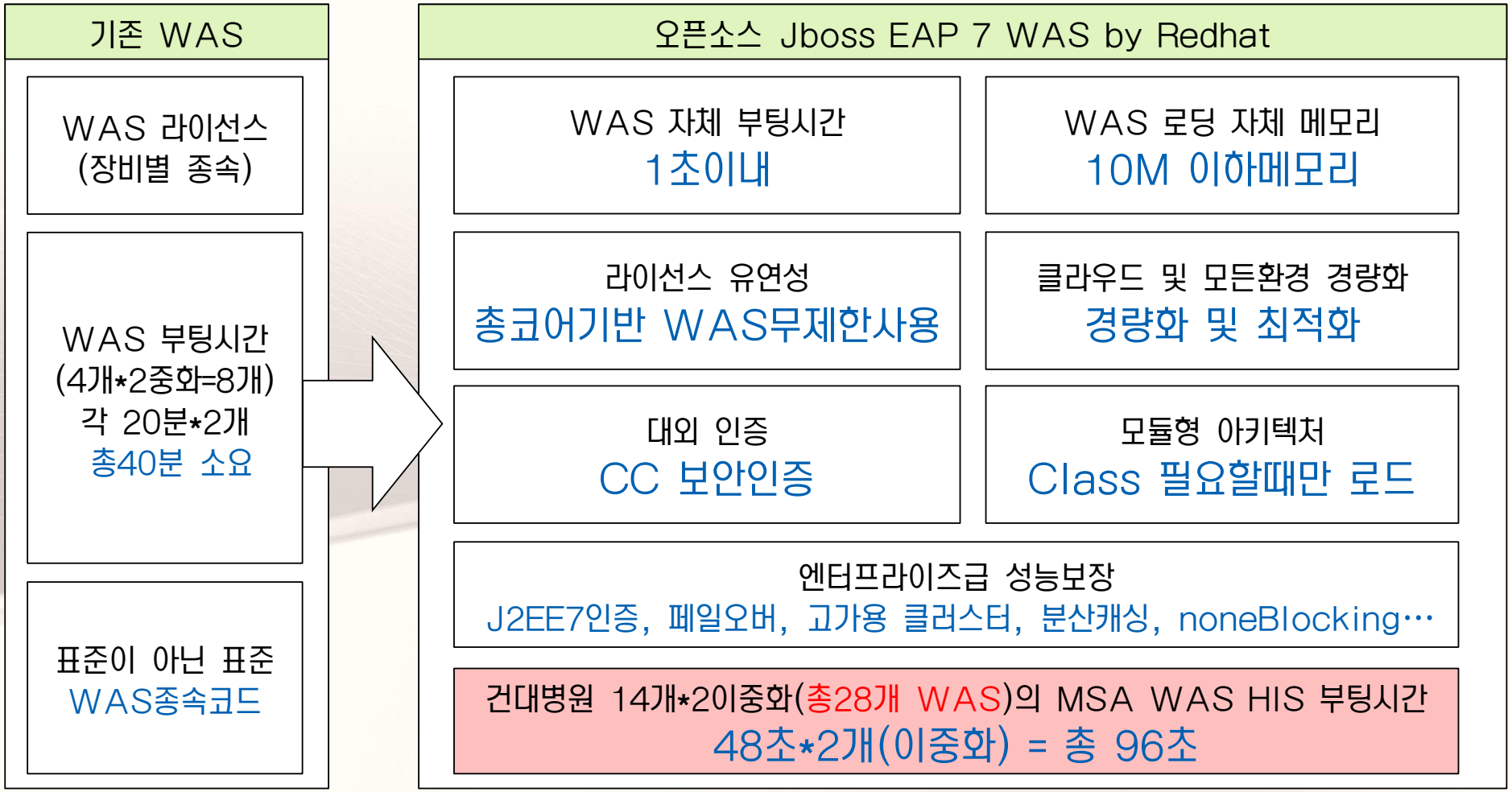
- 계층형 클래스 로더의 문제점 해결
- 모듈 하나에 대해서 하나의 클래스로더
- 각 모듈은 런타임으로 필요로 하는 모듈의 의존성을 정의
- 계층형이 아닌 그래프 구조
- 「클래스 패스」는 사라짐
- 단순하여 초고속



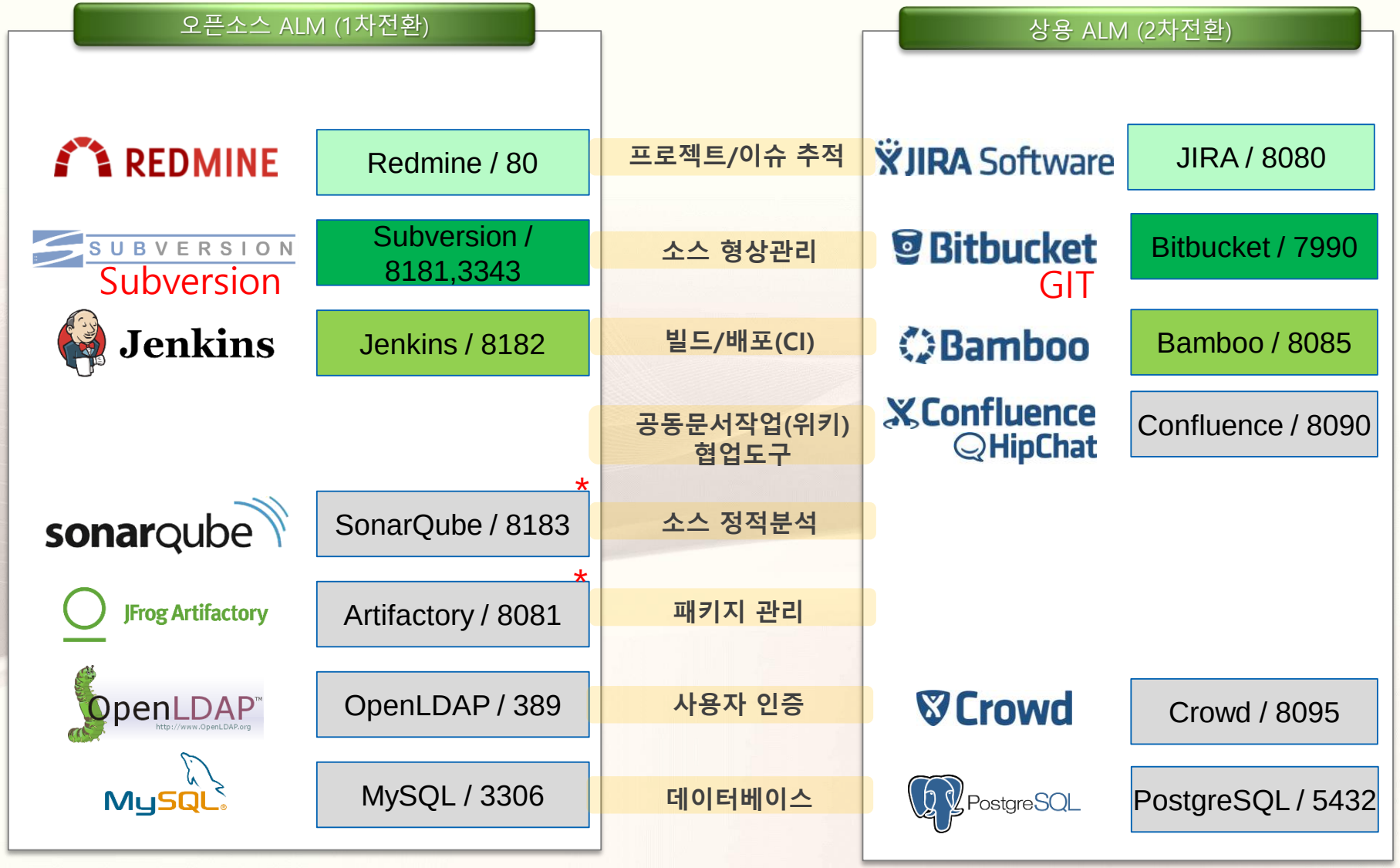


배포구조의 개선

- WAS 인스턴스가 많으면 로드성능의 저하가 기술적 이슈가 될 수 있음

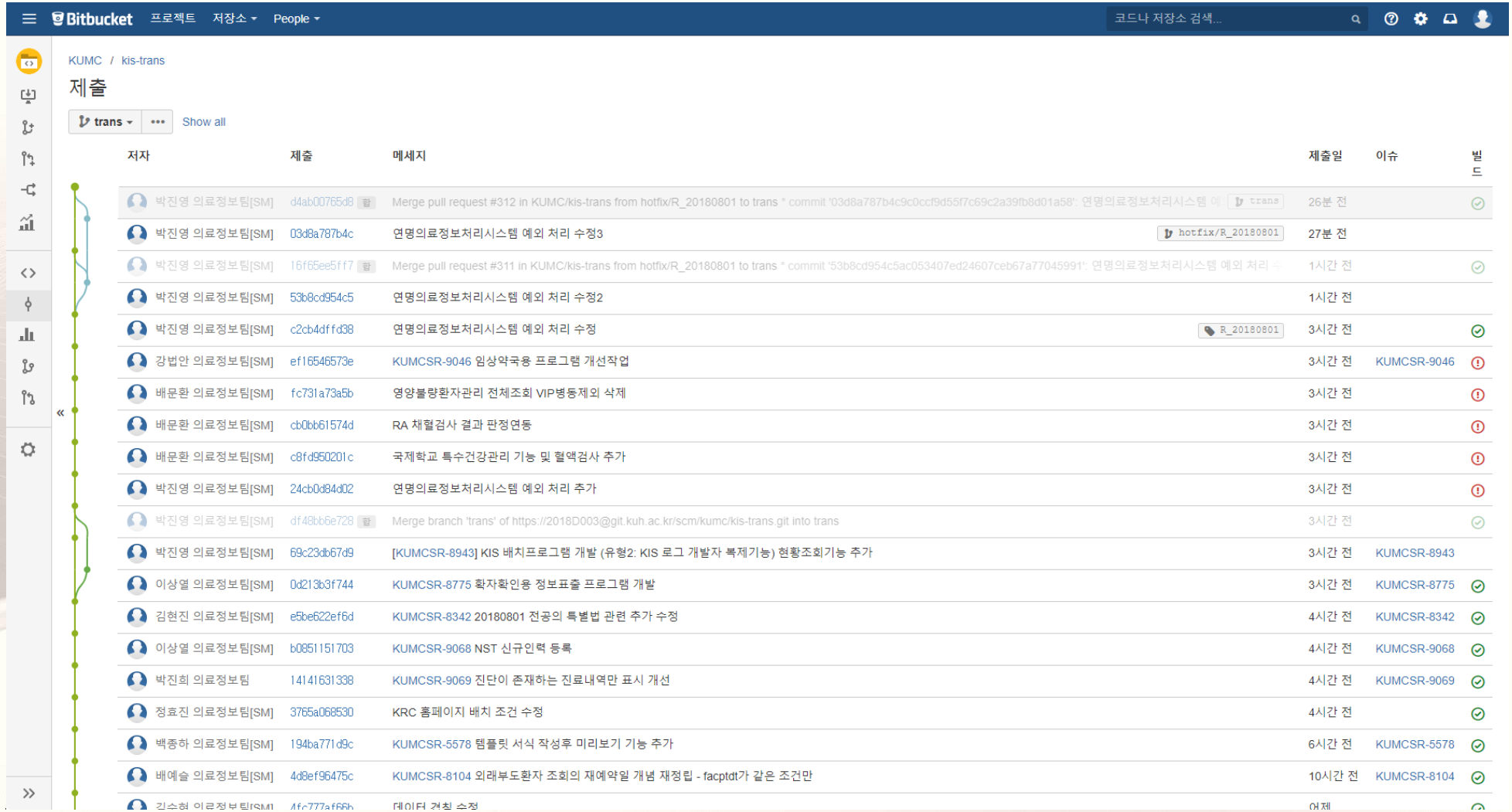


MSA를 위한 유연한 ALM 환경 고도화 전환결정



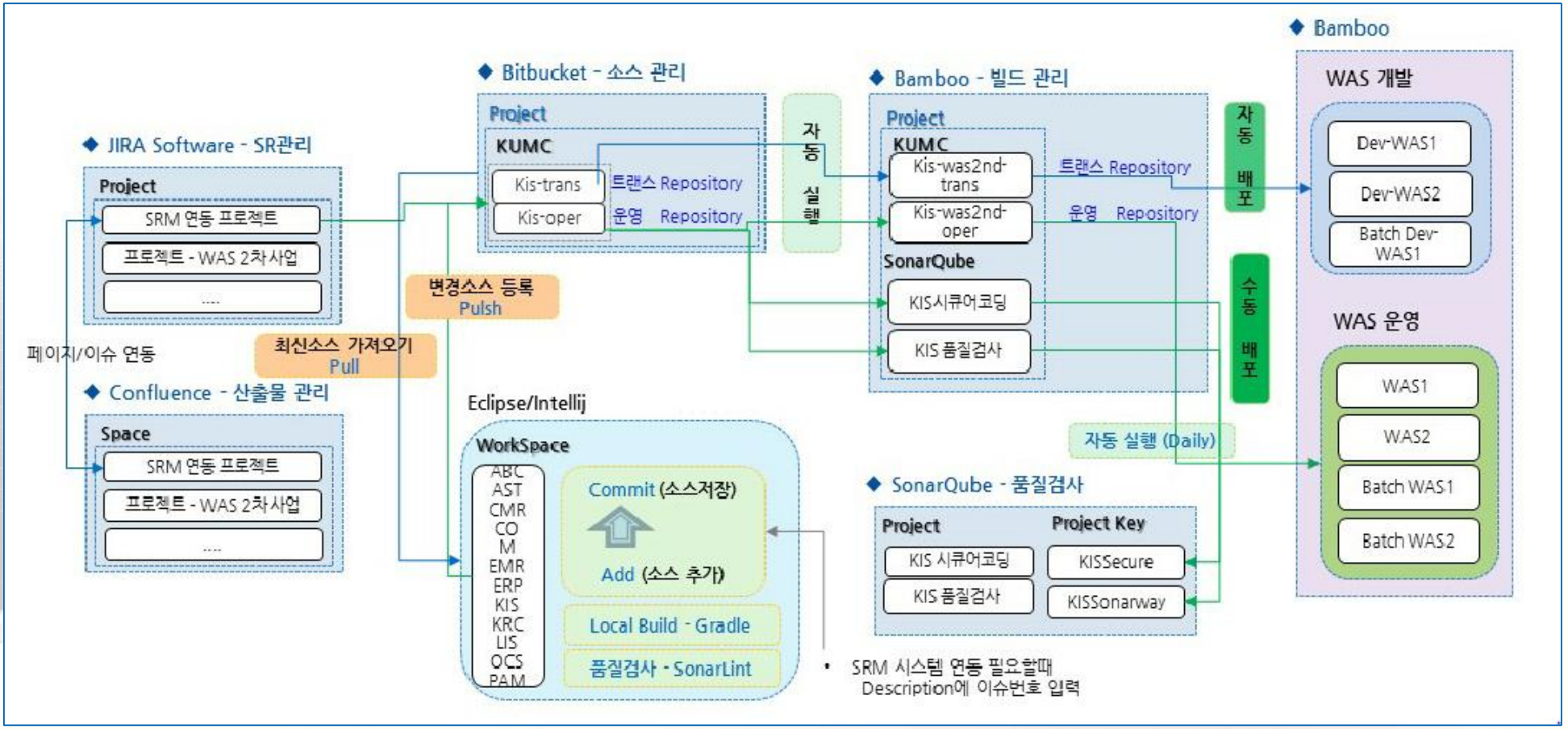
MSA를 위한 Branch 모델을 제공하는 로컬 깃허브(GITHUB) 구성

- 병원용 Local 깃 허브를 구성하여, 업무파트별 분산형 소스브랜치 작업을 자유롭게 진행함



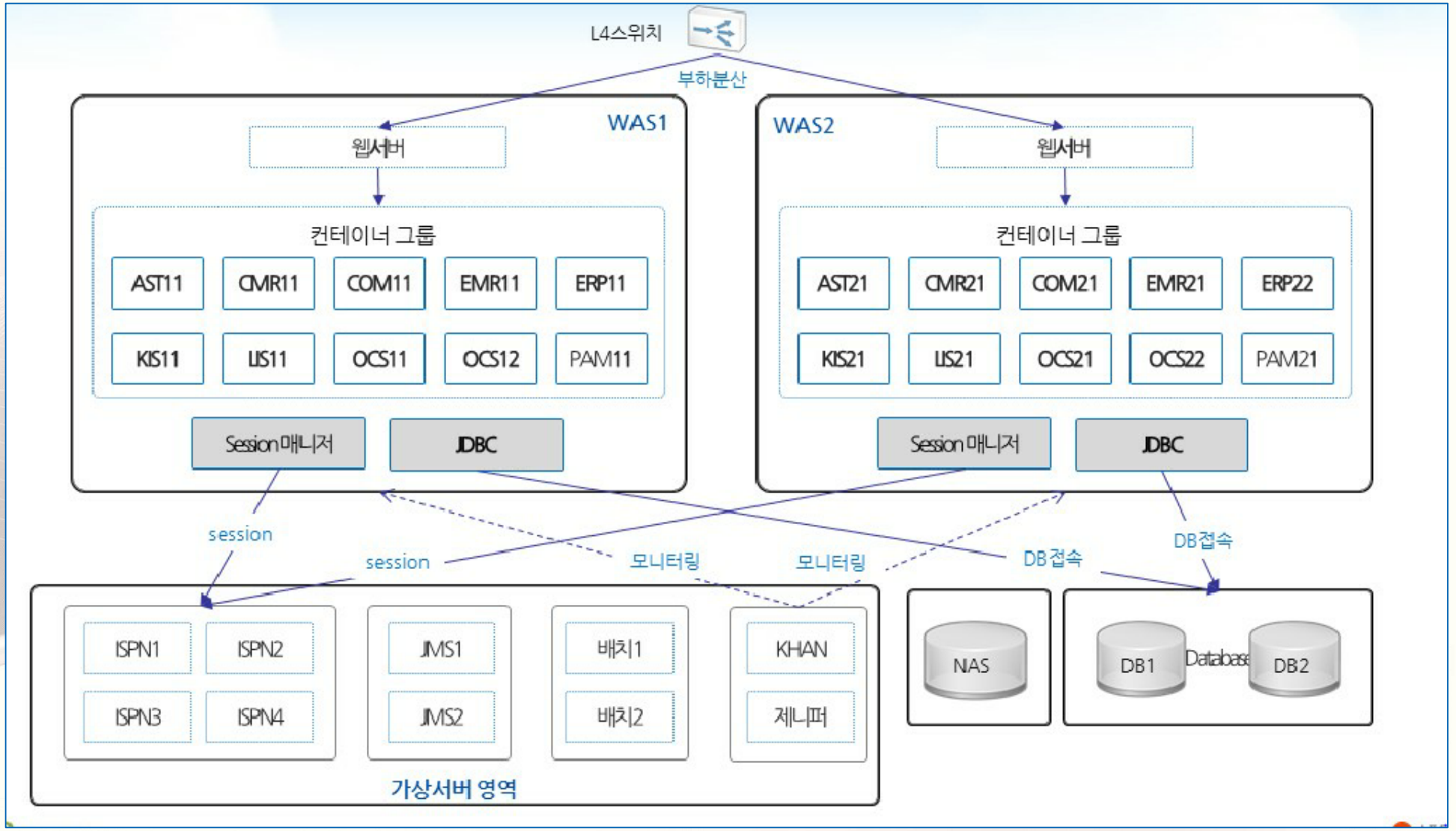
배포환경의 ALM 변경을 통한 MSA 마이크로서비스 구성 지원

- ALM 기반의 서비스 MSA 배포구성



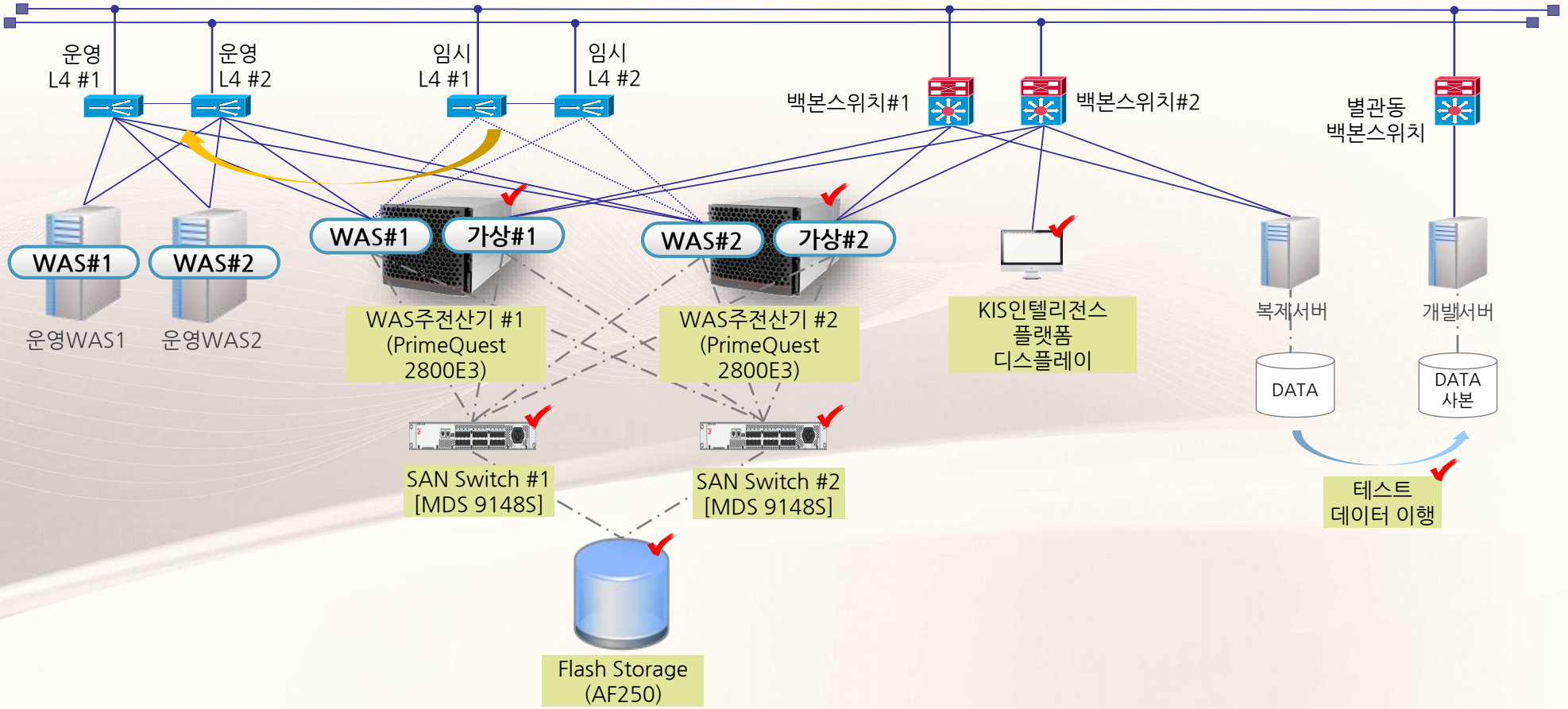
- 운영 MSA 인스턴스별, 배포모듈의 인식설정 및 서비스 인스턴스 실행

건국대학교병원 MSA 기반의 HIS 서비스 기본구성

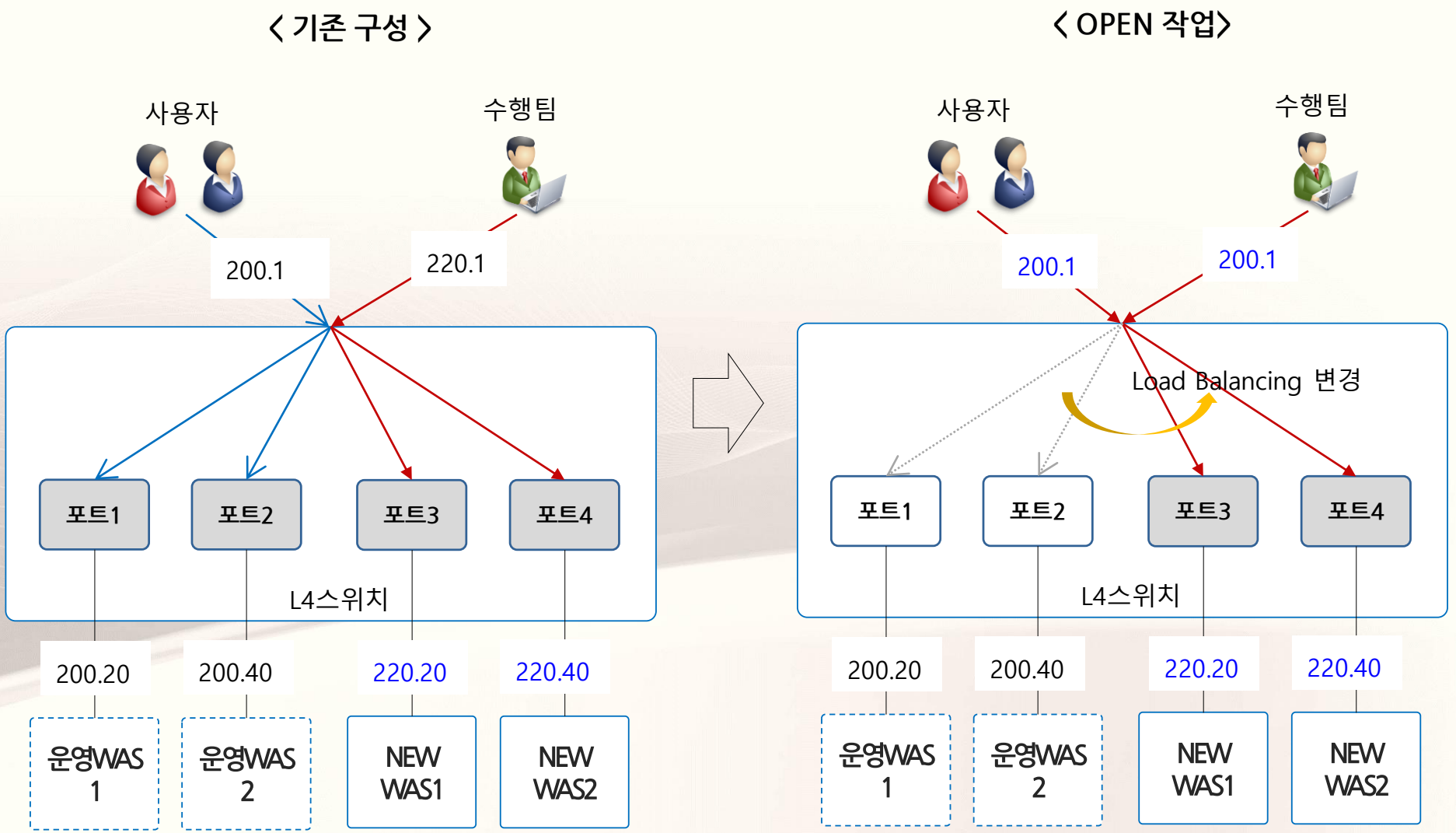


신규서비스 전환: Cross Over 기반의 자연스러운 서비스 전환

- L4 스위치 전환 작업을 통한 오픈 Down Time 최소화 (30초 미만)

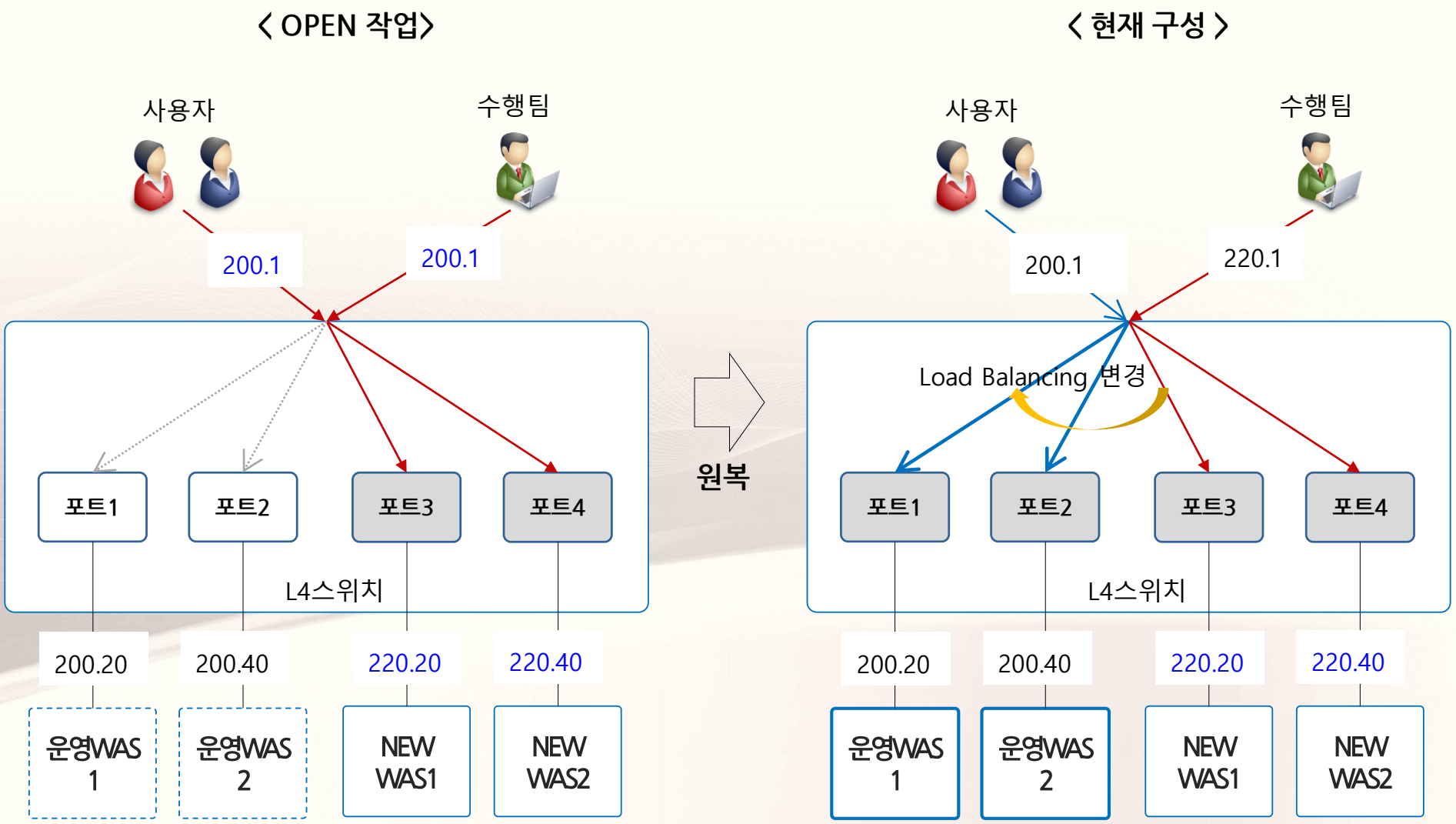


L4스위치 기반 MSA 신규 서비스 오픈 전략



마이크로서비스아키텍처 기반의 HIS 고도화 전환 사례, 건국대학교병원 이제관 기술사

오픈 장애발생시, 원복 시나리오 대응 전략 준비



HIS 마이크로 서비스 아키텍처 모니터링 구성

- KHAN APM 기반의 28개 WAS 모니터링

HIS에서의 MSA 적용결과 Lessons Learned

- MSA 구성체계로 인한 의료정보팀 개발업무 변화 결과
  - MSA 시스템의 구조 장점은?
    - 업무별 사용량 명확, 부하가 심한 업무를 시간대별 모니터링 후 해당 서비스 증설
    - 업무별 영향없이 쾌적한 반영/배포 환경 : 해당 업무의 Branch를 통한 반영 및 적용
    - 빠른 업무별 MSA 미들웨어 구성기반의 부팅으로 인한 배포 공포의 해방
      ✓ (1분 40초 운영적용, 현재 필요시 UI기반 수시적용)
    - 성능 속도, 분석, 성능, 증설, 반영, 안정, WAS확장, 이기종 WAS/FW가능, 개별로그분석, 세션공유
    - 기존 업무에서 계륵과 같았던 EJB 기반의 서비스가 MSA 구성에서는 보다 명확한 전환에 기여함
  - MSA 유지관리에 설정 원칙은?
    - 모든 업무의 MSA의 전환 여부는 비즈니스 업무에 따라 효율성과 생산성, 유지보수성이 결정됨
    - 기존 서비스 업무의 의존성과 DB의 타 업무 의존성 관계를 지속 개발과정에서의 아키텍처 모니터링 필요
    - 신규 시스템은 완전한 MSA 아키텍처 방식 권장 유도
    - 적정기술 채택의 중요성: 일방적 MSA 구성이 아닌 업무적 필요도에 따라 기존 기술인 EAI등의 채널연계, DB JOIN 방식은 여전히 강력하고 효율적인 기술임

- Thanks to RedHat
  - 전세계 EAP 7.X 그룹, 건국대학교병원 요구사항, 기술 선패치 제공 → 공식 정규패치 후반영
  - 막대한 기술요구사항 분석 및 전환에 성심 지원