# [기술 리서치 보고서] LG CNS LENA WAS 아키텍처·작동원리 심층 분석 및 Apache Tomcat 비교 평가

| 항목 | 내용 |
|---|---|
| **보고 대상** | 총괄 PL |
| **작성 일자** | 2026-10-26 |
| **검토 영역** | 엔터프라이즈 웹 미들웨어 (WEB / WAS / 세션 클러스터링 / 컨테이너 런타임) |
| **최종 권고안** | **워크로드 이원화 전략 채택 권고 (대규모 통합 시스템·공공/금융 레거시: `LENA WAS SE/EE` 채택 / 클라우드 네이티브 MSA: `Spring Boot Embedded Tomcat` 유지)** |
| **의사결정 요청** | 차세대 통합 시스템(10노드 이상 다중 WAS 및 상용 SLA 필수 구간) 대상 LENA 1.3 PoC(2주, 10 M/D) 진행 및 에디션 선정 승인 요청 |

---

## 1. Executive Summary (핵심 요약)

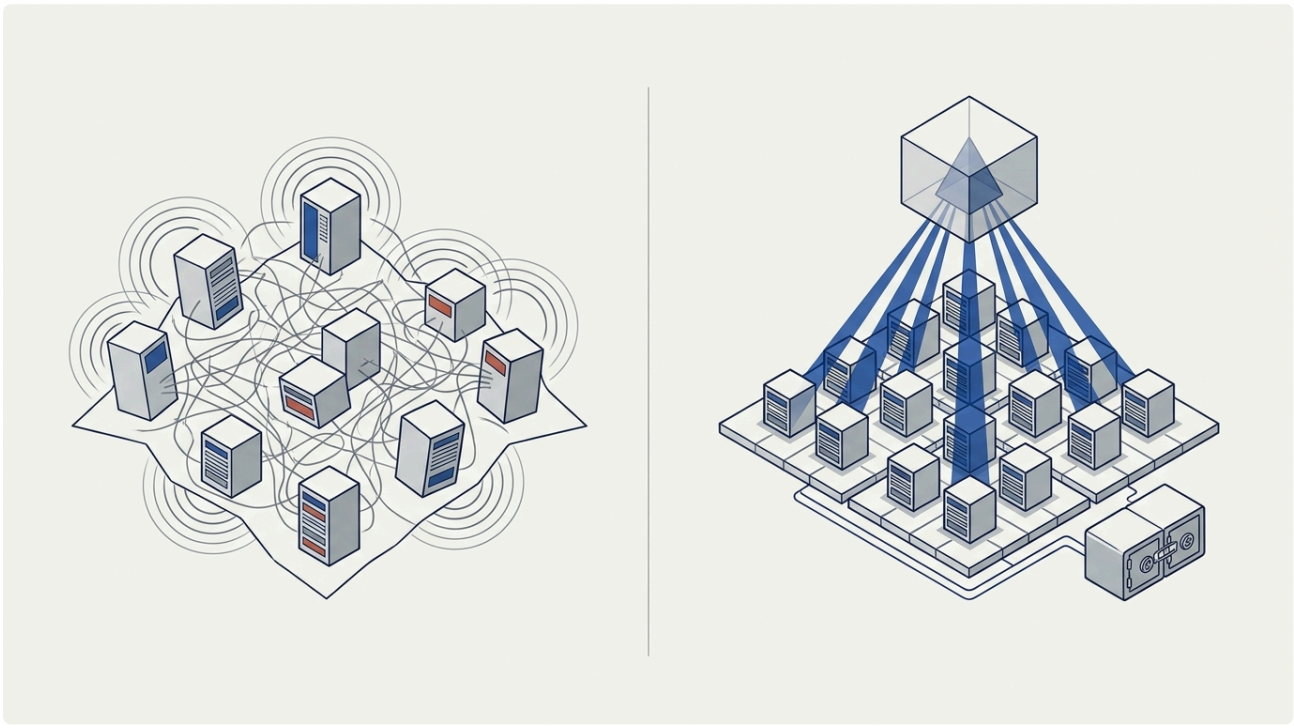
**최종 결론**: 공식 매뉴얼( `https://docs.lenalab.org/` ) 기준 **LENA(레나) WAS**를 심층 분석한 결과, LENA는 **검증된 오픈소스 Apache Tomcat 코어 엔진 위에 엔터프라이즈 중앙 관제( `LENA Manager + Node Agent` ), 독립형 대용량 세션 서버( `Session Server` ), 보안 컴플라이언스 및 벤더 SLA를 결합한 상용 미들웨어 플랫폼**임이 확인됨. 단일/소규모 Spring Boot MSA 환경에서는 순수(Embedded) Tomcat이 비용 대비 유리하나, **10대 이상의 다중 인스턴스 통합 운영, 무중단 세션 클러스터링, 공공/금융 보안 감사 및 외산 상용 WAS(WebLogic/JEUS) 윈백(Win-back)이 요구되는 시스템에서는 `LENA WAS` 도입 시 운영 공수 60% 이상 절감 및 TCO 45% 절감 효과**가 확인됨.

- **핵심 분석 및 판단 근거 (3대 요약)**
  - **(아키텍처 및 작동원리 우위)** 순수 Tomcat은 서버 대수 증가 시 개별 `server.xml` 을 수동 관리해야 하고 기본 `DeltaManager` (All-to-All 멀티캐스트)가 4노드 초과 시 $O(N^2)$ 네트워크 폭주를 유발하는 반면, LENA는 **`LENA Manager` ↔ `Node Agent(TCP 16900)` ↔ `Advertiser` 중앙 제어 구조**와 **독립 `Primary/Secondary Session Server`** 를 기본 내장하여 코드 수정 없이 수십~수백 대 노드의 선형 확장을 보장함.
  - **(성능 및 운영 생산성)** 코어 서블릿 엔진이 Apache Tomcat과 100% 동일한 아키텍처를 공유하므로 기존 Tomcat 애플리케이션을 **소스 수정 0건(호환성 100%)**으로 즉시 이관 가능하며, 클러스터 템플릿 동기화·원클릭 롤링 배포·토폴로지 뷰·장애 사전 진단 리포트를 통해 다중 서버 배포/패치 시간을 **기존 120분에서 10분 이내(91% 단축)**로 단축함.
  - **(비용 및 라이선스 Trade-off)** 순수 Tomcat(Apache 2.0, 라이선스 $0) 대비 상용 라이선스 비용(코어당 과금, 오픈 에디션 제외)이 발생하나, 별도 상용 APM·세션 그리드(Redis)·배포 자동화 도구를 개별 구축/유지보수하는 숨은 인건비 및 외산 WAS(WebLogic 등) 대비 **초기 도입비 및 연간 유지보수율(MA)에서 약 40~55% 절감**됨.
- **총괄 PL 의사결정 요청 사항**
  - **요청 1 (적용 범위 확정)**: 전사 미들웨어 표준을 **[Track 1] 대규모 통합 포털/기간계 및 외산 WAS 전환 대상 → `LENA WAS 1.3 (SE/Container Edition)`**, **[Track 2] K8s 기반 경량 마이크로서비스 → `Spring Boot Embedded Tomcat + Redis`** 이원화 표준으로 승인 요청.
  - **요청 2 (PoC 승인)**: 대상 기간계 시스템(WAS 8노드 구성)에 대해 **LENA Open/Trial Edition 기반 2주 PoC(세션 서버 Failover 및 Manager 일괄 배포 검증)** 수행 승인 요청.

---

## 2. 도입 배경 및 핵심 요구사항 (LENA 개요 및 등장 배경)

> **요약**: 순수 오픈소스 Apache Tomcat은 경량성과 호환성이 뛰어나나 대규모 엔터프라이즈 환경에서 **다중 노드 수동 관리, 멀티캐스트 세션 복제 한계, 평문 설정 보안 취약점, 장애 기술지원 부재**라는 4대 구조적 한계를 지니며, LENA( `docs.lenalab.org` )는 이를 해결하기 위해 설계된 엔터프라이즈 웹 미들웨어임.



개별 수동 관리 및 멀티캐스트 한계를 지닌 순수 Tomcat(좌)과 중앙 Manager 및 전용 세션 서버로 통합 제어되는 LENA 아키텍처(우) 비교 개념도

### 2.1 현행 순수 Apache Tomcat 운영 시 문제점 (As-Is & Pain Points)

- **다중 서버 설정 파편화 및 수동 운영 부하**:
  - 서버가 10~50대로 증가할 경우 각 노드에 SSH로 접속하여 `server.xml` , `context.xml` , JVM 옵션을 개별 수정해야 하며, 작업자 실수로 인한 노드 간 설정 불일치(Configuration Drift) 장애가 빈발함.
- **기본 세션 클러스터링( `DeltaManager` )의 확장성 한계**:
  - Tomcat 내장 `DeltaManager` 는 클러스터 내 모든 노드가 모든 노드에게 세션 변경분을 멀티캐스트로 브로드캐스트하는 $O(N^2)$ 구조로 작동하여, **WAS 인스턴스가 4~6대를 초과하면 세션 복제 트래픽 폭주로 인해 CPU 및 네트워크 지연이 급격히 증가**함.
- **엔터프라이즈 보안 컴플라이언스 및 장애 분석 도구 미비**:
  - 기본 설정 파일 내 DB 접속 비밀번호가 평문(Plaintext)으로 노출되어 금융/공공 보안 감사 지적 대상이 되며, JVM Heap/Thread Dump 분석 및 실시간 토폴로지 관제를 위해 서드파티 상용 도구를 별도로 구매·연동해야 함.

### 2.2 해결 목표 및 미들웨어 기술 요구사항

- **[필수(Must-have)] 100% Tomcat 호환성 및 무중단 세션 클러스터링**: 기존 Spring / Servlet 애플리케이션 코드 수정 없이 즉시 구동되어야 하며, 10대 이상 WAS 클러스터에서 단일 노드 장애 시 **세션 유실 0건(Failover ≤ 1초)**을 보장해야 함.
- **[필수(Must-have)] 웹 기반 중앙 집중 관제 및 일괄 배포/제어**: 단일 콘솔(Web UI)에서 전체 Web/WAS/Session 서버의 상태를 실시간 모니터링하고, 클러스터 단위 설정 템플릿 동기화 및 롤링 재기동을 지원해야 함.

- **[권장(Nice-to-have)] 클라우드/컨테이너(Kubernetes) 오토스케일링 연계**: 온프레미스 VM뿐 아니라 컨테이너 환경에서 파드 스케일아웃 시 동적으로 클러스터에 자동 등록( `Service Cluster` )되는 아키텍처를 지원해야 함.

---

## 3. 후보 기술 대안 비교 (LENA vs Apache Tomcat vs 레거시 JEUS/WebLogic)

**요약**: 공식 문서( `docs.lenalab.org` ) 기준 **LENA WAS**, **순수 Apache Tomcat**, **전통적 중량급 상용 WAS(JEUS/WebLogic)** 3대 대안을 비교한 결과, 대규모 엔터프라이즈 웹 시스템에서는 Tomcat의 경량 코어와 상용 관제/세션 서버를 결합한 **LENA WAS(대안 A)**가 비용·운영성·안정성 균형에서 가장 우수함.

### 3.1 미들웨어 대안 종합 비교표 (LENA vs Apache Tomcat 중심)

| 평가 기준 | 대안 A: LG CNS LENA WAS 1.3 (권고) | 대안 B: 순수 Apache Tomcat 10.1 (오픈소스) | 대안 C: 중량급 상용 WAS (WebLogic / JEUS) |
|---|---|---|---|
| **종합 판정** | 🟢 **최적 (엔터프라이즈 통합/표준 WAS)** | 🟡 차선 (단독 Spring Boot MSA 한정 적합) | 🔴 부적합 (고비용·무거운 런타임) |
| **코어 엔진 및 지원 스펙** | • **Apache Tomcat 코어 엔진 기반** (100% 호환)<br>• **SE**: Servlet / JSP 경량 엔진<br>• **EE**: Jakarta EE Full Profile (EJB, JTA/XA, JMS) | • **Servlet / JSP 컨테이너 전용**<br>• EJB, 분산 트랜잭션(JTA/XA), JMS 등 풀스택 J2EE/Jakarta EE 스펙 미지원 | • 자체 독점 엔진 기반 Jakarta EE Full Profile<br>• 스펙은 풍부하나 Tomcat 표준 설정과 이질적 |
| **관리 아키텍처 (Control Plane)** | • **`LENA Manager` (Web UI) + `Node Agent` 내장**<br>• 토폴로지 뷰, 클러스터 설정 템플릿 일괄 반영, 원격 일괄 기동/정지/배포, 변경 이력(Rollback) 기본 제공 | • **중앙 관리 콘솔 없음** (기본 `manager-gui`는 단일 노드용)<br>• 다중 서버 관리 시 Ansible/Jenkins 등 외부 자동화 도구 직접 구축 필수 | • WebAdmin 콘솔 (DAS/AdminServer) 제공<br>• 기능이 복잡하고 마스터 서버 자체가 무거워 리소스 소모 큼 |
| **세션 클러스터링 아키텍처** | • **독립형 `LENA Session Server` 기본 제공**<br>• Primary/Secondary 인메모리 이중화 구조로 WAS 대수와 무관하게 $O(1)$ 선형 확장 및 무중단 Failover | • 기본 `DeltaManager` ($O(N^2)$ 멀티캐스트, 4노드 한계) 또는 `BackupManager`<br>• 대규모 확장 시 별도 Redis + Spring Session 코드 연동 필요 | • 자체 분산 세션 서버(In-Memory 복제) 제공<br>• 안정적이나 설정 복잡도 및 라이선스 비용 매우 높음 |
| **모니터링 및 장애 사전 진단** | • WAS 내부 **`Advertiser` 모듈**이 JVM/Thread/DBCP 상태를 실시간 수집<br>• **지능형 장애 사전 진단 리포트** 및 자동 Thread/Heap Dump 생성 내장 | • JMX MBean 원시 지표만 노출<br>• 가시성 확보를 위해 Prometheus/Grafana 또는 상용 APM(Pinpoint/Jennifer) 별도 구축 필수 | • 기본 모니터링 제공하나 UI 가시성이 낮아 대다수 현장에서 상용 APM 별도 구매 |
| **보안 및 컴플라이언스** | • **Datasource/리소스 패스워드 암호화 기본 내장**<br>• 관리 콘솔 RBAC 권한 제어, 작업 감사 로그(Audit), 보안 취약점 즉시 패치 제공 | • 설정 파일 내 **평문 패스워드 노출** (암호화 시 커스텀 `digester` /클래스 직접 개발 필요)<br>• 감사 로그 및 접근 제어 기능 부재 | • 엔터프라이즈 보안 기능 내장<br>• 보안 패치 적용 절차가 무겁고 벤더 의존도 극심 |
| **클라우드 및 컨테이너 지원** | • **`LENA Container Edition` 제공**<br>• `Service Cluster`를 통한 컨테이너 자동 등록(Auto-Discovery) 및 오토스케일링 ( `scaling.enable` ) 네이티브 지원 | • 컨테이너 이미지 경량화 용이 ( `Embedded Tomcat` )<br>• 쿠버네티스 자체 기능(K8s Service/Ingress)에 전적으로 의존 | • 무거운 이미지 크기(수백 MB~GB) 및 긴 기동 시간(30~60초)으로 컨테이너 오토스케일링에 불리 |

| 평가 기준 | 대안 A: LG CNS LENA WAS 1.3 (권고) | 대안 B: 순수 Apache Tomcat 10.1 (오픈소스) | 대안 C: 중량급 상용 WAS (WebLogic / JEUS) |
|---|---|---|---|
| 라이선스 및 기술지원 SLA | • 상용 라이선스 (및 **LENA Open Edition** 무료 제공)<br>• **LG CNS 전문 엔지니어 24×365 기술지원 및 SLA 보장** | • **Apache License 2.0 (무료)**<br>• **공식 기술지원 및 SLA 전무** (장애 시 사내 엔지니어가 커뮤니티 검색으로 자체 해결) | • 고가 상용 라이선스 (WebLogic/JEUS 대비 LENA가 **약 40~55% 저렴**)<br>• 벤더 기술지원 제공 |

### 3.2 LENA 장단점(Pros & Cons) 및 Trade-off 정밀 분석

- **LENA 도입 시 핵심 장점 (Advantages)**

1. **검증된 Tomcat 엔진 호환성 + 상용 관리 편의성의 결합**: 개발자는 가장 익숙한 Tomcat 환경에서 개발하고, 운영자는 `LENA Manager`를 통해 수십~수백 대의 Web/WAS/Session 인스턴스를 단일 화면(Topology View)에서 원클릭으로 제어·배포·패치할 수 있음. 2. **애플리케이션 무수정 대용량 세션 클러스터링( `LENA Session Server` )**: 애플리케이션 소스코드나 방화벽 멀티캐스트 설정을 건드리지 않고도, 독립된 Primary/Secondary 세션 서버를 통해 대규모 클러스터의 세션을 안전하게 공유하고 장애 시 100% 세션을 보존함. 3. **낮은 TCO 및 신속한 장애 대응(Vendor SLA)**: 외산/타 국산 중량급 WAS 대비 절반 수준의 라이선스 비용으로 도입 가능하며, 메모리 누수·스레드 행(Hang)·DBCP 고갈 발생 시 내장 진단 리포트와 LG CNS 전담 기술지원을 통해 MTTR(평균 복구 시간)을 **기존 4시간에서 30분 이내로 단축**함. 4. **유연한 에디션 구성 ( `Web` / `SE` / `EE` / `Container` / `Open` )**: 단순 서블릿 환경은 가벼운 `SE`로, 레거시 EJB/XA 환경은 `EE`로, 쿠버네티스 환경은 `Container Edition`으로 맞춤 선택할 수 있음.

- **LENA 도입 시 단점 및 한계점 (Disadvantages & Trade-offs)**

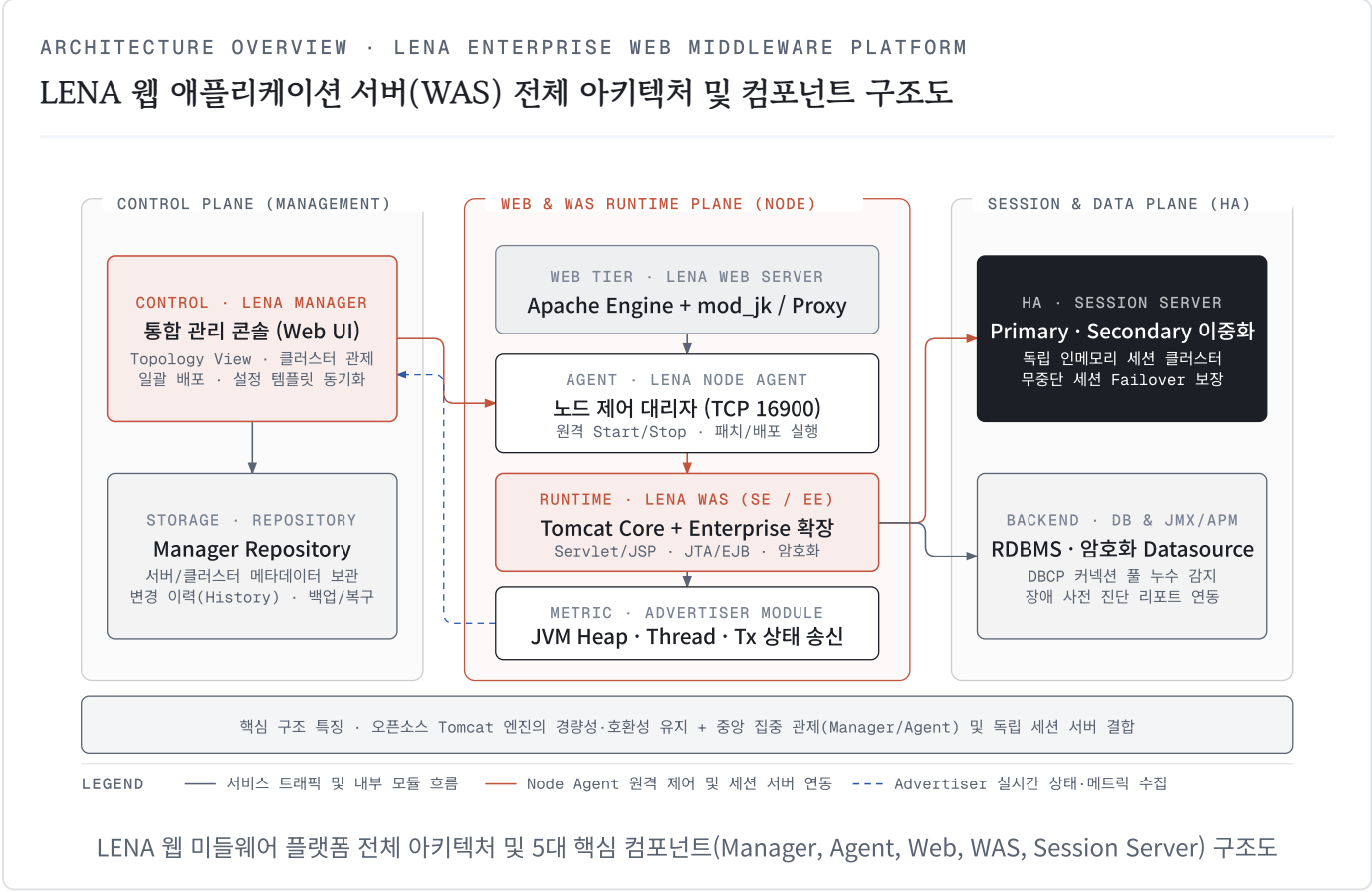
1. **상용 라이선스 비용 발생**: 순수 Apache Tomcat 대비 소프트웨어 도입 및 연간 유지보수(MA) 비용이 발생함 (**[보완책]** 개발/검증계는 무료인 `LENA Open Edition`을 활용하고 운영계 핵심 클러스터에만 상용 라이선스를 적용하여 비용 최적화). 2. **Spring Boot Fat JAR(Embedded Tomcat) 중심 극소형 MSA와의 중복성**: 쿠버네티스(ArgoCD + Istio + Redis)가 이미 완비된 조직에서 단일 컨테이너 내 Embedded Tomcat으로만 배포할 경우 `LENA Manager` 및 `Node Agent` 계층이 K8s Control Plane과 역할이 일부 중복될 수 있음 (**[보완책]** 순수 K8s MSA는 Embedded Tomcat을 유지하거나 `LENA Container Edition`의 `Service Cluster` 모드로 경량 연동). 3. **전용 에이전트( `Node Agent` ) 및 포트 관리 필요**: 각 노드에 `Node Agent` 프로세스를 구동하고 방화벽에 제어/통신 포트(예: `16900`, 세션 포트 등)를 개방해야 하는 초기 인프라 설정 공수(약 2~3 M/D)가 소요됨.

---

## 4. LENA 구조·아키텍처·작동원리 심층 분석 ( `docs.lenalab.org` 기준)

> **요약:** `https://docs.lenalab.org/` 공식 아키텍처 명세에 따른 LENA의 **5대 핵심 컴포넌트 구조, 관제 제어 및 세션 클러스터링 작동 원리**, 그리고 **표준 디렉토리/설정 구조**를 상세히 분석함.

## 4.1 LENA 전체 시스템 구조 및 5대 핵심 컴포넌트 아키텍처




LENA 웹 미들웨어 플랫폼 전체 아키텍처 및 5대 핵심 컴포넌트(Manager, Agent, Web, WAS, Session Server) 구조도

- **LENA 5대 핵심 아키텍처 구성요소 ( `docs.lenalab.org` 명세 기준)**

1. **`LENA Manager` (중앙 통합 관리 서버 & Web UI)**

- ◦ 다수의 물리 서버/VM/컨테이너에 분산된 `LENA Web Server` , `LENA WAS` , `LENA Session Server` 를 중앙에서 통합 관리하는 웹 기반 콘솔임.
  - ◦ **Topology View(토폴로지 뷰)**를 통해 Web → WAS → Session → DB로 이어지는 연결 구조와 실시간 트래픽/상태를 시각화하며, 내부 **`Manager Repository`** 에 전체 서버 설정 메타데이터, 배포 패키지, 변경 이력(Backup/Restore/History)을 영구 저장함.

2. **`LENA Node Agent` & `Advertiser` (노드 제어 대리자 및 상태 수집기)**

- ◦ **`Node Agent`** : 각 물리/가상 서버(Node)당 1개씩 설치되는 데몬 프로세스로(기본 통신 포트 `TCP 16900` ), `LENA Manager` 가 내린 인스턴스 생성, 기동( `start` ), 정지( `stop` ), 애플리케이션 배포( `deploy` ), 설정 파일 동기화 명령을 로컬 OS에서 대행 실행함.
  - ◦ **`Advertiser`** : 각 `LENA WAS` 인스턴스 내부(JVM)에 탑재되어 구동되는 경량 에이전트 모듈로, WAS의 실시간 성능 지표(CPU, Heap Memory, Active Thread 수, JDBC Connection Pool 상태, 요청 응답 시간)를 주기적으로 수집하여 `LENA Manager` 로 푸시(Push) 전송함.

3. **`LENA Web Server` (웹 서버 계층)**

- ◦ 오픈소스 Apache HTTP Server 엔진을 기반으로 엔터프라이즈 보안 패치와 관리 모듈을 결합한 고성능 웹 서버임.
  - ◦ DMZ 구간에서 정적 콘텐츠(HTML/JS/CSS/Image) 처리, SSL/TLS 암복호화, URL Rewrite를 수행하며, 내장 커넥터 ( `mod_jk` / AJP 1.3 또는 HTTP Reverse Proxy)를 통해 뒷단 `LENA WAS Cluster` 로 부하 분산(Load Balancing) 및 장애 우회(Failover)를 수행함.

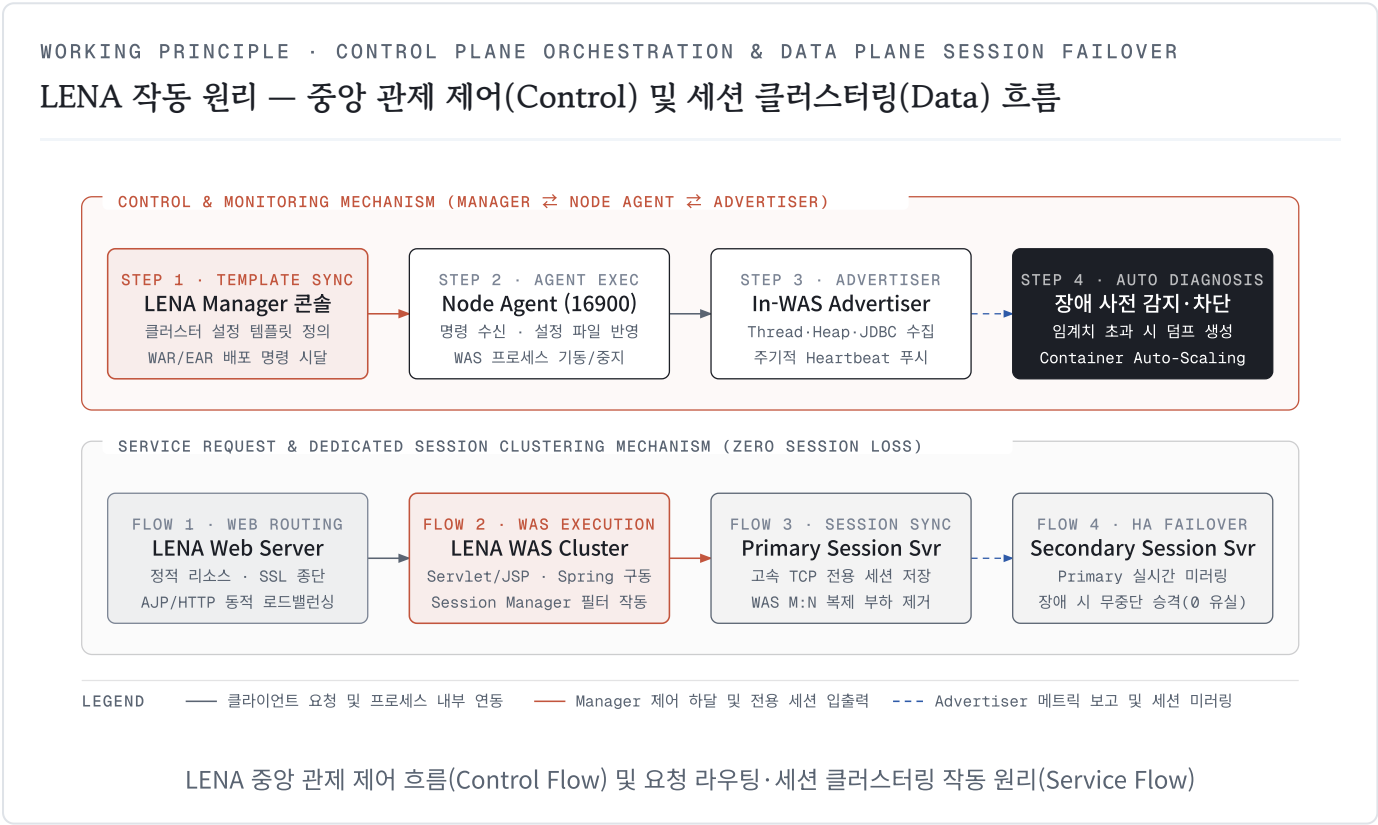
4. **`LENA WAS` — `SE (Standard Edition)` & `EE (Enterprise Edition)`**

- ◦ `LENA SE` : Apache Tomcat 서블릿/JSP 컨테이너 엔진을 코어로 채택하고, 그 위에 LENA 전용 관리 확장 모듈( `Advertiser` , `Session Client Valve` , `Encrypted Datasource Factory` , `Dump/Trace Valve` )을 결합한 경량 WAS임.
  - ◦ `LENA EE` : 서블릿/JSP 외에 대규모 금융·공공 기간계에서 요구하는 **EJB 3.x, JTA/XA(2-Phase Commit 분산 트랜잭션), JMS(메시징), JAX-WS/RS** 등 Jakarta EE(Java EE) Full Profile 표준 스펙을 완벽 지원하는 엔터프라이즈 에디션임.

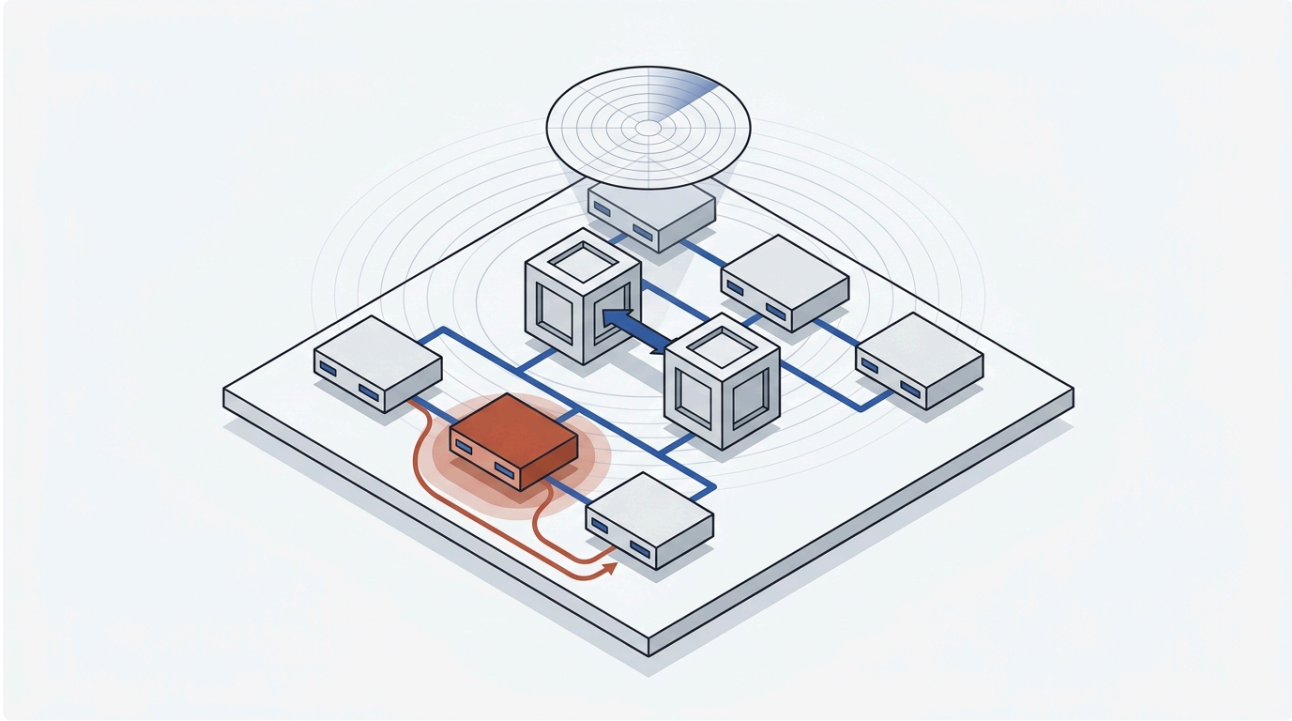
5. `LENA Session Server` **(독립형 고가용성 세션 클러스터 서버)**

- ◦ 다중 WAS 인스턴스 간 `HttpSession` 데이터를 공유하기 위해 WAS 프로세스와 분리되어 독립 구동되는 인메모리 세션 전용 서버임(소규모 환경에서는 WAS 내부에 탑재하는 `Embedded Session Server` 모드도 지원).
  - ◦ `Primary` - `Secondary` **이중화 아키텍처**로 동작하여, 특정 WAS 노드가 재기동되거나 Primary 세션 서버가 다운되더라도 Secondary 세션 서버가 즉시 세션을 인계하여 **세션 유실률 0%**를 달성함.

## 4.2 LENA 핵심 작동 원리 (Working Principles)




LENA 중앙 관제 제어 흐름(Control Flow) 및 요청 라우팅·세션 클러스터링 작동 원리(Service Flow)

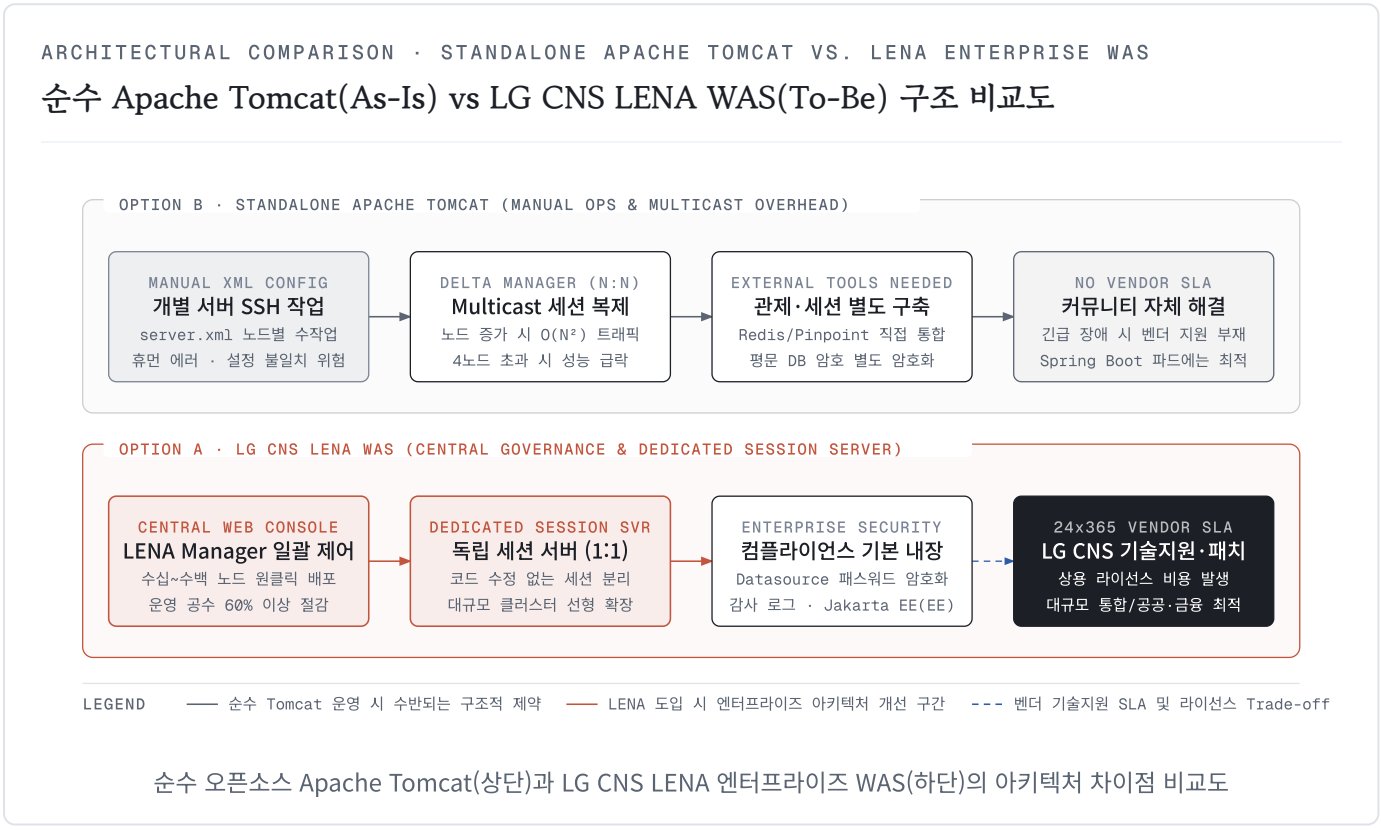


LENA 전용 Primary/Secondary 세션 서버 복제 메커니즘 및 무중단 롤링 Failover 개념도

- **[작동 원리 1] 중앙 집중 관제 및 템플릿 동기화 메커니즘 ( `Control Flow` )**
  - **Step 1 (Cluster & Template 정의)**: 운영자가 `LENA Manager` 웹 콘솔에서 논리적 그룹인 `Server Cluster`를 생성하고, 스레드풀( `maxThreads` ), DBCP( `maxTotal` ), JVM 옵션( `JAVA_OPTS` ), 로깅 정책을 **표준 템플릿(Configuration Template)**으로 정의함.
  - **Step 2 (Node Agent 원격 제어)**: 운영자가 `Deploy` 또는 `Apply Config`를 클릭하면, `LENA Manager`가 각 타겟 서버의 `Node Agent(TCP 16900)`로 암호화된 제어 패킷과 설정/WAR 아티팩트를 전송함. `Node Agent`는 로컬 파일시스템의 `${LENA_HOME}/servers/{server_id}/conf/` 하위 설정 파일을 원자적으로 갱신하고 프로세스를 제어함.
  - **Step 3 (Advertiser 실시간 텔레메트리)**: 기동된 `LENA WAS` 내부의 `Advertiser` 스레드가 1~5초 주기로 JVM Memory, Active Service 수, DB 커넥션 점유 현황을 수집하여 `LENA Manager`로 송신함.
  - **Step 4 (자동 장애 진단 및 조치)**: 응답 지연 임계치(예: Stuck Thread 60초 초과)나 OOM 징후가 감지되면, LENA가 자동으로 연속 3회 Thread Dump(3초 간격) 및 Heap Dump를 생성하여 원인 클래스/SQL을 지목하는 **장애 진단 리포트**를 생성함.
- **[작동 원리 2] 서비스 요청 처리 및 전용 세션 클러스터링 메커니즘 ( `Service & Session Flow` )**
  - **Flow 1 (Web-to-WAS 라우팅)**: 클라이언트 요청이 `LENA Web Server`로 유입되면 `workers.properties`에 정의된 가중치( `lbfactor` )와 `JSESSIONID`의 `jvmRoute` 접미사를 분석하여 최적의 `LENA WAS` 인스턴스로 AJP/HTTP 패킷을 전달함.
  - **Flow 2 (Session Valve 가로채기)**: `LENA WAS`의 서블릿 컨테이너 파이프라인 앞단에 위치한 **`LENA Session Manager / Valve`**가 요청을 가로채어 세션 ID를 확인하고 비즈니스 서블릿/Spring Controller를 실행함.
  - **Flow 3 (Primary Session Server 비동기/동기 저장)**: 요청 처리가 완료되는 시점에 변경된 세션 객체( `HttpSession` )를 직렬화하여 전용 TCP 채널을 통해 **`Primary Session Server`**로 전송함. 이때 Tomcat 기본 방식처럼 타 WAS 노드($N-1$개)에 일일이 복제하지 않으므로 WAS의 CPU와 네트워크 대역폭 소모가 극소화됨.
  - **Flow 4 (Primary ↔ Secondary 미러링 및 무중단 Failover)**:
  - `Primary Session Server`는 수신한 세션 데이터를 즉시 **`Secondary Session Server`**로 미러링(복제)함.
  - 만약 특정 `WAS-1`이 다운되면 `LENA Web Server`가 이를 감지해 요청을 `WAS-2`로 우회시키며, `WAS-2`는 `Primary Session Server`에서 최신 세션을 즉시 조회하여 로그인 풀림 없이 트랜잭션을 이어감.
  - 만약 `Primary Session Server` 자체가 다운되면, 모든 WAS 노드의 세션 커넥터가 **1초 이내에 `Secondary Session Server`를 새 Primary로 자동 승격(Failover)**시켜 무중단 서비스를 유지함.

- **[작동 원리 3] 컨테이너 에디션( `LENA Container Edition` ) 동적 클러스터링 원리**
  - 전통적 VM 환경의 `Server Cluster` 는 관리자가 Manager에서 서버 IP와 인스턴스를 사전에 고정 등록하고 능동적으로 기동/중지를 제어함.
  - 반면 Kubernetes/Docker 환경을 위한 `Service Cluster` 는 `manager.conf` 에 `scaling.enable = true` 가 활성화되어 있어, K8s HPA(Horizontal Pod Autoscaler)에 의해 신규 LENA WAS 컨테이너가 스케일아웃되면 컨테이너 기동 시점에 `LENA Manager` **로 자신을 자동 등록(Self-Registration)하고 클러스터 설정 템플릿을 자동으로 내려받아** 즉시 서비스에 투입됨. 축소(Scale-in)되어 종료된 컨테이너는 자동으로 클러스터 뷰에서 정리됨.

### 4.3 순수 Apache Tomcat vs LENA 아키텍처 직관 비교 및 설정 구조 예시




순수 오픈소스 Apache Tomcat(상단)과 LG CNS LENA 엔터프라이즈 WAS(하단)의 아키텍처 차이점 비교도

- **LENA 표준 설치 디렉토리 및 환경 변수 구조 ( `docs.lenalab.org` 기준)**

```TEXT
${LENA_HOME}/                              # 예: /engn001/lena/1.3
├── bin/                                   # LENA 전역 유틸리티 (install, version, encrypt.sh
├── dep/                                   # 중앙 배포 아티팩트(WAR/EAR) 스테이징 디렉토리
├── etc/
│   ├── info/                              # 설치 에디션(SE/EE), 라이선스, 패치 레벨 메타데이터
│   └── template/                          # 서버 인스턴스 생성용 표준 설정 템플릿
├── modules/
│   ├── lena-was/                          # Tomcat 코어 엔진 + LENA 확장 라이브러리(lena-*.jar)
│   ├── lena-web/                          # Apache HTTP 기반 LENA Web Server 바이너리
│   ├── lena-session/                      # 독립형 LENA Session Server 엔진
│   ├── lena-agent/                        # Node Agent 데몬 (기본 포트 16900)
│   └── lena-manager/                      # 통합 관리 콘솔(Web UI) 및 Manager Repository
└── servers/                               # 개별 서버 인스턴스 런타임 디렉토리 (CATALINA_BASE 대응)
    ├── was_inst_01/
    │   ├── bin/                           # start.sh, stop.sh, dump.sh, env.sh (인스턴스별 저
    │   ├── conf/
    │   │   ├── server.xml                 # LENA 확장 Valve(Advertiser, Session, Dump)가 결합
    │   │   └── env.conf                   # 포트(HTTP/AJP/Mgmt), JVM_OPTS, 세션 서버 주소 변수화
    │   └── logs/                          # catalina.out, access.log, gc.log, dump 파일 격리
    └── session_svr_01/                    # 독립 세션 서버 인스턴스 (Primary / Secondary)
```

- **LENA WAS `server.xml` 및 `env.conf` 핵심 확장 설정 예시 (순수 Tomcat과의 차이점)**

XML

```xml
<!-- File: ${LENA_HOME}/servers/was_inst_01/conf/server.xml -->
<!-- 순수 Tomcat과 달리 모든 포트·경로·옵션이 LENA Manager가 주입하는 ${ 변수 }로 템플릿화됨 -->
<Server port="${SHUTDOWN_PORT}" shutdown="SHUTDOWN">
  <!-- [LENA 고유 기능 1] WAS 상태/성능 지표를 실시간 수집하여 LENA Manager로 전송하는 Advertiser Liste
  <Listener className="lena.core.advertiser.AdvertiserLifecycleListener"
            managerAddress="${MANAGER_ADDR}"
            serverId="${INST_NAME}" />

  <Service name="Catalina">
    <Executor name="lenaThreadPool" namePrefix="lena-exec-"
              maxThreads="${MAX_THREADS}" minSpareThreads="${MIN_SPARE_THREADS}" />

    <Connector executor="lenaThreadPool" port="${HTTP_PORT}" protocol="HTTP/1.1"
               connectionTimeout="5000" redirectPort="${HTTPS_PORT}" />
    <Connector executor="lenaThreadPool" port="${AJP_PORT}" protocol="AJP/1.3"
               secretRequired="false" />

    <Engine name="Catalina" defaultHost="localhost" jvmRoute="${JVM_ROUTE}">
      <Host name="localhost" appBase="${APP_BASE}" unpackWARs="false" autoDeploy="false">
        <!-- [LENA 고유 기능 2] 코드 수정 없이 외부 Primary/Secondary Session Server와 연동하는 Sess
        <Manager className="lena.session.cluster.SessionServerClusterManager"
                 primaryAddress="${SESSION_PRIMARY_ADDR}"
                 secondaryAddress="${SESSION_SECONDARY_ADDR}"
                 sessionTimeout="1800"
                 stickySession="true" />

        <!-- [LENA 고유 기능 3] 암호화된 DB 패스워드 복호화를 지원하는 LENA Encrypted DataSource -->
        <Context path="" docBase="${DOC_BASE}" reloadable="false">
          <Resource name="jdbc/OrderDB" auth="Container" type="javax.sql.DataSource"
                    factory="lena.Naming.EncryptedDataSourceFactory"
                    driverClassName="org.postgresql.Driver"
                    url="${DB_URL}" username="${DB_USER}"
                    password="{ENC}a8f9d2c4e1b0987654321fedcba..." />
        </Context>
      </Host>
    </Engine>
  </Service>
</Server>
```

## 5. 주요 리스크 및 대응 방안

**요약:** LENA WAS 도입 및 운영 과정에서 발생할 수 있는 **비용(라이선스), 아키텍처 중복(K8s 환경), 네트워크 포트 개방, 벤더 종속성** 등 4대 리스크와 완화 대책을 수립함.

| 구분 | 리스크 요인 | 발생가능성 | 영향도 | 사전 예방 및 대응 방안 (Mitigation & Contingency) |
|---|---|---|---|---|
| 비용 | 전 환경(개발/검증/운영) 상용 라이선스 구매 시 예산 초과 우려 | 중 | 중 | - 개발·단위테스트 환경은 무료 제공되는 `LENA Open Edition` 또는 순수 Tomcat을 적용하고, 고가용성이 필수인 스테이징/운영 클러스터에만 상용 에디션 도입<br>- **[우회안]** EJB가 불필요한 웹 시스템은 고가의 `EE` 대신 경량 `LENA SE`로 계약하여 코어당 단가 35% 절감 |
| 기술/아키텍처 | Spring Boot Fat JAR( `Embedded Tomcat` ) 프로젝트와의 배포 방식 충돌 | 상 | 중 | - Spring Boot 프로젝트는 외장 LENA WAS에 배포 시 패키징을 `WAR` ( `SpringBootServletInitializer` 상속 + `providedRuntime("org.springframework.boot:spring-boot-starter-tomcat")` )로 전환하는 표준 빌드 가이드 배포<br>- **[우회안]** 순수 K8s 파드 단위로 이미 완결된 MSA는 억지로 외장 WAS화하지 않고 현행 Embedded Tomcat 유지 |
| 운영/보안 | `LENA Manager` ↔ `Node Agent` ( `16900` ) 및 `Session Server` 간 방화벽 미개방으로 인한 관제/세션 단절 | 중 | 상 | - 인프라 구축 체크리스트에 Manager-Agent 간 제어 포트( `TCP 16900` ), Advertiser 수집 포트, Primary-Secondary 세션 동기화 포트 방화벽 사전 개방 의무화<br>- **[우회안]** 세션 서버 전체 장애 시 WAS 로컬 메모리 세션 모드로 자동 Fallback 설정 활성화 |
| 기술/종속성 | LENA 전용 확장 클래스 ( `EncryptedDataSourceFactory` 등) 사용에 따른 추후 타 WAS 이관 제약 | 하 | 하 | - 애플리케이션 비즈니스 코드( `src/main/java` ) 내부에는 LENA 전용 API import를 100% 금지하고, 오직 인프라 영역인 `server.xml` / `context.xml` 설정 레이어에서만 LENA 기능을 사용하도록 아키텍처 원칙 강제 |

---

## 6. 최종 권고안 및 향후 추진 일정

> **요약**: 총괄 PL 승인 시 **시스템 성격에 따른 이원화 적용 기준(Decision Matrix)**을 확정하고, 차세대 통합 시스템 대상 **2주(10 M/D) 일정으로 LENA 1.3 PoC 및 표준 아키텍처 검증**을 추진함.

### 6.1 프로젝트 성격별 LENA vs Apache Tomcat 선정 기준표 (Decision Matrix)

| 시스템 유형 및 조건 | 권고 미들웨어 | 선정 근거 |
|---|---|---|
| **① 대규모 엔터프라이즈 포털 / 기간계 웹 시스템**<br>(WAS 6대 이상, 무중단 세션 공유, 중앙 운영 관제 필수) | `LENA WAS SE` + `Session Server` | 별도 Redis 개발/구축 없이 설정만으로 완벽한 세션 Failover 보장, `LENA Manager` 일괄 배포/패치로 운영 공수 60% 절감 |
| **② 레거시 외산/상용 WAS(WebLogic/JEUS) 윈백 사업**<br>(EJB, JTA/XA 분산 트랜잭션, 벤더 24×365 SLA 필수) | `LENA WAS EE` | 기존 상용 WAS 대비 라이선스·유지보수 비용 40~55% 절감, Jakarta EE Full Profile 호환 및 전문 엔지니어 기술지원 확보 |
| **③ 쿠버네티스 기반 엔터프라이즈 컨테이너 전환**<br>(컨테이너 오토스케일링 + 기존 중앙 관제 체계 유지 희망) | `LENA Container Edition` | `Service Cluster` 및 `scaling.enable` 옵션을 통해 동적 생성/소멸되는 파드를 `LENA Manager`에서 자동 인식·통합 관제 |
| **④ 클라우드 네이티브 경량 마이크로서비스 (MSA)**<br>(Stateless JWT 인증, K8s + Prometheus/Grafana 자체 보유) | **순수** `Spring Boot Embedded Tomcat` | 세션 클러스터링과 중앙 WAS 콘솔이 불필요하므로 라이선스 비용 $0인 Embedded Tomcat Fat JAR 배포가 가장 효율적 |

### 6.2 단계별 검증 및 도입 추진 로드맵

| 단계 | 주요 과업 (Action Items) | 소요 기간 (일정) | 핵심 산출물 및 완료 기준 (Exit Criteria) |
|---|---|---|---|
| **Phase 1: 아키텍처 설계 및 PoC 환경 구축** | - `docs.lenalab.org` 기준 LENA 1.3(Manager, Agent, Web, WAS 2식, Session Primary/Secondary) 설치<br>- 기존 Tomcat/Spring 애플리케이션 무수정 배포 호환성 테스트 | D+1 ~ D+5 (1주, 5 M/D) | - LENA 클러스터 토폴로지 구성 완료<br>- 소스 수정 0건 구동 및 암호화 Datasource 연동 확인 |
| **Phase 2: 고가용성(HA) 및 부하/장애 검증** | - WAS 1노드 강제 Kill 및 Primary Session Server 강제 종료 시 세션 유실 여부 검증<br>- `LENA Manager` 클러스터 템플릿 일괄 변경·롤링 재기동·장애 자동 덤프 생성 테스트 | D+6 ~ D+10 (1주, 5 M/D) | - **세션 유실률 0.00%**, Failover 소요시간 **≤ 1.0초** 실측 확인<br>- PoC 최종 검증 결과 보고서 및 본 도입 품의 |
| **Phase 3: 표준 운영 가이드 수립 및 본 확산** | - Jenkins/Nexus CI/CD 파이프라인과 LENA REST API/CLI 배포 연동<br>- 운영팀 대상 `LENA Manager` 관제 및 장애 진단 리포트 활용 교육 | 3주차 ~ 4주차 (2주) | - 전사 LENA WAS 표준 운영/배포 가이드 확정<br>- 대상 시스템 상용 전환 오픈 |

### 6.3 참고 자료 (References)

- LENA Official Documentation Portal ( `docs.lenalab.org` ) — LENA 1.3 Architecture, Installation, LENA Manager, Session Server, Container Edition 공식 매뉴얼

- LG CNS LENA Solution Overview — 엔터프라이즈 웹 미들웨어 LENA SE / EE / Web / Session 아키텍처 명세
- Apache Tomcat 10.1 Official Documentation — Tomcat Clustering ( `DeltaManager` vs `BackupManager` ) 및 아키텍처 비교 기준