LMS Backend Architecture Guide

> **프로젝트명**: MZC 1st Project - 학습관리시스템
>
> **버전**: 1.0
>
> **아키텍처**: Docker 기반 컨테이너 인프라 (Nginx + Spring Boot + MySQL + Redis)
>
> **기술 스택**: Docker Compose, Nginx, Spring Boot 3.x, Java 21, MySQL 8.0, Redis 7
>
> **팀**: MZC Team 2

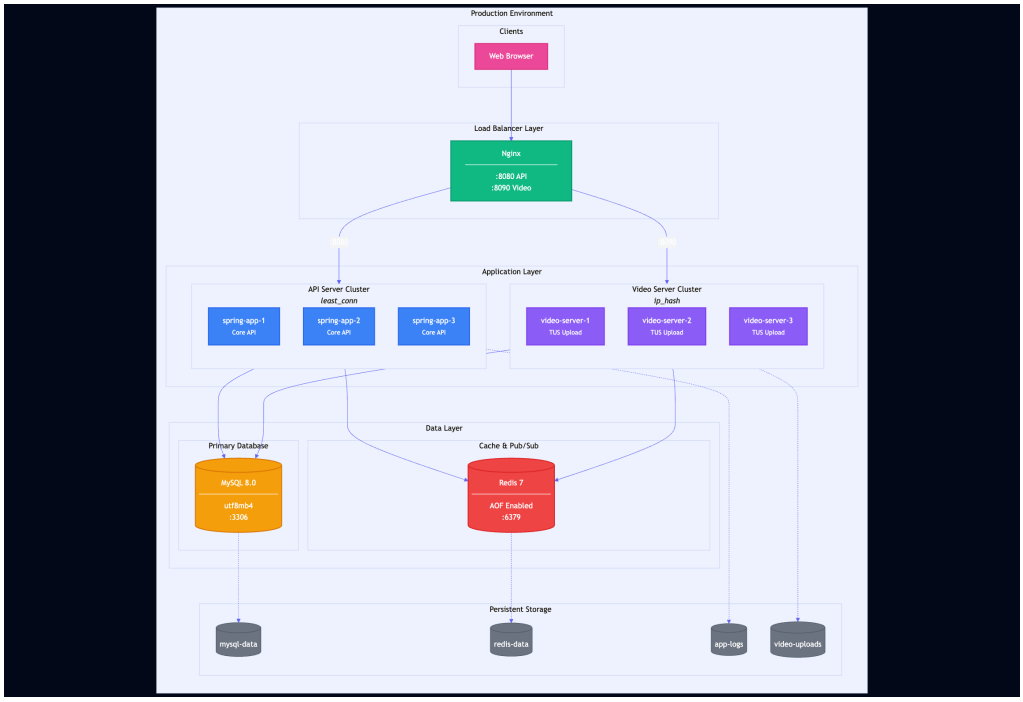
---

1. Overview

본 문서는 LMS(Learning Management System) 백엔드 시스템의 인프라 아키텍처를 설명합니다. 시스템은 Docker 기반 컨테이너 환경에서 운영되며, 환경별로 최적화된 구성을 제공합니다.

---

2. System Architecture Diagram



2.1 Architecture Overview

Layer	Components	Description
Client	Web Browser	사용자 인터페이스
Load Balancer	Nginx	트래픽 분산 및 라우팅
Application	Spring Boot × 3, Video Server × 3	비즈니스 로직 처리
Data	MySQL 8.0, Redis 7	데이터 저장 및 캐싱
Storage	Docker Volumes	영속 데이터 저장소

---

3. Environment Configuration

3.1 Environment Types

Environment	Compose File	Purpose	Characteristics
Local	docker-compose-local.yml	로컬 개발	인프라(DB/Cache)만 실행, IDE에서 앱 직접 실행

Environment	Compose File	Purpose	Characteristics
Development	docker-compose.yml	통합 테스트	전체 스택 단일 인스턴스 구성
Production	docker-compose.prod.yml	운영 환경	멀티 인스턴스 로드 밸런싱 구성

3.2 Service Port Mapping

Service	Local	Development	Production
Nginx	-	80(이미지만)	8080, 8090
Spring API	8080	8080	Internal
Video Server	8090	8090	Internal
MySQL	3306	3306	3306
Redis	6379	6379	6379
phpMyAdmin	8081	8081	-
Redis Commander	8082	8082	-

4. Container Specifications

4.1 Application Containers

Core API Server (Spring Boot)

Image: ddingsh9/mzc-core-api:latest
Runtime: Eclipse Temurin JRE 21
JVM Options:
  - -XX:MaxRAMPercentage=75.0
  - -XX:+UseG1GC
Health Check: /actuator/health

Build Specifications:

- Multi-stage Docker build
- Gradle 8.11.1 + JDK 21
- Dependency caching for optimized build time

Video Server

Image: ddingsh9/mzc-video-server:latest
Purpose: TUS 프로토콜 기반 대용량 비디오 업로드 처리
Health Check: /actuator/health

4.2 Infrastructure Containers

MySQL 8.0

Image: mysql:8.0
Character Set: utf8mb4
Collation: utf8mb4_unicode_ci
Health Check: mysqladmin ping
Volume: mysql-data:/var/lib/mysql

Redis 7

Image: redis:7-alpine
Persistence: AOF (Append Only File)
Command: redis-server --appendonly yes
Health Check: redis-cli ping
Volume: redis-data:/data

---

5. Load Balancing Strategy

5.1 API Server (least_conn)

```
upstream spring-app {
    least_conn;
    server spring-app-1:8080;
    server spring-app-2:8080;
    server spring-app-3:8080;
}
```

선택 근거:

- 요청 처리 시간이 불균등한 API 특성에 적합
- 현재 연결 수가 가장 적은 서버로 요청 분배
- 자동 서버 상태 감지 및 장애 서버 제외

5.2 Video Server (ip_hash)

```
upstream video-server {
    ip_hash;
    server video-server-1:8080;
    server video-server-2:8080;
    server video-server-3:8080;
}
```

선택 근거:

- TUS(Resumable Upload Protocol) 업로드 세션 일관성 유지
- 동일 클라이언트 IP는 동일 서버로 라우팅

- 업로드 중단 후 재개 시 세션 데이터 접근 보장

---

6. Nginx Configuration

6.1 Performance Optimization

```
worker_processes auto;          # CPU 코어 수 자동 감지
worker_connections 1024;        # Worker당 최대 동시 연결

# Gzip Compression
gzip on;
gzip_vary on;
gzip_min_length 1024;
gzip_types text/plain text/css application/json application/javascript;
```

6.2 Proxy Settings

Setting	API Server	Video Server
connect_timeout	60s	120s
send_timeout	60s	120s
read_timeout	60s	120s
client_max_body_size	50M	500M

6.3 Static File Serving

```
location /static/profile/ {
    alias /var/www/uploads/files/;
    expires 30d;
    add_header Cache-Control "public, immutable";
    add_header X-Content-Type-Options "nosniff";
    add_header Access-Control-Allow-Origin *;
}
```

캐싱 전략:

- 프로필 이미지: 30일 캐시
- immutable 헤더로 불필요한 재검증 방지

---

7. Health Check Configuration

7.1 Container Level

Service	Check Method	Interval	Timeout	Retries	Start Period
MySQL	mysqladmin ping	10s	5s	5	30s
Redis	redis-cli ping	10s	5s	5	-
Spring API	wget / actuator/ health	30s	10s	3	60s
Video Server	wget / actuator/ health	30s	3s	3	60s

7.2 Dependency Management

```yaml
depends_on:
  mysql:
    condition: service_healthy
  redis:
    condition: service_healthy
```

시작 순서:

1. MySQL / Redis (병렬 시작)
2. Health check 통과 대기
3. Spring API / Video Server (인프라 healthy 후 시작)
4. Nginx (모든 앱 서버 healthy 후 시작)

---

8. Volume Management

8.1 Persistent Volumes

Volume	Mount Path	Purpose
mysql-data	/var/lib/mysql	데이터베이스 영속화
redis-data	/data	Redis AOF 파일
app-logs-{1,2,3}	/app/logs	인스턴스별 로그 분리
video-uploads	/app/uploads	비디오 파일 공유 스토리지
video-temp-{1,2,3}	/app/temp	인스턴스별 임시 처리 공간

8.2 Shared Volumes

```yaml
# 모든 API 인스턴스가 동일 업로드 디렉토리 공유
volumes:
  - ./uploads:/app/uploads
```

---

9. Network Architecture

9.1 Docker Network

```yaml
networks:
  lms-network:
    driver: bridge
```

Internal DNS Resolution:

- 컨테이너명으로 서비스 간 통신
- 예: mysql, redis, spring-app-1

9.2 Traffic Flow

```
[Client]
    |
    ├── :8080 ──→ [Nginx] ──→ [spring-app-{1,2,3}] ──→ [MySQL/Redis]
    |
    └── :8090 ──→ [Nginx] ──→ [video-server-{1,2,3}] ──→ [MySQL/Redis]
```

---

10. Security Considerations

10.1 Environment Variables

프로덕션 환경에서는 모든 민감 정보를 환경 변수로 관리:

```yaml
environment:
  DATABASE_PASSWORD: ${DATABASE_PASSWORD}
  JWT_SECRET: ${JWT_SECRET}
  ENCRYPTION_KEY: ${ENCRYPTION_KEY}
```

10.2 Network Isolation

- 애플리케이션 컨테이너는 외부 포트 노출 없이 Nginx를 통해서만 접근
- 관리 도구(phpMyAdmin, Redis Commander)는 프로덕션에서 제외

10.3 Security Headers

```
add_header X-Content-Type-Options "nosniff";
```

---

11. Deployment Commands

11.1 Local Development

```
# 인프라만 실행 (MySQL, Redis, 관리도구)
docker compose -f docker-compose-local.yml up -d

# IDE에서 Spring Boot 직접 실행
./gradlew bootRun
```

11.2 Development Environment

```
# 전체 스택 빌드 및 실행
docker compose up -d --build

# 로그 확인
docker compose logs -f spring-app
```

11.3 Production Deployment

```
# 환경변수 파일 준비
cp .env.example .env
vi .env  # 실제 값 설정

# 프로덕션 배포
docker compose -f docker-compose.prod.yml up -d

# 무중단 롤링 업데이트
docker compose -f docker-compose.prod.yml up -d --no-deps spring-
    app-1
docker compose -f docker-compose.prod.yml up -d --no-deps spring-
    app-2
docker compose -f docker-compose.prod.yml up -d --no-deps spring-
    app-3
```

---

12. Monitoring Endpoints

Endpoint	Purpose
/actuator/health	헬스체크
/health	Nginx 레벨 헬스체크

> **문서 작성일**: 2025-12-22
>
> **버전** : 1.0
>
> **작성자**: 송명주