공통 프로젝트 회고

1. 프로젝트 한 줄 요약

7 주 동안 AI 기반 커플 소통 중재 웹 서비스를 개발하여, 부모의 얼굴을 합성한 가상 AI 아이가 커플의 갈등을 해결하고 소통을 도와주는 혁신적인 플랫폼을 구축했다. Docker-compose 와 Jenkins 를 활용한 완전 자동화된 CI/CD 파이프라인 구현으로 개발 효율성을 크게 향상시켰다.

2. 프로젝트 개요

- **프로젝트명**: AI 아이 키우기 커플 웹 서비스
- **기간**: 7 주
- **팀 구성**: 6 명 (인프라/백엔드 1 명, 백엔드 2 명, 프론트엔드 3 명)
- **내 역할**: **인프라 및 백엔드 담당** (서버 구축, CI/CD 파이프라인, 실시간 통신 구현)
- **기술 스택**:
  - Frontend: React, TypeScript
  - Backend: Spring Boot, Java
  - Database: MySQL
  - Infrastructure: AWS (EC2, RDS, S3), Vercel
  - DevOps: Jenkins, Docker, Nginx, LiveKit
  - AI: OpenAI (GPT-4o, GPT-4o-mini, DALL-E-3), Gemini (Imagen-3.0)
- **목표**: AI 아이가 커플 간의 갈등을 중재하고 효과적인 소통을 도와주는 웹 서비스 개발

3. 주요 성과와 하이라이트

- **인프라 아키텍처 완성**: LiveKit 과 Spring Boot 를 Docker-compose 로 통합 구축하여 실시간 통신 환경 완성
- **완전 자동화 CI/CD 구현**: Jenkins 파이프라인을 통해 코드 배포부터 서버 재시작까지 자동화 달성
- **멀티 AI API 통합**: OpenAI 와 Gemini API 를 조합하여 텍스트 상담, 이미지 생성, 네컷만화 생성 기능 구현
- **실시간 기능 구현**: 커플 간 실시간 소통과 AI 아이와의 인터랙션을 위한 안정적인 서버 환경 구축
- **AWS 클라우드 인프라**: EC2, RDS, S3 를 활용한 확장 가능한 서버 아키텍처 설계

4. 도전 과제와 해결 과정

주요 도전: LiveKit 포트 충돌 관리

- **문제**: LiveKit 이 다수의 포트를 사용하면서 Docker 컨테이너 간 포트 충돌과 네트워크 이슈 발생
- **원인 분석**:
    - LiveKit 의 WebRTC 통신을 위한 다중 포트 필요성
    - Docker-compose 환경에서의 포트 매핑 복잡성
    - Spring Boot 와 LiveKit 간의 네트워크 통신 설정 미스매치
- **해결 방법**:
    - Docker-compose.yml 파일에서 포트 범위 매핑 최적화
    - LiveKit 설정 파일에서 포트 사용 범위 명시적 지정
    - 네트워크 격리를 위한 Docker 네트워크 별도 구성
- **결과**: 안정적인 실시간 통신 환경 구축 완료, 서버 재시작 시에도 포트 충돌 없이 정상 동작

5. Lessons Learned

기술적 배움

- **Docker-compose 활용법**: 복합 서비스 환경에서의 컨테이너 오케스트레이션 실무 경험 축적
- **Jenkins CI/CD**: 실제 프로덕션 환경에서의 자동화 파이프라인 구축과 운영 노하우 습득
- **LiveKit 실시간 통신**: WebRTC 기반 실시간 서비스의 인프라 설계와 포트 관리 전문성 확보

협업 배움

- **인프라 중심 역할**: 팀의 개발 효율성을 높이는 인프라 담당자의 책임감과 중요성 인식
- **문제 해결 소통**: 포트 충돌 같은 인프라 이슈를 팀원들에게 명확히 설명하고 해결 과정 공유하는 능력 향상

개인 성장

- **DevOps 마인드셋**: 개발과 운영을 통합적으로 바라보는 관점 확립
- **문제 해결 접근법**: 복잡한 인프라 이슈를 체계적으로 분석하고 단계별로 해결하는 방법론 체득

6. 다음에 시도하고 싶은 것

- **Kubernetes 도입**: Docker-compose 를 넘어선 본격적인 컨테이너 오케스트레이션 플랫폼 적용

- **모니터링 시스템 구축**: Prometheus, Grafana 를 활용한 실시간 서버 모니터링 환경 구성
- **마이크로서비스 아키텍처**: 현재의 모놀리식 구조를 서비스별로 분리하여 확장성 높은 시스템 설계
- **무중단 배포**: Blue-Green 배포 또는 Rolling 배포 전략을 통한 서비스 가용성 극대화

차별화 요소

🏆 프로젝트 하이라이트

"3 일 밤샘 끝에 찾아온 기적의 순간"

프로젝트 5 주차, LiveKit 포트 충돌로 인해 실시간 통신이 계속 끊기는 상황이 발생했다. 팀원들은 이미 AI 아이와의 대화 기능을 완성해놓은 상태였는데, 정작 실시간으로 연결되지 않아 모든 기능이 무용지물이 될 뻔했다.

3 일간 밤을 새워가며 Docker 네트워크 설정을 수십 번 바꿔보고, LiveKit 공식 문서를 파헤치며 포트 매핑을 재설계했다. 새벽 4 시쯤 마지막 시도에서 드디어 "연결 성공" 메시지가 떴을 때의 그 순간은 아직도 생생하다.

이 해결 덕분에 최종 발표에서 실시간으로 커플의 대화를 중재하는 모습을 완벽하게 시연할 수 있었다.

📈 프로젝트에서 나의 역할

시작: 프로젝트 초기에는 Jenkins 와 Docker 를 한 번도 실제로 사용해본 적이 없었다. CI/CD 라는 용어조차 개념적으로만 알고 있는 상태였다.

중간: LiveKit 도입을 결정하면서부터 본격적인 도전이 시작되었다. Docker-compose 파일을 작성할 때마다 컨테이너가 제대로 올라오지 않고, Jenkins 파이프라인은 매번 다른 곳에서 실패했다. 일주일 동안 하루에 평균 20 번씩 빌드 실패를 경험했다.

결과: 프로젝트 막바지에는 코드 푸시 한 번으로 자동으로 테스트, 빌드, 배포까지 완료되는 완전 자동화 시스템을 구축했다. 팀원들이 "이제 배포 걱정 없이 개발에만 집중할 수 있다"고 말했을 때, 인프라 담당자로서의 보람을 크게 느꼈다.

💡 자유 형식 질문

Q: 다음 프로젝트를 한다면 절대 하지 않을 것은?

A: 포트 관리 없이 무작정 새로운 서비스를 추가하는 것. 이번에 LiveKit 을 도입하면서 포트 계획 없이 진행했다가 3 일간 고생했다. 다음에는 프로젝트 초기에 아키텍처 설계 단계에서 각 서비스별 포트 사용 계획을 미리 세우고, Docker 네트워크 구성도 사전에 설계할 것이다. "일단 돌아가게 만들고 나중에 정리하자"는 생각이 얼마나 위험한지 뼈저리게 깨달았다.

7. 마무리 한 줄

이번 프로젝트는 단순한 웹 서비스 개발을 넘어 인프라부터 AI 까지 아우르는 풀스택 경험이었으며, 특히 DevOps 영역에서의 자신감을 크게 높일 수 있었다. 다음 프로젝트에서는 더욱 안정적이고 확장 가능한 인프라로 팀의 성공을 뒷받침하겠다.