**이동우 BACKEND ENGINEER**

**이메일**: dongwoo.lee.9711@gmail.com

**연락처**: 010-6309-3377

**깃허브**: github.com/dongwooooooo

**SUMMARY**

실사용자의 경험을 고려해 문제를 정의하고, 서비스 성능 개선과 운영 안정성을 함께 개선할 수 있는 구조를 주도적으로 설계하고 구현한 경험이 있습니다.

Spring Boot, Django, Nest.JS 기반의 다양한 웹 서비스 개발 경험이 있습니다.

**PROJECT**

**하루연결** Backend(2023.12~2024.2)

실시간 다중 사용자 일기 작성과 키워드 분석을 통한 AI 이미지 자동 생성 서비스

**Tech stack** - Django, Django Channels, Redis, Nginx, Docker, AWS, RabbitMQ, Celery

**<직면한 문제>**

**1. AI 이미지 생성 병목 현상**

• AWS Comprehend를 통해 다중 키워드를 추출한 후 DALL-E Open API를 순차적으로 호출하면서, 이미지 생성과정에서 병목 현상이 발생하여 서비스의 응답 시간이 37.6초에 달하는 심각한 성능 문제 발생

**2. 과도한 S3 이미지 저장 비용 및 로딩 성능 저하**

• 생성되는 AI 이미지(1~3MB)의 용량이 서비스 요구 수준에 비해 커서 S3 저장 비용이 불필요하게 발생했으며, 이미지 로딩 성능 저하의 원인

**<문제 해결 과정>**

**1. 비동기 병렬 처리를 통한 응답 속도 개선**

• 이미지 생성, 후처리 과정을 Celery워커로 위임하여 여러 이미지 생성을 병렬로 처리하도록 구현

• RabbitMQ를 메시지 브로커로 활용하여, 등록된 Celery작업이 완료되면 Django Channels의 Channel Layer에 완료 메시지를 발행

• 이미지 생성 프로세스가 기존 37.5초에서 15초로 단축되었으며, 완료 시 사용자에게 실시간으로 결과를 전달하여 경험을 향상

**2. 이미지 최적화를 통한 비용 절감 및 성능 향상**

• Python 이미지 라이브러리를 활용하여 서비스 요구사항에 알맞게 이미지를 압축하고 리사이징을 적용

• 이미지 용량을 90% 절감할 수 있었으며, S3 저장 비용을 절감하는 동시에 이미지 로딩 속도를 향상

**TecheerZip** Backend(2025.3.2~) [링크](https://www.techeerzip.cloud/)

개발 동아리 활동과 회원들의 이력서를 공유하는 사이트

**Tech stack** - Spring, Redis, RabbitMQ, Docker Swarm, Jira, Slack, GCP

**<직면한 문제>**

**1. 특정 API 응답 지연**

• 주요 API의 응답시간이 1~3초 가량의 체감 지연이 발생하여 사용자 경험을 저해

• 기존 모니터링 시스템으로 어느 API에 문제가 있는지 파악할 수 있었으나, 구체적으로 해당 API 내의 어떤 로직이 문제인지 빠르게 식별하기 어려운 상황

**2. 단일 장애 지점 및 불안정한 서비스 장애 인지 체계**

• 이력서 키워드 추출, 크롤링 등 외부 서비스와 직접적인 의존 관계 구조로 인해, 해당 외부 서비스에 문제가 발생할 경우 메인 서비스의 주요 기능이 중단되는 단일 장애 지점이 존재

• 외부 서비스 문제 뿐만 아니라 내부 서비스 오류 발생 시에도 즉각적인 인지가 어려워 장애 대응 시간이 길어짐

**3. 복잡한 환경에서의 자원 관리 및 서비스 운영 비효율**

• 메인(Spring), 크롤링과 이력서 추출(Go), 레거시(Nest.js)의 다른 언어 기반 서비스들이 혼재

• 크롤링이나 이력서 추출처럼 단기적으로 상당한 자원을 소비하는 작업들을 단일 클라우드 서버에서 운영할 경우, 서버 비용 부담과 자원 경합으로 인한 비효율성이 발생

• 이러한 환경에서는 개별 서비스의 독립적인 배포와 서비스 확장, 롤백에 제약

**4. 불안정한 배포 프로세스**

• 개발 완료된 기능이 별도의 테스트 환경 없이 바로 프로덕션으로 배포

• 그 결과 단일 테스트에선 확인되지 않았던 문제점들이 실서비스 환경에서 직접적으로 발견되었고,

• 잦은 에러 발생과 핫픽스로 이어져 서비스의 안정성을 크게 저해

**<문제 해결 과정>**

**1. 트레이싱을 통한 특정 API 지연 확인 및 성능 개선**

• OpenTelemetry tracing과 Jaeger를 도입하여 서비스 전반의 트레이스를 수집하고 시각화

~~• 이를 통해Prisma의 N+1 문제와 이미지, 이력서 파일을 mulitpart form data로 받아 응답 지연이 발생~~

~~• 쿼리 최적화와 multipart 요청을 분리하여 응답 시간을 단축~~

**2. 이벤트 기반 아키텍처 전환 및 통합 장애 알림 시스템 구축**

• 외부 서비스 요청을 이벤트 기반으로 전환하여 직접적인 의존성을 제거

• 외부 서비스API 요청 실패 및 내부 서비스 오류 발생 시 자동으로 DB에 이벤트 또는 로그를 저장하고 Slack 알림을 전송하는 시스템을 구축

• 덕분에 장애 발생 시 빠르게 인지하고 대응할 수 있도록 유지보수 환경을 개선

**3. Docker Swarm 기반 고가용성 아키텍처 구축**

• 각 서비스를 독립적인 컨테이너 환경으로 격리하고 서비스 규모에 부합하는 통합 운영 시스템을 구축

• 이를 통해 서로 다른 언어 기반 서비스 간의 의존도를 낮추고, 각 서비스를 독립적으로 배포 및 관리

• 결과적으로 수평확장과 장애 대응이 용이한 고가용성 구조를 확보

**4. 스테이징 서버 및 CI/CD 파이프라인 구축**

• 프로덕션과 동일한 스테이징 서버를 구축하여 실서비스와 유사한 환경에서 기능 테스트를 수행한 후 배포하도록 프로세스를 개선

• GitHub Actions 기반의 CI/CD 파이프라인을 구축하여 단위 테스트, 프로덕션/스테이징 Docker 이미지 빌드, Slack 배포 승인 알림, 자동 배포까지의 과정을 자동화

• 이러한 노력으로 배포 안전성을 크게 높였으며, 개발 생산성이 향상