


포트폴리오1: 🛒 이커머스 서비스 서버

개인 프로젝트

2024.10-2024.12

 <https://github.com/jimyungkoh/ecommerce-service-server>

Server	#NestJS #Prisma #MySQL #Docker #Kafka #Jest
--------	---

트러블슈팅 🌟 (문제와 해결)

문제: 대용량 상품 조회 성능 문제 해결 (1,000만 건)

해결 과정:

신상품순 조회(상품 수정일) 데이터 특성 분석

데이터 시딩 기준: 2010-01-01 ~ 현재까지의 기간; 밀리초[datetime(3)] 단위

특징: 카디널리티 1로 각 상품의 수정 시간이 거의 유일한 값을 가짐

성능 영향: 높은 카디널리티로 인해 인덱스 효율성 극대화 예상

```
CREATE INDEX idx_product_updated_at ON product (updated_at);
```

가격순 조회(가격) 데이터 특성 분석

데이터 시딩 기준: 1,000원 ~ 10,000,000원까지의 값; 100원 단위

특징: 카디널리티 0.01로 같은 가격대의 상품이 한 가격당 약 100개씩 존재함

성능 영향: 카디널리티가 낮더라도 값이 정렬 상태로 저장되므로 쿼리 성능이 크게 향상될 것으로 예상됨

```
CREATE INDEX idx_product_price ON product (price);
```

결과:

신상품순 조회

개선 전

5.01초

개선 후

0.000154초

Table scan → Sort → Limit

Index scan → Limit

스캔 행 수: 10,000,000

스캔 행 수: 20

정렬 방식: 전체 데이터 정렬

정렬 방식: 인덱스 정렬 활용

가격순 조회

개선 전

8.22초

개선 후 (가격 낮은순) ↑

0.015초

개선 후 (가격 높은순) ↓

0.014초

Table scan → Sort → Limit

Index scan → Limit

Index scan (reverse) → Limit

스캔 행 수: 10,000,000

스캔 행 수: 20

스캔 행 수: 20

예상 비용: 1.05e+6

예상 비용: 0.121


예상 비용: 0.121


실제 시간: 8220ms

실제 시간: 14.9ms


실제 시간: 13.7ms


해결 과정:

**정합성 확보**

 **비관적 락(Pessimistic Lock) 도입**

- 재고 테이블에 row-level lock 적용
- 동시 접근 시 순차적 처리로 정합성 보장
- 초과 주문 원천 차단

**처리량 개선**

 **이벤트 기반 트랜잭션 분리**

- 주문 생성/재고 감소/결제/주문 완료 단계 분리
- 비동기 처리로 API 응답 시간 감소
- 서비스 간 결합도 감소

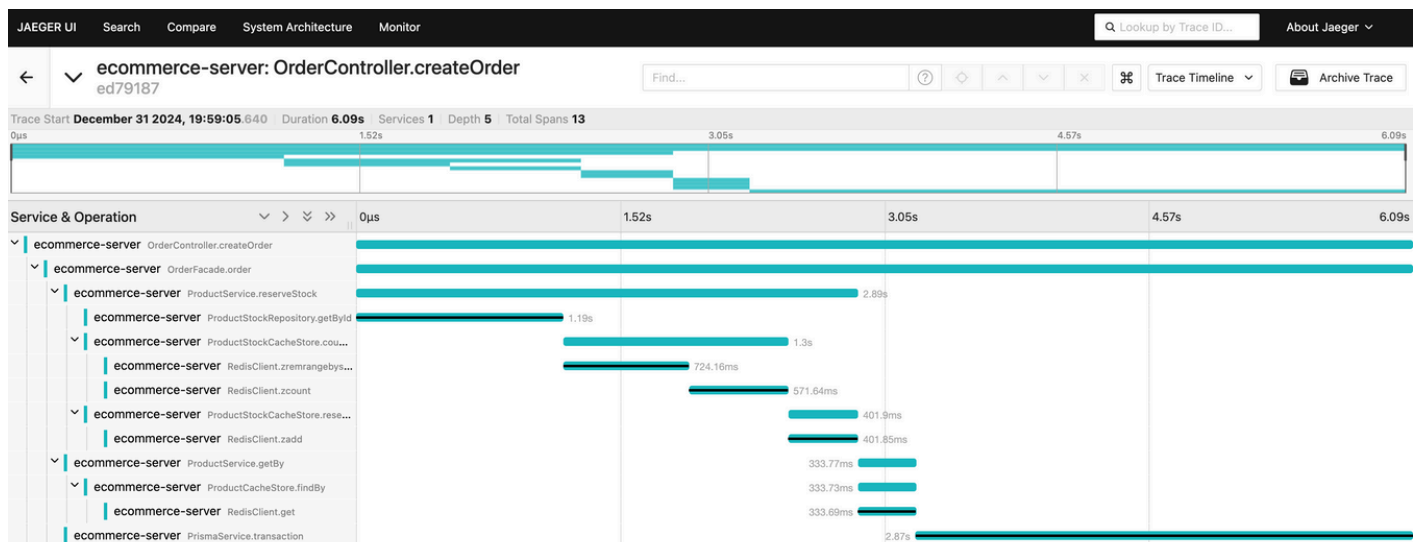
결과: 평균 응답 시간 51.6%, 요청 처리량 96% 개선

평균 응답 시간	↑ 51.6%	처리량	↑ 96%
개선 전	5.19 s	개선 전	177 회/s
개선 후	2.51 s	개선 후	347 회/s

기타 성과

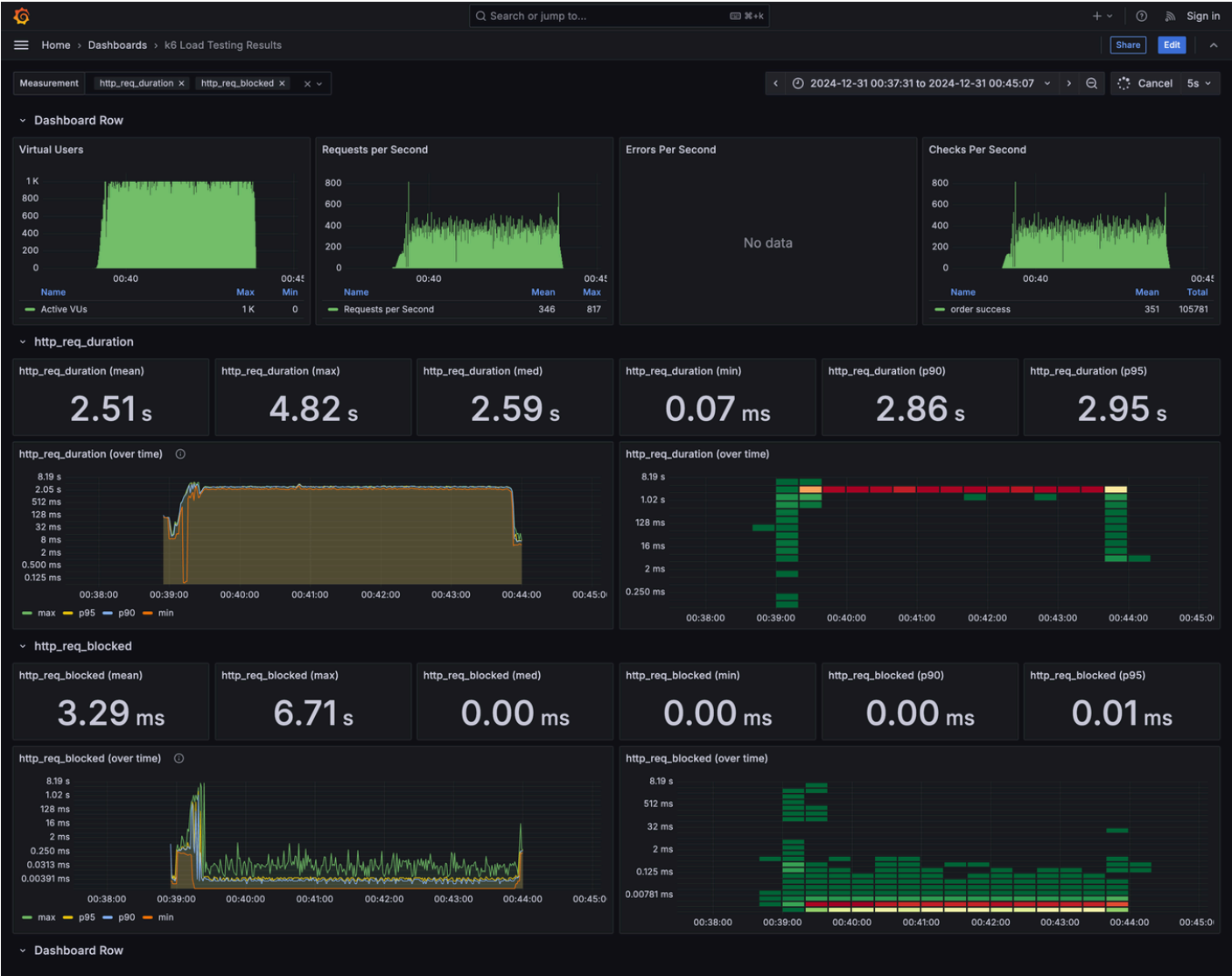
분산 추적 시스템 구축

비동기 장애 추적이 어려웠던 주문-재고-결제 플로우에 OpenTelemetry 기반 추적 시스템을 구축하여 응답시간 프로파일링
Trace ID 기반으로 5계층 13개 트랜잭션의 의존성을 시각화하여 장애 원인 분석 시간 단축



부하 테스트 모니터링 환경 구축 (Grafana - InfluxDB - k6)

실제 프로덕션 환경의 트래픽 패턴을 시뮬레이션하기 위해 k6로 부하 테스트 자동화
시계열 데이터베이스(InfluxDB)로 테스트 결과를 저장하여 성능 변화 추이와 패턴 분석 가능
주요 성능 지표(VUser, RPS, 응답시간)를 Grafana로 실시간 모니터링하여 즉각적인 병목 구간 감지



팀프로젝트

사진 방명록 공유 플랫폼
2024.05 -2024.09


역할

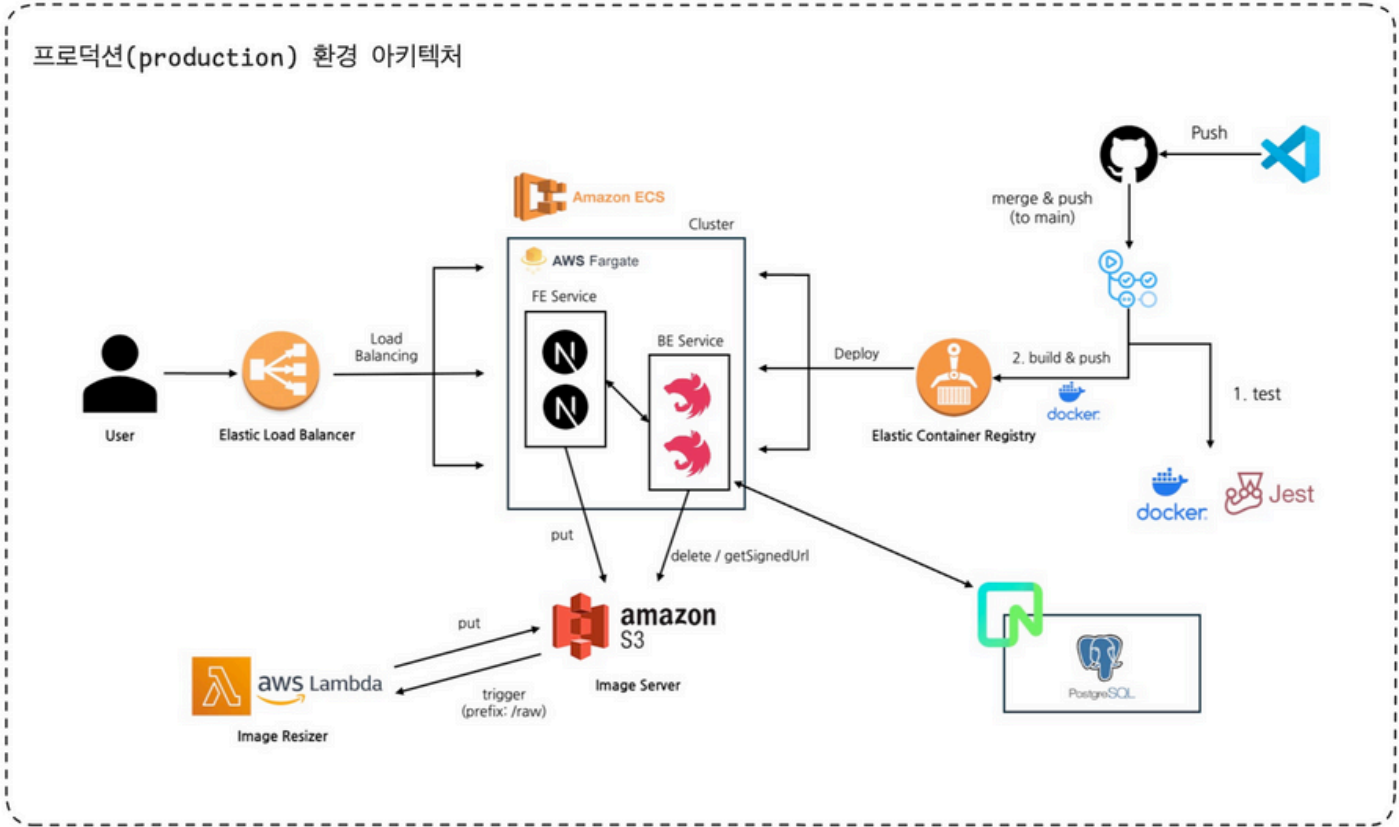
Client	<ul style="list-style-type: none">방명록(링크 생성, 작성, 상세) 페이지공통 컴포넌트 설계폴라로이드 사진 필터 구현
Server	<ul style="list-style-type: none">방명록 도메인 API 설계 및 구현이미지 최적화 시스템 구축AWS 클라우드 인프라 구축 및 서비스 배포·운영



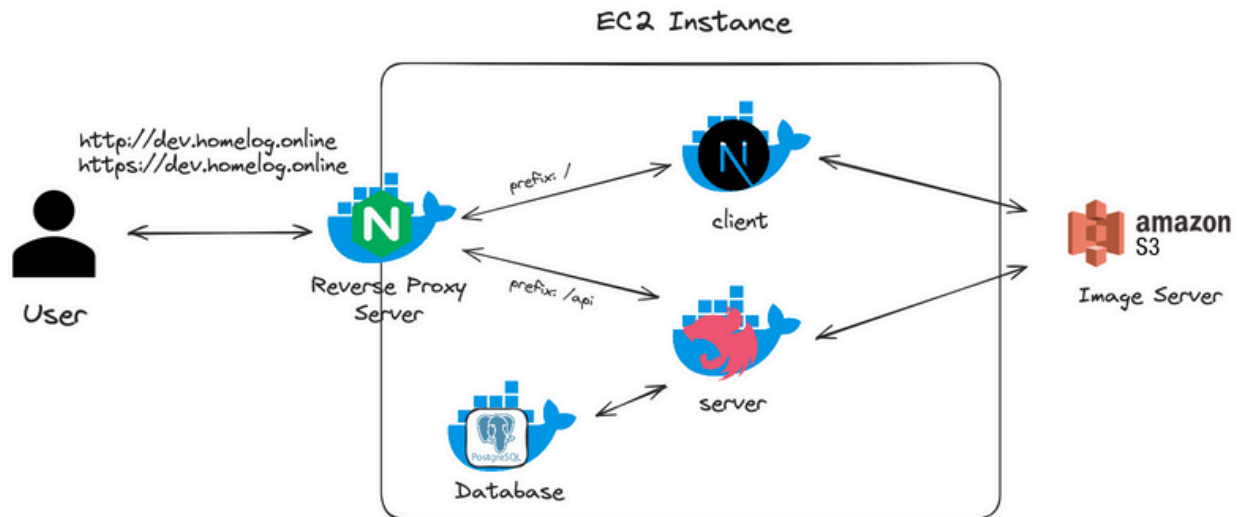
타임스탬프 필터를 지원하는 사진 기반 방명록 웹

클라이언트  <https://github.com/jimyungkoh/homelog-client>

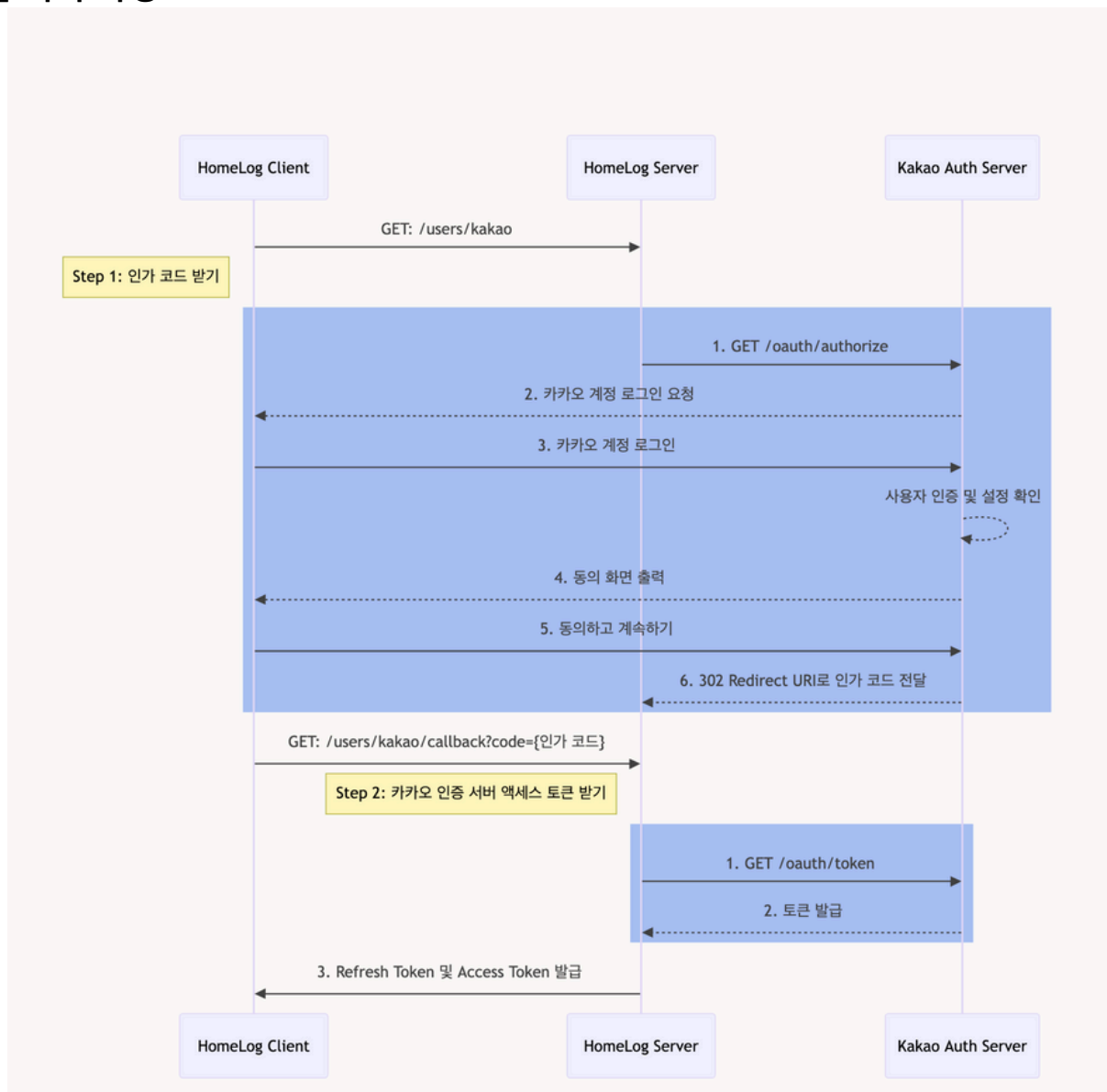
서버  <https://github.com/jimyungkoh/homelog-server>



개발(dev) 환경 아키텍처



로그인 처리 과정



트러블슈팅 🌟 (문제와 해결)

문제: 개발 환경 배포시 비용 문제

- 프로덕션 환경에서 ECS, ALB 사용
 - 월 고정비용 약 \$65 발생

해결

- 개발 환경에서 EC2 Spot + Nginx 조합 사용으로 ECS와 ALB 조합 대체
 - 월 고정비용 약 77% 절감

문제: 이미지 업로드 및 저장 문제

- 특정 브라우저에서 필터 적용 후 저장 기능 오류 발생

해결

- 원인
 - 사파리 브라우저와 캡처 라이브러리 간 낮은 호환성
- 해결책
 - 라이브러리 사용 중지 후 Canvas API 로 이미지를 합성 후 저장하는 방식 채택

문제: 페이지 로딩 속도 저하 문제

- 필터 처리한 이미지를 원본 그대로 웹에 노출

해결

- 원인
 - 무거운 원본 이미지의 파일 크기
- 해결책
 - 업로드된 이미지를 자동 리사이징 후 저장(AWS 람다)
 - 화면 크기에 맞는 최적화된 이미지 제공
 - 평균 파일 크기 98% 감소 (640x960px 기준)

포트폴리오3: 💧 Sweatier

팀프로젝트 / 백엔드, DevOps

수준별 운동 매칭 서비스

2024.02 - 2024.03

역할

- 사용자 랭킹 배치 프로세스
- GCP 클라우드 인프라 구축 및 서비스 배포/운영
- 검색 API
- 소셜 로그인 인증 시스템 구현 (OAuth 2.0)



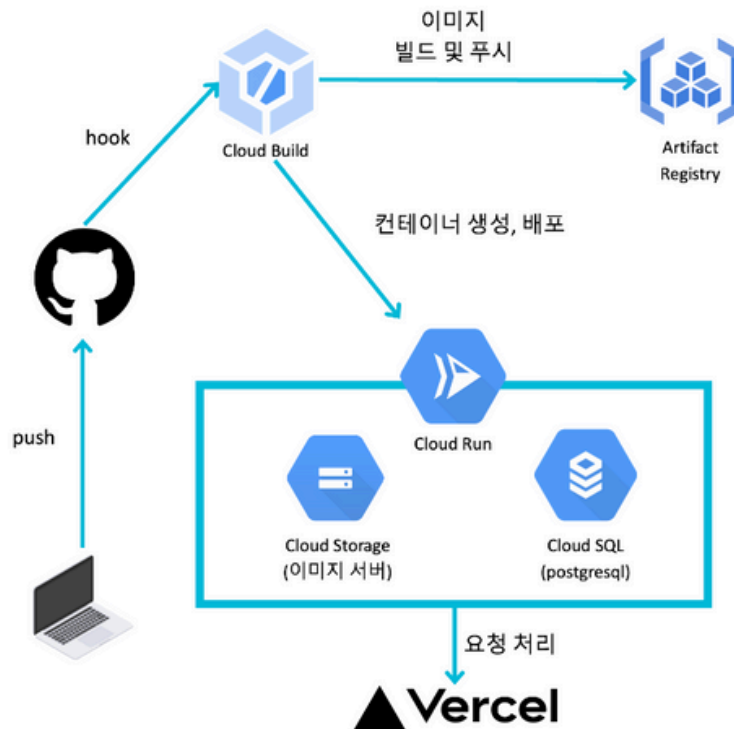
랭크 시스템을 지원하는 수준별 운동 매칭 서비스

#NestJS #TypeScript #Prisma #PostgreSQL
#JWTTokenOAuth2.0 #Docker
#GCP(Cloud SQL, Cloud Storage, Artifact Registry, Cloud Build) #PostmanAPI

서버 

<https://github.com/jimyungkoh/sweatier-server>

프로덕션(production) 환경 아키텍처



트러블슈팅 🌟 (문제와 해결)

문제 1: 예외 처리의 비효율성

- 도메인별 에러 코드 관리
 - 서버 측 에러 코드 중복
- 컨트롤러에서 비즈니스 예외 처리
 - 여러 라우트 메서드에서 코드 중복
 - 컨트롤러 복잡성 증가

해결 1

- 에러 코드를 한 곳에서 관리
- 서비스 로직에서 서비스 예외 발생(Error 코드 상속)

문제 2: 서비스 계층 테스트 코드의 부재

- 코드 수정 후 매번 API 테스트로 기능 확인이 필요해 비효율적

해결 2

- 단위 테스트 코드 작성
 - 복잡한 테이블 구조에는 모킹을 활용해 테스트 시간 단축

문제 3: 수동 빌드·배포의 비효율성

- 개발 생산성 저하
- 신규 배포 시 서비스 중단 발생

해결 3

- CI / CD 파이프라인 도입을 통한 빌드·배포 자동화
- 배포 스크립트를 통해 새 서버 가동이 정상적으로 완료된 경우에만 기존 서버 대체