김찬혁 Portfolio

Backend Developer · 트래픽 설계와 운영 안정성을 함께 책임지는 개발

010-3738-1882
dissolve1882@naver.com
github.com/inhadissolve

Proficiency

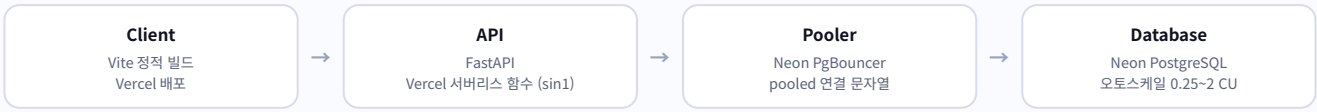
- FastAPI · NestJS 기반 REST API를 설계하고 **실사용자 대상 서비스를 직접 운영**할 수 있습니다.
- 서버리스 환경의 인스턴스 단위 커넥션 특성을 이해하고, **커넥션 고갈이 발생하지 않는 풀 구조**를 설계할 수 있습니다.
- 폴링 주기와 사용자 체류 시간을 근거로 **예상 요청량과 인프라 예산을 역산**해 피크 트래픽을 사전에 대비할 수 있습니다.
- AWS ALB · EC2 Auto Scaling · RDS 구성을 **Terraform으로 정의**하고 GitHub Actions OIDC로 배포할 수 있습니다.
- CloudWatch 지표로 장애 원인을 추적하고, **헬스체크 기반 자동 복구 구조**를 설계할 수 있습니다.
- pytest · Jest · Testcontainers로 **회귀 테스트와 E2E 검증 환경**을 구축할 수 있습니다.
- OpenAI API 호출에 **예산 · 생성 한도 가드레일**을 적용해 운영 비용을 통제할 수 있습니다.

Project Portfolio

서버리스 환경에서 커넥션 · 폴링 부하 · 컴퓨트 예산을 역산해 피크 기간 서비스 전면 정지를 사전에 차단

프로젝트 메모라이즈 — 암기 학습 웹서비스 (2025.11 ~ 현재, 1인 기획 · 개발 · 운영) · **스택** React 19.2 · TypeScript 5.9 · FastAPI 0.140 · SQLAlchemy 2.0 · PostgreSQL(Neon) · Vercel Serverless
규모 전체 회원 449명 · 최근 14일 학습 완료 16,496건 · 최고 동시접속 36명 · 피크 기간 예상 동시접속 100~300명

01 전체적인 아키텍처



요청 단위로 함수 인스턴스가 생성되는 구조이므로, **인스턴스 수 × 풀 크기**가 곧 DB 커넥션 수가 됩니다. 드라이버는 psycopg2 + SQLAlchemy 2.0을 사용하고, 서버리스 환경을 감지해 **pool_size 1 / max_overflow 2(인스턴스당 최대 3)**, 로컬 개발은 10 / 10으로 분기합니다.

02 문제 원인

1) **서버리스에서 커넥션 풀은 곱셈으로 늘어난다.** 일반 서버 기준으로 풀을 크게 잡으면, 인스턴스 하나가 요청 하나를 처리하는 서버리스에서는 동시 인스턴스 수만큼 커넥션이 곱절로 늘어 DB의 **max_connections 한도**에 그대로 부딪힙니다.
2) **부하를 좌우하는 건 동접자 수가 아니라 체류 시간이었다.** 코드를 전수 조사한 결과 로그인 사용자는 화면을 열어두기만 해도 문의 배지 **30초**, 알림 **30초**, 접속 heartbeat **2분** 주기로 요청이 나가, **1인이 탭 하나를 1시간 열어두면 약 270회**가 발생했습니다.
3) **진짜 위험은 커넥션이 아니라 월 컴퓨트 예산이었다.** 사용 중인 플랜은 월 **100 CU-hour**와 저장 **0.5GB**가 상한이고, 예산을 소진하면 다음 결제 주기까지 컴퓨트가 정지해 **사이트 전체가 열리지 않습니다.** 피크 기간에 오토스케일이 2 CU로 하루 8시간 유지되면 하루 16 CU-hour, **6일이면 월 예산을 모두 소진**하는 계산이 나왔습니다.

03 해결 과정

1) **환경 감지 기반 커넥션 풀 분기.** 서버리스 여부를 런타임에 판별해 **pool_size 1 / max_overflow 2**로 제한하고, 로컬 개발 환경만 기존 크기를 유지하도록 분리했습니다. 인스턴스당 커넥션 상한을 3개로 고정해 동시 인스턴스가 늘어도 DB 한도에 도달하지 않게 했습니다.
2) **pooled 연결로 커넥션을 병목에서 제외.** PgBouncer를 경유하는 pooled 연결 문자열을 표준으로 문서화했습니다. 오토스케일 상한인 **2 CU 기준 max_connections 839**를 확보해, 수백 개의 동시 함수 인스턴스가 떠도 커넥션은 병목이 되지 않는 구조임을 수치로 확인했습니다.
3) **폴링 주기 전수 조사 후 예상 요청량 산출.** 화면별 폴링 주기와 발생 조건을 표로 정리해 **1인 시간당 약 270회**라는 기준값을 만들고, 피크 동시접속 100~300명을 대입해 컴퓨트 예산 소모 속도를 역산했습니다. 부하 기준을 **"동접자 수"에서 "체류 시간 × 폴링 주기"**로 바꾼 것이 핵심 판단이었습니다.
4) **리전 정렬로 왕복 지연 제거.** 함수 배포 리전이 기본값이면 DB까지 왕복으로 요청마다 수백 ms가 추가되는 문제를 확인하고, 함수 리전을 **DB와 동일한 sin1로 고정**했습니다. 리전 설정을 되돌리면 안 되는 이유를 저장소 규칙으로 남겨 재발을 막았습니다.
5) **업그레이드 판단 신호를 수치로 정의.** "몇 명부터 위험"이 아니라 **컴퓨트 예산 70~80% 도달**, **저장 공간 70~80% 도달**을 전환 기준으로 정하고, 예측 가능한 피크 주간에는 시작 전에 선제적으로 플랜을 올리는 운영 규칙을 세웠습니다.

04 테스트 · 검증

실제 부하는 **관리자 운영 대시보드**로 관측했습니다. 시간대별 최고 동시접속자와 발생 시각, 기능별 이용 회원 수, AI 호출 실패 건수와 누적 추정 비용을 상시 확인해 설계 가정과 실측값을 대조했습니다. 회귀 검증은 **백엔드 pytest 685개 · 프론트 331개**로 유지하며, 서버가 받는 요청 형식뿐 아니라 **실제 화면이 보내는 파라미터 형태까지** 테스트에 포함해 배포 후 재발을 막았습니다.

05 결과

0건	839	36명	16,496건
피크 기간 예산 소진 다운	확보한 max_connections	실측 최고 동시접속	14일 학습 완료

1) **피크 기간 다운타임 0.** 컴퓨트 예산 소진으로 시험 기간 중 서비스 전체가 정지하는 시나리오를 사전에 계산해 차단했고, 피크 구간을 중단 없이 통과했습니다.
2) **커넥션 고갈 위험 제거.** 인스턴스당 3개 제한과 pooled 연결 조합으로, 오토스케일 상한에서도 커넥션 여유를 수치로 확인할 수 있는 구조를 만들었습니다.
3) **판단 기준의 전환.** 트래픽을 "몇 명이 접속하는가"가 아니라 "얼마나 오래 머무르며 어떤 주기로 요청을 보내는가"로 보게 되었고, 이후 인프라 결정을 감이 아닌 계산으로 내리게 되었습니다.

Case 2. 원인 불명의 5분 다운타임 추적

운영 중 반복되던 약 5분간의 접속 불가를
CPU 크레딧 소진으로 규명하고 자동 복구 구조로 전환

프로젝트 HomeFit — 청년 주거 · 금융 의사결정 서비스 (2026.06 ~ 2026.08, UMC 10th · 10인 팀 · Backend Team Lead) · **스택** NestJS 11 · Node.js 22 · Prisma 6 · PostgreSQL 18.4(RDS) · EC2 t3.small · Terraform 1.8 · AWS Provider 5.80

01 문제 원인

1) **증상은 있는데 로그에는 흔적이 없었다.** 운영 중 약 5분간 접속이 끊겼다가 **별도 조치 없이 자동으로 복구**되는 현상이 반복됐습니다. 애플리케이션 로그에는 명확한 오류가 남지 않아 원인 파악이 어려웠습니다.
2) **버스터블 인스턴스의 크레딧이 바닥나 있었다.** CloudWatch 지표를 추적한 결과 **t3 버스터블 인스턴스의 CPU 크레딧이 소진**되어 처리 속도가 급격히 떨어졌고, 이로 인해 ALB 헬스체크에 연속으로 실패하고 있었습니다.
3) **5분이라는 시간은 복구 설정의 합이었다.** 헬스체크 30초 간격 · 연속 3회 실패로 **90초 만에 unhealthy 판정**, 이후 Auto Scaling이 인스턴스를 교체하고 새 인스턴스가 기동해 healthy로 잡히기까지의 시간이 더해져 약 5분의 공백이 만들어진 것이었습니다.

02 해결 과정 · 결과

1) **복구를 우연이 아닌 보장으로.** Auto Scaling의 헬스체크 타입을 **ELB 기준**으로 두어, 크레딧이 고갈되어 응답하지 못하는 인스턴스가 자동으로 교체되도록 확정했습니다. min 1 / max 3 구성으로 교체 중에도 서비스가 유지됩니다.
2) **고갈 이후가 아니라 이전에 감지. CPUCreditBalance 알람**을 추가해 크레딧이 바닥나기 전에 신호를 받도록 했고, UnHealthyHostCount · 5XX · RDS 메모리 · 커넥션 수 알람을 함께 정의해 장애 징후를 지표로 관측할 수 있게 했습니다.
3) **같은 구성을 코드로 고정.** 헬스체크 주기와 임계값, ASG 정책을 **Terraform으로 정의**해 사람이 콘솔에서 바꾸는 것이 아니라 코드 리뷰를 거쳐 변경되도록 만들었고, 데모데이 현장 운영을 중단 없이 마쳤습니다.

그 외 주요 프로젝트

수담(手談) — 실시간 수어 인식 · 번역
2025.07 ~ 2025.09 · 7인 팀 팀장 · 2회 수상

JPEG 프레임을 MediaPipe로 처리해 10프레임 × 194차원 텐서로 변환하고, 라벨 · 신뢰도 · 상위 후보를 반환하는 FastAPI · Docker 추론 API를 구현했습니다. 학습에 없던 샘플별 정규화를 실시간 예측 스크립트에서 제거해 학습 · 추론 입력 스케일을 일치시켰고, 비수어 동작을 none으로 묶는 학습 로직과 신뢰도 임계값 · 3회 연속 일치 · 중복 누적 방지로 오탐과 중복 출력을 제어했습니다.

Python 3.10 · MediaPipe 0.10 · TensorFlow 2.13 · FastAPI 0.110 · Docker

RealGain — 산업 데이터 이상탐지 AI
2025.09 ~ 2025.12 · 기업연계 · 대상(1위)

24채널 진동 BIN 데이터를 RCP 채널쌍별 Orbit 이미지로 변환해 torchvision ResNet18로 정상 · 이상을 분류하고, Grad-CAM으로 판단 근거를 시각화했습니다. --json 출력 옵션으로 백엔드에서 바로 연동 가능한 추론 모듈로 만들었습니다.

Python 3.10 · PyTorch 2.8 · torchvision 0.23 · Grad-CAM · Pandas 2.2

GitHub github.com/inhadissolve · **Email** dissolve1882@naver.com · **Blog** blog.naver.com/dissolve_chan