Portfolio

수집부터 저장까지

데이터가 흐르는 길을 직접 설계하는 개발자

최수림입니다.

ruriaim@gmail.com

010-9023-0261

프로필

수집부터 저장까지, 데이터가 흐르는 길을 직접 설계하는 개발자 **최수림**입니다.

데이터가 어디서 시작해 어디에 닿는지를 먼저 생각합니다.
기능을 구현하기 전에 흐름을 먼저 설계하고, 그 흐름이 끊기지 않도록 처음부터 끝까지 직접 만드는 것을 좋아합니다.
백엔드 파이프라인 설계부터 인프라 구축까지, 데이터가 흐르는 길 위에서 시스템을 완성해온 개발자입니다



최수림 Choi su-rim
2000.09.04

AWARDS / CERTIFICATES	2026.04	AI 기반 올인원 채용 로드맵 플랫폼 **프로젝트 우수상**
EDUCATION / EXPERIENCE	2025.07 ~ 2026.06	**삼성청년SW·AI아카데미 14기**
	2025.01 ~ 2025.06	**코리아IT아카데미 개발 국비 훈련과정**
	2023.03 ~ 2025.08	**부경대학교 컴퓨터공학과 졸업**
	2019.03 ~ 2022.12	**동서대학교 컴퓨터공학과 재학**
Certification	2025.11	**ADSP**
	2025.09	**SQLD**
	2025.01	**PCCE**
	2024.09	**정보처리기사**

기술 스택



■■■■□
Spring Boot

- Spring 생태계를 기반으로 REST API 서버 설계 및 구현 가능
- JPA를 활용한 데이터 모델링
- Security와 JWT로 인증·인가 처리 경험



■■■■□
MySQL

- 관계형 데이터 모델링 및 쿼리 최적화 경험
- 멀티 DB 환경에서 MySQL의 강점을 살린 역할 분리 설계



■■■□□
Redis

- Cache, Stream, Pub/Sub 등 다양한 활용 패턴 적용 경험
- Redis Stream 기반 MSA 파이프라인 설계 및 구현



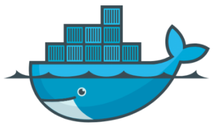
■■■■□
Jenkins

- GitLab Webhook 연동으로 CI/CD 파이프라인 자동화
- Blue/Green 무중단 배포 스크립트 직접 구현



■■■■□
Nginx

- 리버스 프록시 설정 및 HTTPS 인증서 적용 경험
- Blue/Green 무중단 배포 환경에서 트래픽 라우팅 설정



■■■■□
Docker

- Docker Compose로 멀티 컨테이너 환경 설계 및 운영 경험
- 개발/운영 환경 분리 및 7개 서비스 오케스트레이션 경험

프로젝트 목차



개발 직군 취업준비생 을 위한
AI 기반 올인원 채용 로드맵 플랫폼

꿀잡

Power21
EMS

MSA 기반 마이크로그리드
EMS 실시간 모니터링 및 AI 예측 시스템

AIEMS

TIKIT

스마트 명함 교환 및 출입 권한 플랫폼

TIKIT

AIEMS 2026.04 ~ 2026.05 (인원 : 6명)

▢ 프로젝트 개요

MSA 기반 마이크로그리드

에너지 관리 시스템

주요 기능

- 실시간 전력 모니터링
- 우선순위 기반 자동 제어
- 토폴로지 기반 구역별 전력 제어
- 알람, 이벤트 관리
- 대시보드 시각화



기여도 · 사용 기술 · 담당역할

AIEMS

사용 기술 · 기여도

20%

6인 팀
프론트 2명 · 백엔드 3명(본인 담당) · 인프라 1명
EMS 백엔드 전체 단독 설계

Python · asyncio
수백 ms 주기 다수 장치 동시 처리 → **비동기 I/O**

Redis Stream
Consumer Group · **at-least-once** · Kafka 대비 경량

MQTT · Mosquitto
IoT 경량 pub/sub · **장치 추가 시 무수정 확장**

TimescaleDB
시계열 특화 압축·쿼리 · PostgreSQL 기반

PostgreSQL
상태·제어 메타 등 **관계형** 데이터 저장

담당 역할

[실시간 데이터 파이프라인]
MQTT · Redis Stream · TimescaleDB

- **MQTT(Mosquitto)** 로 IoT 장치 telemetry 수신 → **Redis Stream** 으로 발행
- **Redis Consumer Group** 으로 서비스별 독립 소비 구조 구현
- **TimescaleDB** 배치 쓰기 — 1초 윈도우 집계로 INSERT 부하 최소화
- 처리 완료(ACK)를 저장 이후로 설계해 장애 시 데이터 유실 방지

[전력 수급 계산 로직]
Python 인메모리 그래프 모델링

- 전력망 토폴로지를 **그래프 자료구조로 모델링** (노드=자원, 엣지=선로/스위치)
- 장치 상태와 결합해 **구역별 공급 가능량·부족분 계산**
- 1홉 직접 연결만 공급원으로 인정해 **물리적으로 도달 가능한 전력만 집계**
- 토폴로지를 HTTP로 수신·캐싱하고, 장애 시 직전 값으로 폴백

[우선순위 룰 엔진]

- 안전·ESS·디젤·태양광·부하 **5단 룰 엔진** 설계
- **Safety 규칙 최상위 고정** — 임계값 도달 시 즉시 개입
- 장치별 명령 충돌 시 최고 우선순위만 선택

EMS 구현

AIEMS

마이크로그리드를 제어하는 EMS 백엔드 — 4개 서비스를 이벤트 버스로 엮은 **백엔드 단독** 설계

마이크로그리드 — 제어 대상 (시뮬레이터)

Solar	ESS	Diesel	Load
×2	×4	×2	×3

선로·스위치로 연결된 **다중 구역(zone)**— 단선 시 구역 격리

▲▼ MQTT · telemetry ↑ / 제어 명령 ↓

EMS 백엔드 — 단독 설계·구현

- Ingestion — 기능 1
- State Processor
- Control — 기능 2
- DB Writer

서비스 간 직접 호출 없이 **Redis Stream 이벤트 버스**로 연결

▽

실시간 대시보드
Socket.IO 실시간 푸시 · 상태/경보 시각화

4종 · 11대
자원 유형 · 장치

0.1~1초
장치별 발행 주기

다중 구역
토폴로지 기반 제어

IN — Ingestion 기능 1

MQTT telemetry 수신·정규화 후 **1초 윈도우**로 통일해 메시지 버스로 중계

ST — State Processor

장치 상태 계산 + 토폴로지 **전력 흐름** 산출, 대시보드 실시간 푸시

CT — Control 기능 2

우선순위 **Rule Engine**으로 자원 동원·안전 판단 후 MQTT 명령 발행

DB — DB Writer

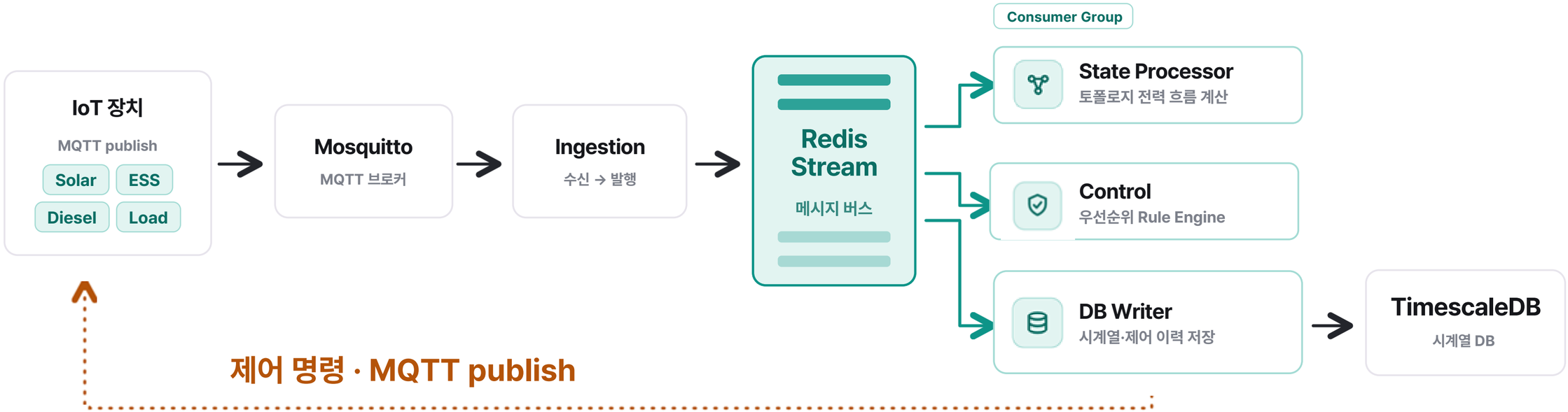
모든 저장을 **단일 게이트**로 통합 — 1초 배치 집계 · 이벤트 로깅

구현 기능 1

AIEMS

실시간 텔레메트리 수집·중계 파이프라인

IoT 장치가 종류별로 각각 0.1초 ~ 1초씩 telemetry를 발행하는데
발행 속도가 빠른 장치는 이벤트 버스를 점유하고,
서비스가 서로 직접 연결되면 한 곳의 장애·지연이 전체로 번질 위험이 있습니다.



IoT 장치가 MQTT로 발행한 데이터를 Ingestion이 정규화해 메시지 버스(Redis Stream) 로 중계하고,
State Processor · Control · DB Writer가 각자 필요한 데이터만 독립적으로 소비하도록 설계해
한 서비스의 장애나 지연이 다른 서비스로 전파되지 않는 구조를 구현했습니다

구현 기능 2

AIEMS

우선순위 Rule Engine — 10초 주기마다 자원을 평가·동원하는 제어 의사결정 엔진

100 SAFETY	안전 차단	안전 한계 도달 시 즉시 정지
50 ESS	배터리(ESS)	충·방전으로 수급 우선 조정
30 DIESEL	디젤 발전	부족분 디젤 발전 보충
20 SOLAR	태양광	잉여 태양광 활용
— LAST	부하 차단	최후 수단 · 부하 차단

숫자가 곧 자원 동원 순서 – ESS로 부족분을 메울 수 있으면 디젤 동작 X

동일 장치 충돌 시 최고 우선순위 명령만 선택
토폴로지 정보 없을 땐 Fallback 로직으로 항상 제어

1. **전력 흐름 계산** — 구역별 공급·부하·부족분(net·deficit) 산출
2. **5개 룰 평가** — 각 룰이 담당 자원을 보고 제어 명령 후보 생성
3. **충돌 해소** — 한 장치에 명령이 겹치면 최고 우선순위만 채택
4. **안전 룰 우선** — 임계 위반 시 다른 판단 무시하고 failsafe 발행
5. **명령 발행** — 최종 명령을 MQTT로 장치에 전달

룰	동작 / 켜는 조건	멈추는 조건
Safety	—	SOC·연료·온도 임계 시 **즉시 정지**
ESS	부족 + SOC 여유 → **방전**	잉여 + SOC 낮음 → **충전**
Diesel	구역 부족 + ESS 불가 → **기동**	잉여 + 최소운전(300s) 경과
Solar	잉여 태양광 활용	—
Load	—	전 수단 소진 시 **부하 차단**

트러블 슈팅

AIEMS

	1. ACK를 영속화보다 먼저 처리해 데이터 유실	2. 같은 경보가 매 제어 주기마다 반복 누적
상황	- 정상 처리된 센서 데이터가 간헐적으로 **DB에 누락** - 에러 로그 없이 정상 처리로 집계 → 원인 추적이 어려운 유실	- 상태가 지속되면 **10초 제어 주기마다** 똑같은 경보 발생 - 알림 로그가 **같은 내용으로 중복**
원인	- 메시지 수신 시 메모리 버퍼에 담고 **즉시 "처리 완료" 표시** - 실제 DB 저장은 **1초 주기로 모아 일괄 실행** - 적재~저장 사이 장애 시, 메시지는 이미 완료 처리돼 재전송되지 않고 저장만 누락	- 제어 로직이 **매 주기마다 상태를 검사** - 위반 상태가 **지속되는 동안 매번 새 경보** 생성 → 같은 사건이 수십 번 기록
해결	- "처리 완료" 시점을 버퍼 적재가 아닌 **DB 저장 완료 시점**으로 이동 - **저장에 성공한 메시지만** 모아서 완료 처리 - 서버가 중간에 죽어도 저장 안 된 메시지는 **다시 받아 처리**	- 경보 1회 발송 시 Redis에 **"이미 보냄" 표시**, 표시가 있으면 건너뜀 - 상태가 정상 회복되면 이력 제거 → 재발 시 다시 발행 - 서버 재시작에도 Redis 표시가 남아 **중복 발송 방지**
성과	**0 건** 크래시 시 데이터 유실 최대 1초분 유실 가능 → 구조적으로 0건 보장	**99% ↓** 동일 경보 발생량 1시간당 360건 → 2건 (발생 1 + 해소 1)

회고

AIEMS

MSA의 이점을 살리면서, 팀 상황에 맞춰 정합성을 우선한 설계

단일 게이트 구조



모든 DB 쓰기를 한 곳으로

순서·중복 방지 + 1초 집계 정확성

정합성	가용성	확장성
확보	양보	양보

왜 단일인가

서비스마다 DB를 두는 순수 MSA는 운영 비용이 큽니다.
그래서 독립 배포·비동기 통신 같은 MSA의 핵심 이점은 유지하되,
데이터 저장만 단일 게이트로 모아 순서·중복 문제와 1초 집계의 정확성을 확보하는 방향을 선택했습니다.

함께 고려한 트레이드오프

- **가용성** — 멈추면 전체 저장이 정지하는 단일 장애점
- **확장성** — 집계 상태가 메모리에 있어 인스턴스를 늘려 분산 불가
- **Redis Stream이 단기 장애는 흡수 (길어지면 메모리 압박·지연)**

이후 개선 방향

- **디바이스 단위 파티셔닝** — 정합성 유지하며 수평 확장
- 집계 상태를 메모리가 아닌 **Redis에 보관** → 인스턴스 교체 시에도 이어받기

한정된 자원 속에서 우선순위를 판단하는 법을 배웠습니다

선택의 트레이드오프와 개선 방향까지 함께 고민하게 됐습니다

개요

꿀잡

꿀잡 2026.02 ~ 2026. 03 (인원 : 6명)

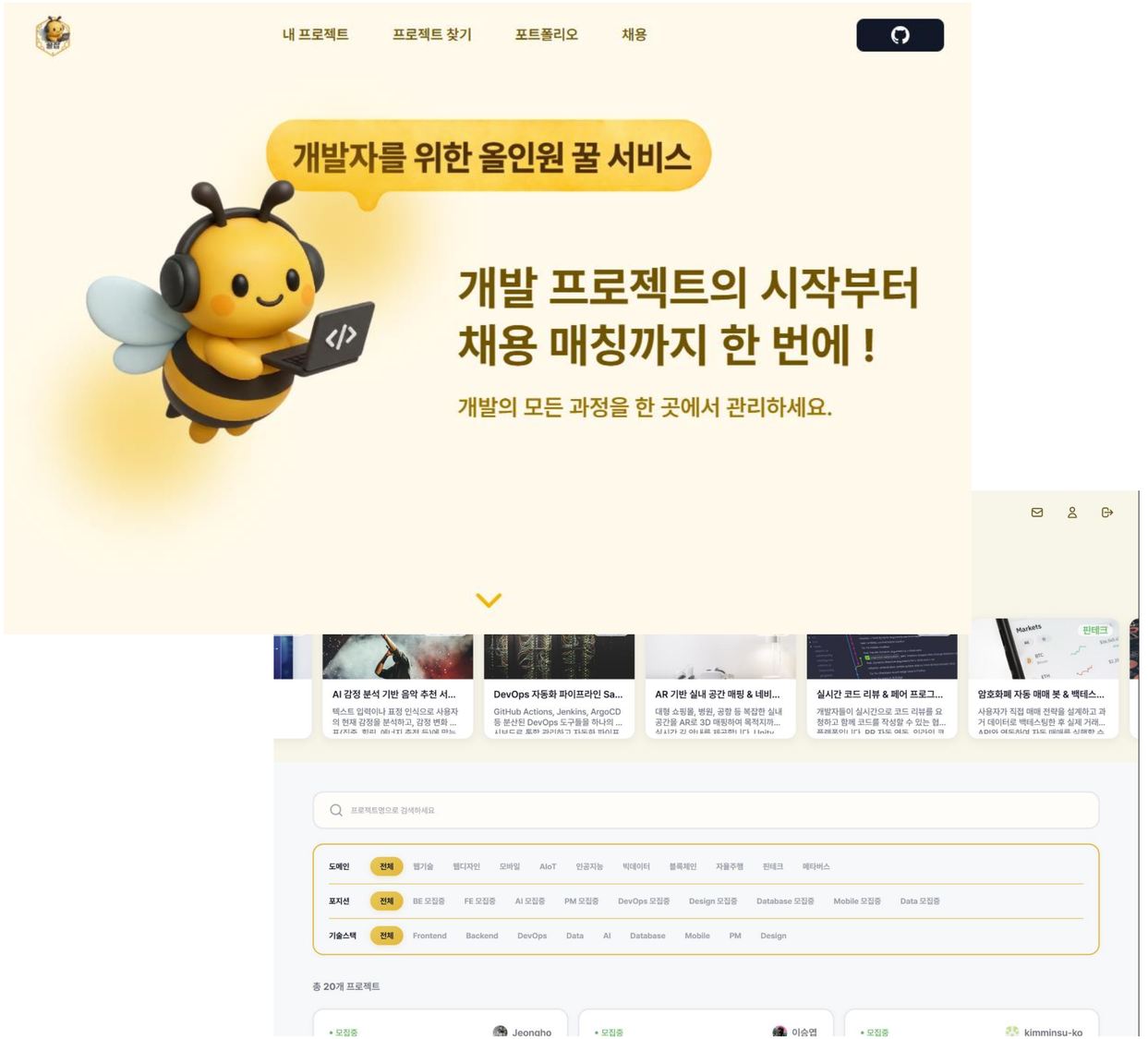
▢ 프로젝트 개요

개발 직군 취업준비생을 위한

AI 기반 올인원 채용 로드맵 플랫폼

주요 기능

- AI 채용공고 추천
- 프로젝트 탐색, 팀 빌딩
- 팀원 추천
- 트러블 슈팅 정리
- 포트폴리오 관리



기여도 · 사용 기술 · 담당역할

꿀잡

사용 기술

Neo4j

스킬 그래프 + 벡터 ANN을 **단일 Cypher**로 통합

Python OpenAI Embedding

임베딩 배치 · 크롤러 · 커트라인 **사전 계산**

Docker · Nginx · Jenkins

7개 컨테이너 · HTTPS · **Webhook 자동 배포**

담당 역할

[인프라 · CI/CD 자동화]

- Docker 멀티 컨테이너 7개 운영
- Jenkins 4개 파이프라인 자동 배포
- Blue/Green 무중단 배포 구축

[추천 백엔드 설계]

- 추천 API 설계
- MySQL · Neo4j · Redis 멀티 DB
- 그래프 + 벡터 단일 쿼리로 통합

[AI 데이터 파이프라인]

- 채용공고 크롤러 6,100건 데이터 수집
- OpenAI 임베딩 배치
- 커트라인 사전 계산 캐싱

20% 개인 기여도

6인 팀 · 인프라 · 추천 · AI

구현 기능 1

꿀잡

추천 매칭 파이프라인

1 사용자 프로필 수집
스킬 · 목표 · 경험 · README

▼

2 OpenAI 임베딩 생성
프로필 → 벡터화 · 야간 배치 사전 계산

▼

3 Neo4j 통합 조회
그래프 탐색 + 벡터 ANN을 단일 Cypher로 통합 (왕복 1회)

▼

4 정규화 · 가중 합산
선형 정규화 · 가중 합산을 쿼리 레이어에 위임

▼

5 적합도 점수 · 5단계 등급
6,100개 공고 실시간 적합도 산출

핵심 설계 · 단일 쿼리 통합

스킬 그래프 탐색

벡터 ANN 검색

▼ 통합

단일 Cypher 쿼리

Neo4j 왕복 2회 → 1회

적합도 5단계 등급

최적합 | 적합 | 보통 | 미흡 | 부족

사용자 분포 백분위 기반 상대평가

구현 기능 1

꿀잡



Neo4j는 매칭 연산, MySQL은 메타데이터·정렬, Caffeine는 캐시로 역할을 분리

하나의 추천 API가 각 계층을 단일 흐름으로 조율합니다.

트러블 슈팅

꿀잡

1. 추천 API 응답 속도 단계적 개선

상황

약 6,100개 공고를 처리하는 추천 API 응답이 **1.4초까지 지연**

원인

스킬 그래프 탐색과 벡터 ANN 검색을 앱에서 별도 호출 → **Neo4j 왕복 2회**
정렬(적합도·마감순)을 바꿀 때마다 동일 스코어링을 재실행 → **가장 비싼 연산 반복**

해결

쿼리 통합 : 그래프 + 벡터 검색을 단일 Cypher로 통합(왕복 1회), 정규화·가중 합산을 쿼리 레이어로 위임
연산 분리 : 스코어링 결과를 Caffeine 캐시(TTL 1h)에 저장, 단순 정렬은 MySQL ORDER BY로 분리

성과

95% ↓ **추천 API 응답 시간**
1.4초 → 0.07초 · 병목 진단 후 구조 재설계

2. 공고마다 다른 점수 분포를 백분위 커트라인으로 정규화

상태가 지속되면 **10초 제어 주기마다** 똑같은 경보 발생
알림 로그가 **같은 내용으로 중복**

제어 로직이 **매 주기마다 상태를 검사**
위반 상태가 **지속되는 동안 매번 새 경보** 생성 → 같은 사건이 수십 번 기록

경보 1회 발송 시 Redis에 **"이미 보냄" 표시**, 표시가 있으면 건너뜀
상태가 정상 회복되면 이력 제거 → 재발 시 다시 발행
서버 재시작에도 Redis 표시가 남아 **중복 발송 방지**

5 단계 **공고별 상대 등급**
절대 기준 → 상대평가 전환, 상위 백분위(1~99%) 동시 산출

회고

꿀잡

데이터 수집부터 추천까지 설계 · 구현 하면서 배운 점

01
크롤링
채용공고 수집

→

02
정제
표기 통일 · 노이즈 제거

→

03
적재
MySQL · Neo4j · 벡터

→

04
추천 API
스킬 매칭 + 의미 유사도

성능을 지키기 위한 핵심 설계 결정 3가지

무거운 계산을 어디서 할까?

계산은 DB·배치에 맡기고, 앱은 조립만

점수·등급·백분위 같은 무거운 계산을 앱이 아니라 **Neo4j 쿼리와 사전 배치** 로 옮겼습니다. 사용자 요청 때는 이미 계산된 값을 **조립만** 하면 돼 실시간 응답이 가벼워집니다.

실시간 경로 최소화

두 개의 DB를 어떻게 합칠까?

Neo4j·MySQL을 한 번의 조회로 결합

추천 점수는 Neo4j, 공고 원본은 MySQL에 있습니다. 하나씩 조회하면 요청이 폭증(N+1)하므로, ID를 모아 **IN 절로 한 번에** 가져와 메모리에서 합쳤습니다.

DB 조회 N+1 → 1회

같은 요청을 매번 다시 계산해야 할까?

한 번 계산한 결과는 캐시로 재사용

필터 조건의 순서가 달라도 **같은 키** 로 묶이도록 정렬해 캐시(Caffeine)에 저장했습니다. 같은 조회는 다시 계산하지 않고 바로 응답합니다.

응답 0.9초 → 0.07초

개요

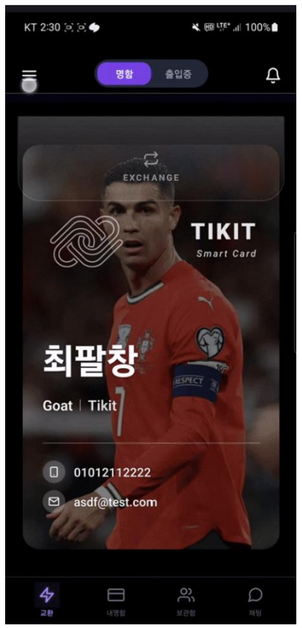
TIKIT 2025.12 ~ 2026. 01 (인원 : 6명)

▢ 프로젝트 개요

명함 OCR과 NFC 출입 권한을 결합한 모바일 명함 플랫폼

주요 기능

- 명함 교환
- 위치/검색
- OCR & 시맨틱 검색
- NFC 기기 등록 및 권한 부여





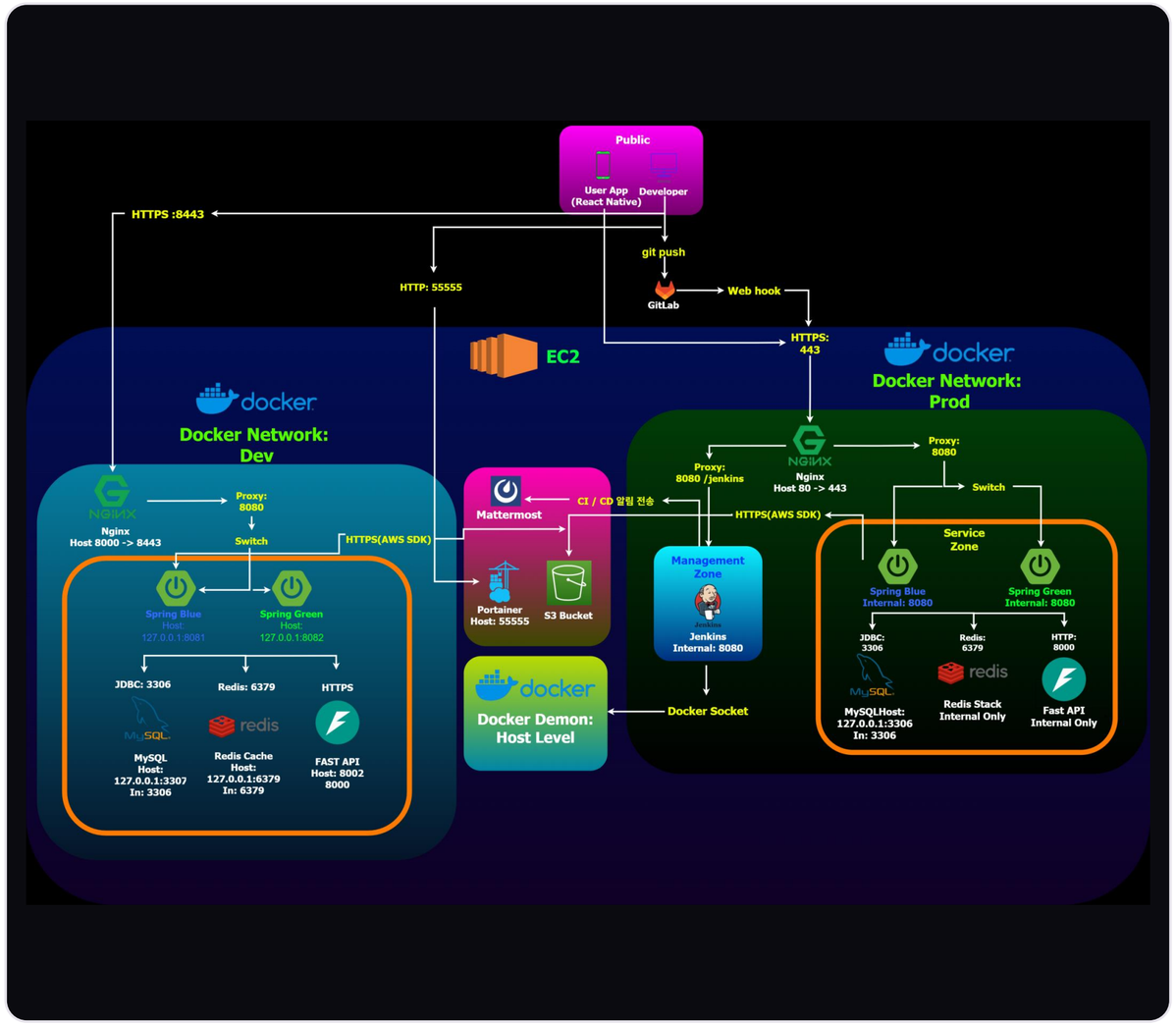




기여도 · 사용 기술 · 담당역할

TIKIT

- 인프라 아키텍처



담당 역할

[인프라 · 무중단 배포 구축]

- 전체 서비스 Docker 컨테이너화, Dev/운영 분리
- Blue/Green 무중단 배포, 설계, Nginx 동적 업스트림 전환
- Jenkins CI/CD 자동화 · Webhook 빌드 · Mattermost 알림

[AI 시맨틱 검색 인프라]

- Redis Stack 벡터 검색(HNSW) 구성
- 명함 사진 임베딩 -> 시멘틱 검색 파이프라인
- **3072차원 · COSINE 인덱스 설계**

[외부 서비스 연동]

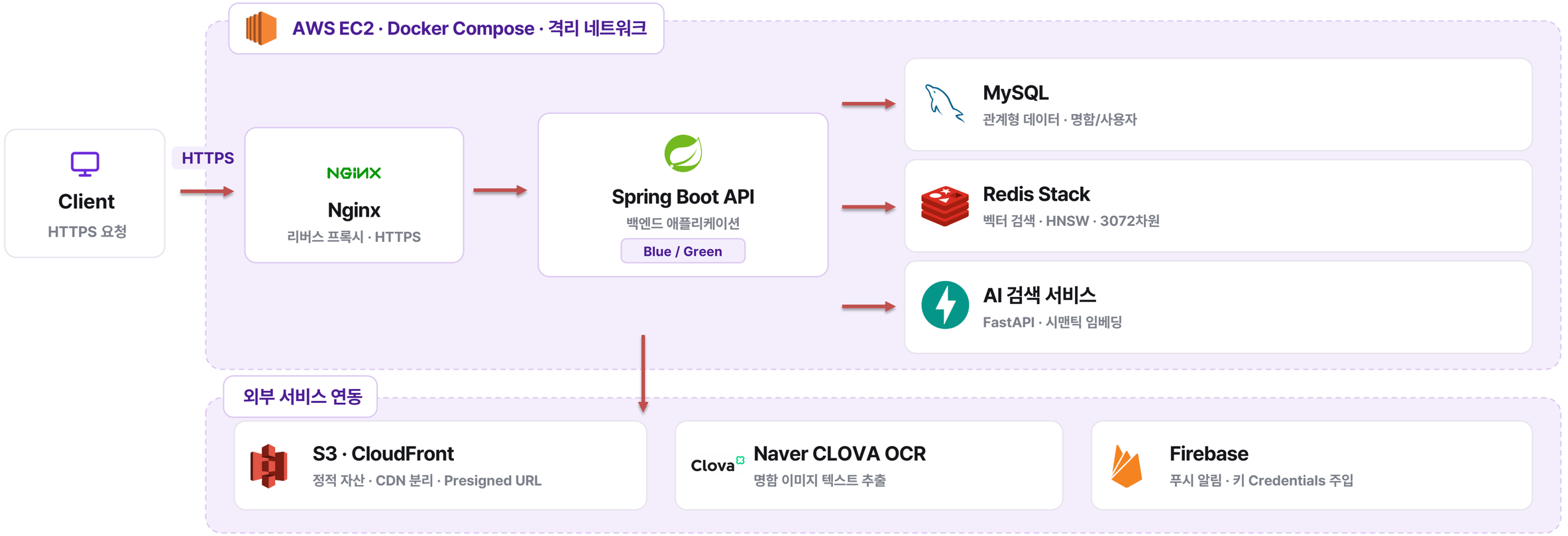
- **AWS S3 · CloudFront** · CDN 분리 · Presigned URL
- **CLOVA OCR** 연동 · 명함 이미지 텍스트 추출
- **Firebase 푸시 알림 연동**

20% 개인 기여도

6인 팀 · 인프라

구현 기능 1

TIKIT

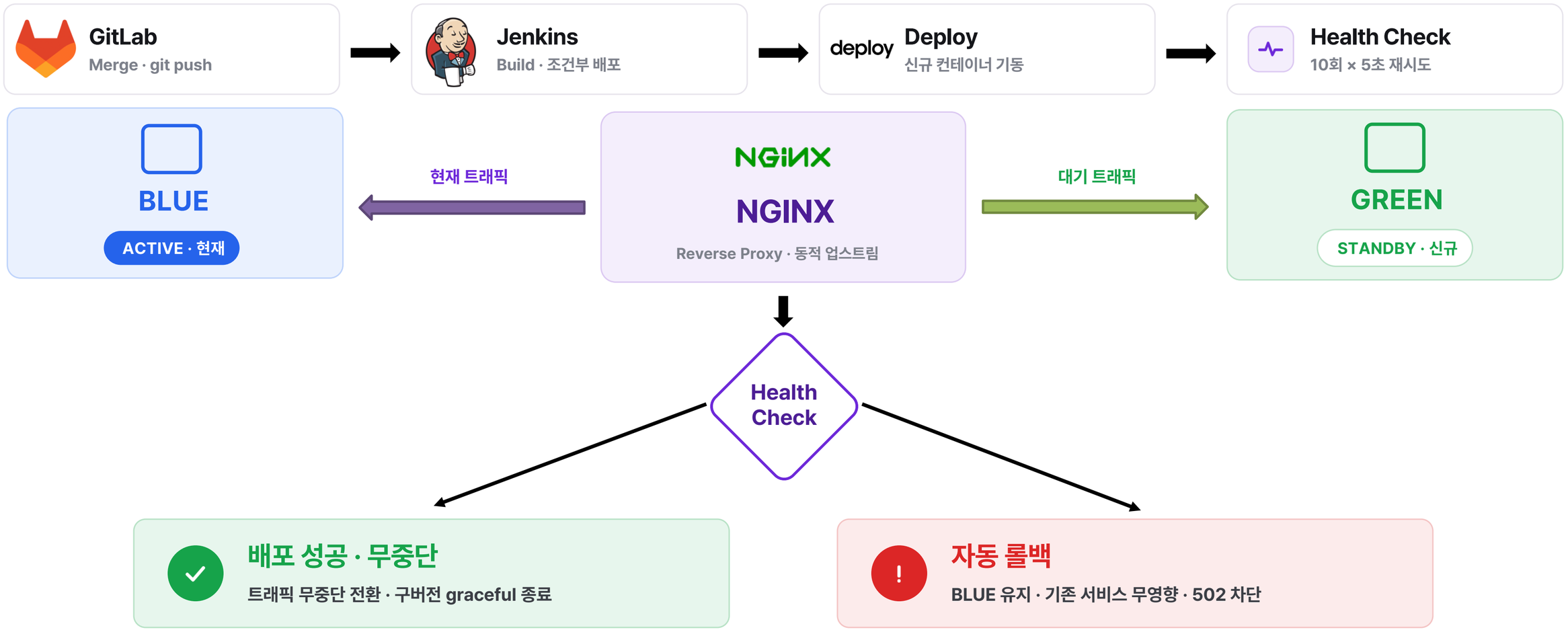


단일 EC2 위에 Nginx · Spring · 멀티 DB를 컨테이너로 격리 구성하고, 외부 서비스는 느슨하게 연동

한 대의 인스턴스로 비용과 운영 복잡도를 동시에 통제했습니다.

구현 기능 2

TIKIT



헬스체크를 통과한 GREEN으로만 트래픽을 전환

서비스 중단 없이 배포하고, 장애 시 자동 롤백되도록 설계했습니다.

트러블 슈팅

TIKIT

	1. 무중단 배포 중 헬스체크 실패 — 502 오류	2. AI 서비스 불필요한 전체 재빌드
상황	신규 컨테이너 기동 직후 응답 전 Nginx가 트래픽 전환 → 일부 요청 **502 오류**	백엔드 코드만 변경해도 AI 서비스가 항상 **재빌드·재배포** → 빌드 시간 증가
원인	Spring Boot 초기화(약 15~20초) 대비 **헬스체크 타임아웃이 너무 짧음**	변경 범위와 무관하게 **모든 서비스를 순차 재빌드**하는 파이프라인 구조
해결	재시도 **10회 × 5초** · 통과 후에만 전환하도록 엄격히 순서화	AI 코드 변경 감지 로직 추가 → 변경 시에만 재배포, 그 외 **스킵**
성과	**0** 건 502 오류 해소	**99%** ↓ 배포된 코드 감지 후 Docker 재빌드

회고

TIKIT

프로젝트 성과

무중단 배포 달성

Blue/Green 전략으로 **서비스 중단 없이** 배포 · 헬스체크 기반 안전한 전환

자동화 파이프라인

GitLab → Jenkins → Blue/Green 전 과정 자동화 · AI **조건부 배포**

벡터 검색 인프라

Redis Stack 명함 시맨틱 검색 · **3072차원 / COSINE**

보안 · 빌드 최적화

운영 포트 외부 차단 · 키 Credentials 주입 · **멀티스테이지** 이미지 최소화

프로젝트 회고

아쉬웠던 부분

단일 EC2에 전체 스택 운영 → **리소스 경합** · 서비스 분리 설계 필요

개선 방안

AI 서비스 별도 인스턴스 분리 또는 **컨테이너별 리소스 제한**으로 안정성 확보

학습 결과

인프라는 부가 작업이 아니라 **서비스 안정성과 생산성을 결정하는 핵심 설계**

깨달은 점

배포 파이프라인을 잘 만들면 **팀 전체의 개발 속도가 달라진다**