TECHNICAL PORTFOLIO · SPOTLIGHT

김현욱 대표 기술 포트폴리오

데이터 수집, DB/인프라, on-premise 동기화, 보안, CI/CD 운영 개선을 중심으로 대표 항목 9개를 한 장씩 읽히는 A4 문서 구조로 정리했습니다. 웹 상세 페이지의 카드형 화면을 캡처하지 않고, PDF 열람에 맞게 핵심 배경·작업·임팩트·역할을 재배치했습니다.

Data / Systems Engineer | 2019–2026 | 대표 항목 9개 | portfolio-equ.pages.dev | hyunwook711@naver.com | github.com/hyunwook711

범위
Data Platform · DB/Infra · on-premise delivery · Security · CI/CD

구성
대표 항목 9개를 각 1페이지 문서로 재구성

읽는 법
각 항목은 배경 → 역할 → 핵심 작업 → 임팩트 순서로 정리

포함 항목

01 · DATA PLATFORM ⓪ · 2022–23

ETL DB 영역 분리 — 수집/배포/PROD 경계 재설계

수집·배포·서비스가 한 DB에 묶여 수집 과정의 데이터가 고객사까지 그대로 흘러가던 구조를, 수집 DB와 배포 DB(메타·파일 2계열)로 나누고 바이너리 로그 반영 순서를 잡아 PROD 정합성을 확보한 재설계

기간	소속	역할
2022–2023	래브라도랩스(LabradorLabs)	Data Engineer · 구조 재설계

MySQL · Binary Log · ETL · DB 아키텍처 · Medallion Architecture

작업자의 메모

고객사로 나갈 데이터와 만드는 중인 데이터를 같은 그릇에 두지 않는다 — 이 작업은 그 한 문장에서 출발했습니다. 수집이 실패하거나 다시 돌 때마다 그 흔적이 배포본에 묻어가면, 문제가 생겨도 어디서부터 손대야 할지 보이지 않으니까요.

배경

초기에는 ETL 전 과정이 하나의 DB에서 이루어졌습니다. 수집 작업과 배포용 데이터, 서비스 PROD까지 한 곳에 있다 보니 수집 과정의 중간 데이터까지 고객사로 그대로 전달되는 구조였습니다.

역할

Data Engineer로서 영역 분리 설계와 배포 DB 2계열 분류, binlog 반영 경로 재구성을 수행했습니다. 운영 중인 서비스와 고객사 동기화에 직접 닿는 변경이라, 경계를 먼저 세운 뒤 데이터 흐름을 옮기는 순서로 진행했습니다. 이때 세운 수집/배포 DB 경계는 지금의 데이터 플랫폼에도 그대로 살아 있습니다.

핵심 작업

- 한 DB에 섞여 있던 ETL을 수집 담당 영역과 배포 담당 영역으로 분리 — 수집 작업의 시행착오가 배포 데이터에 닿지 않는 경계를 먼저 세움
- 배포 DB를 오픈소스 메타정보(취약점·라이선스·버전 등)와 파일 정보 2계열로 분류 — 파일 데이터가 참조하는 메타정보가 항상 먼저 존재하도록 의존 방향을 정리
- 두 계열의 바이너리 로그가 메타정보 → 파일 정보 순서로 PROD DB에 반영되도록 전달 경로를 재구성
- 수집 과정에서만 쓰이는 불필요한 데이터를 PROD에서 제거 — 고객사로 나가는 DB를 배포에 필요한 것만 남게 슬림화

임팩트

- 정합성 확보 — 메타 먼저, 파일 나중이라는 반영 순서가 구조로 보장되어 참조가 깨지는 문제를 차단
- PROD 슬림화 — 수집용 중간 데이터를 걷어내 고객사로 나가는 DB에 배포에 필요한 데이터만 유지
- 책임 경계 — 수집 영역의 변경·실험이 배포와 PROD에 번지지 않는 구조적 격리의 출발점

웹 상세: https://portfolio-equ.pages.dev/items/etl-db-separation.html

02 · INFRA COST · 2024–26

AWS → IDC/In-house 인프라 이전 & 클라우드 비용 절감

AWS에서 운영하던 데이터 백엔드·수집 인프라를 자체 IDC/In-house 서버로 이전하고 검증을 거쳐 전환한 뒤 AWS 자원을 종료해 반복 과금을 끊은 작업

기간
2024–2026 (AWS 자원 종료 ~2026-04)

소속
래브라도랩스(LabradorLabs)

역할
데이터 엔지니어 · 인프라

AWS | EC2 / EFS / RDS | S3 | IDC / In-house | Elasticsearch | MySQL Replication

작업자의 메모

클라우드 비용을 줄이는 일은 숫자만 보면 단순해 보이지만 실제로는 옮긴 뒤에도 같은 데이터가 같은 방식으로 서비스되는지 확인하는 작업이 더 컸습니다. 검색엔진 인덱스와 DB 데이터를 검증 가능한 경로로 옮긴 이유도 그 때문입니다.

배경

데이터 백엔드와 수집 인프라가 AWS(EC2 / EFS 등) 위에서 운영되면서 클라우드 비용 부담이 누적됐습니다. 자원을 직접 통제하기도 어려운 구조였습니다.

역할

데이터 엔지니어 · 인프라 담당으로서 데이터 이송 → 정합성 검증 → 자체 인프라 재구성 → 전환·AWS 종료를 단독으로 수행했습니다. DB primary/replica·수집 서버·모니터링을 자체 환경에 재구성하고 네트워크·백업·복원 체계를 정비해 반복 비용을 줄이면서 데이터 인프라를 직접 통제하는 구조를 만들었습니다.

핵심 작업

- ① 이송S3 경유로 자체 서버에 데이터 적재
- ② 검증인덱스 문서 수 일치 등 정합성 확인
- ③ 재구성DB primary/replica·수집·모니터링 구축
- ④ 전환·종료전환 완료 후 AWS 자원 종료

임팩트

- 비용 절감 — 전환 완료 후 AWS 자원 종료로 반복 과금 중단 (예: AWS K8s 클러스터의 EC2 8대 + EFS 스토리지 과금 중단)
- 운영 통제권 — 데이터 인프라를 자체 환경에서 직접 운영·통제할 수 있는 구조로 전환
- 안전 전환 — 대용량 데이터를 정합성 검증 후 무손실로 이전

웹 상세: https://portfolio-equ.pages.dev/items/aws-to-idc-migration.html

03 · CUSTOMER DELIVERY · 2024–26

on-premise 데이터 동기화 배포 & 심층 기술지원

외부에서 제공되는 replica 연결을 보안상 허용하기 어려운 고객사 환경을 위해, on-premise DB 백업본과 바이너리 로그 기반 데이터 동기화 체계를 맞춤 배포하고 심층 이슈를 해결한 작업

기간
2024–2026 (상시)

소속
래브라도랩스(LabradorLabs)

역할
데이터 엔지니어 · 2026년부터 데이터 엔지니어링 리드

`on-premise` `MySQL` `Binary Log` `데이터 동기화` `중계 경로` `폐쇄망`

작업자의 메모

고객사 on-premise 환경은 보안 정책이 다 달라서 하나의 정답 구성을 밀어 넣기 어렵습니다. 그래서 기본형, 파일 검수, 시간 제한, 폐쇄망, 중계 경로처럼 조건별 구성을 나눠야 했습니다.

배경

제품 데이터는 중앙 운영 DB에서 만들어지고 고객사는 이 데이터를 자사 인프라 안에서 직접 운영해야 합니다. 운영 DB의 replica를 직접 제공·연결하는 방식은 고객사 보안 정책상 허용되기 어려워 고객사 내부에 DB를 두고 초기 백업본 + 바이너리 로그 변경분으로 동기화하는 별도 구조가 필요했습니다.

역할

데이터 엔지니어로서 on-premise DB 백업본과 바이너리 로그 기반 데이터 동기화 체계의 설계·구현·운영을 단독으로 수행했으며 고객사 환경별 배포와 심층 기술지원을 책임졌습니다. 1차 지원은 다른 인원이 맡았지만 시스템을 가장 잘 아는 담당자로서 다른 손에서 풀리지 않는 이슈의 최종 해결자 역할을 수행했습니다. 2026년부터는 데이터 엔지니어링 리드로 전환 작업과 지원 표준화를 함께 이끌었습니다.

핵심 작업

- 환경별 맞춤 설치·구성 — 고객사 환경 제약에 맞춰 데이터 동기화 모듈을 시나리오별(기본·파일 검수·시간 제한·폐쇄망·중계 경로)로 설치·구성
- 설치 형상관리 — 고객사별 DB 백업본 버전 · 동기화 모듈/프록시 버전 · 설치일을 추적해 환경 차이를 체계적으로 관리
- 심층 기술지원 (최종 해결자) — 기술지원팀이나 외주가 1차로 해결하지 못하는 DB · 바이너리 로그 기반 동기화 · 동기화/정합성 · 네트워크 이슈를 시스템 담당자로서 직접 진단·해결
- 고객사 현안 대응 — 업로드/반영 실패, 동기화 지연, TLS·네트워크 오류, 분석 결과 오탐처럼 제품 데이터와 고객사 환경이 만나는 지점의 문제를 로그·DB 상태·배포 형상까지 연결해 추…

임팩트

- 안정적 배포·동작 — 환경이 제각각인 고객사 on-premise에서 DB·동기화 구성의 안정 배포와 동작을 책임
- 끝까지 해결 — 1차 지원에서 풀리지 않는 심층 문제를 시스템 담당자로서 최종적으로 해결
- 현안 추적 — 고객사별 환경 차이와 오류 유형을 문서·이슈로 남겨 재현, 재처리, 패치 검증까지 이어지게 함
- 표준화·온보딩 — 가이드·시나리오·등록 절차 문서화로 지원 품질을 표준화하고 인수인계를 수월하게

웹 상세: https://portfolio-equ.pages.dev/items/onprem-binlog-shipping-support.html

04 · DB/INFRA · 2025

DB 인덱스 최적화 & 용량 53% 절감

수년간 운영되며 비대해진 중앙 운영 DB를, 실사용 쿼리 분석 기반으로 불필요 인덱스를 걷어내고 SQL을 튜닝해 용량과 성능을 동시에 개선

기간	소속	역할
2025	래브라도랩스(LabradorLabs)	데이터 엔지니어 · 인덱스 진단 및 최적화 주도

MySQL · EXPLAIN · Index Tuning · SQL Optimization · Query Analysis

작업자의 메모

이 작업에서 제일 조심한 부분은 “인덱스를 많이 지웠다”가 아니라 “지워도 되는지 증명했다”였습니다. 운영 DB에서 감으로 인덱스를 건드리는 순간 성능 문제는 더 커질 수 있기 때문입니다.

배경

수년간 운영되어 온 중앙 운영 DB는 그동안 누적된 미사용 인덱스와 비효율 쿼리로 인해 저장 용량이 비대해지고 전반적인 성능이 저하된 상태였습니다.

역할

데이터 엔지니어로서 쿼리 수집 → 실행 계획 분석 → 인덱스 정리 → SQL 튜닝의 전 과정을 단독으로 수행했습니다. 운영 중인 중앙 운영 DB를 다루는 만큼, 실제 사용 패턴을 근거로 안전하게 변경 범위를 좁혀가며 진행했습니다.

핵심 작업

- 실제 사용되는 쿼리 95개를 EXPLAIN으로 분석해 전체 인덱스 311개 중 224개(72%)가 실행 계획에서 전혀 쓰이지 않음을 정량적으로 증명
- 불필요 인덱스를 삭제하고 복합 인덱스의 컬럼 순서를 실제 실행 계획에 맞게 재구성
- SQL 튜닝: OR 조건을 UNION으로 전환하고 LIKE 와일드카드 패턴을 인덱스가 타도록 최적화

임팩트

- DB 용량 53.8% 절감 — 2.6TB → 1.2TB
- 초기 설치 시간 절반 — DB 생성 12시간 → 6시간
- 비용 절감 — 저장 자원 축소로 운영 비용에 직접 기여
- 서비스 품질 향상 — 쿼리 효율 개선으로 응답 성능 개선

웹 상세: https://portfolio-equ.pages.dev/items/db-index-optimization.html

05 · DATA PLATFORM ① · 2026

ETL 데이터 수집 플로우 재정립

재수집 때마다 1,000만~1억 건의 외부 API 요청이 필요하던 원테이크 수집을, RAW 보존·재파싱 구조의 오브젝트 스토리지 기반 파이프라인으로 재설계

기간
2026 (3월~ 진행)

소속
래브라도랩스(LabradorLabs)

역할
데이터 엔지니어링 리드 · 파이프라인 재정립 주도

ETL · ELT · Data Pipeline · Object Storage · SeaweedFS · DB / Infra

작업자의 메모

원테이크 수집은 처음에는 단순하지만 재수집이 필요해지는 순간 외부 API 비용과 실패 위험이 그대로 다시 옵니다. 1,000만~1억 건 요청을 반복해야 하는 구조라면 파이프라인부터 바꾸는 편이 맞았습니다.

배경

제품 데이터는 수집(Gathering) → 정제(Curation) → 배포(Distribution)로 이어지는 파이프라인으로 운영됩니다. 단계별로 처리 방식과 저장 위치가 제각각이라, 흐름을 한눈에 파악하기 어렵고 데이터 규모가 늘수록 확장에 부담이 있었습니다.

역할

데이터 엔지니어링 리드로서 파이프라인 재정의 → 스토리지 설계 → 마이그레이션 전략 수립을 단독으로 수행했습니다. 수집부터 배포까지 전체 흐름을 표준화하고 파일시스템에서 오브젝트 스토리지로 옮기는 로드맵을 세워 확장 가능한 데이터 기반을 마련하고 있습니다.

핵심 작업

- RAW 데이터를 오브젝트 스토리지에 원본 보존하고 상태 관리 DB로 수집(Collector)과 정제(Worker)를 분리 — 재처리는 외부 요청 없이 재파싱으로 해결
- Cache Hit/Miss 패턴(이미 있으면 즉시 반환)과 version_id 기반 수정 이력 추적으로 데이터 재현 가능성 확보
- 오픈소스 RAW 데이터 수집용 오브젝트 스토리지 설계
- 파일시스템 → 오브젝트 스토리지(SeaweedFS) 마이그레이션 전략·로드맵 수립

임팩트

- 외부 요청 제거 — 재수집 때마다 반복되던 1,000만~1억 건의 API 호출이 재처리 시 0건으로 (RAW 재파싱)
- 재처리 시간 단축 — rate limit에 묶여 수일씩 걸리던 기간 재수집을 내부 재파싱으로 대체 (설계 산정 예: 2일 → 5분)
- 추적·재현 가능 — version_id 이력 추적과 원본 보존으로, 이상 데이터의 원인 규명·재현이 가능
- 수집 플로우 표준화 — 수집→정제→배포 책임 명확화, 신규 소스 편입·운영 인계 용이

웹 상세: https://portfolio-equ.pages.dev/items/etl-flow-redesign.html

06 · DATA PLATFORM ② · 2026

오픈소스 RAW 데이터 오브젝트 스토리지 구축 (SeaweedFS)

오픈소스 생태계의 메타·바이너리·Git 원본 30억~60억 건을 외부 의존 없이 재현 가능하게 보존하는 자체 미러 인프라 설계·구축

기간	소속	역할
2026-05 (설계) ~ 06 (구축)	래브라도랩스(LabradorLabs)	데이터 엔지니어링 리드 · 설계·구축 주도

`SeaweedFS` `S3` `Data Lake` `Haystack` `XFS` `ZFS` `Forgejo`

작업자의 메모

외부 API와 원천 저장소가 계속 살아 있을 것이라고 가정하면 수집 시스템은 편해 보입니다. 하지만 오픈소스 데이터는 사라지거나 바뀌기 때문에, 원본을 내부에서 다시 재현할 수 있어야 했습니다.

배경

오픈소스 생태계의 메타정보(npm·PyPI·Maven·Go 패키지, CVE·OSV·SPDX 라이선스), 바이너리(tarball·wheel·jar 등), Git 원본(RAW 저장소)을 대규모로 수집·보존하는 자체 미러 인프라가 필요했습니다. 원본을 스키마 강제 없이 그대로 보존해 두는 층이라는 점에서, 사내 데이터 레이크의 저장 기반에 해당합니다.

역할

데이터 엔지니어링 리드로서 저장소 분리 설계 → 기술 선택 근거 정리 → 구축 착수를 단독으로 수행했습니다. 데이터의 수정 패턴에 맞춰 오브젝트 스토리지와 파일시스템을 나누고, 작은 파일 대량 저장(SeaweedFS·Haystack)과 git 무결성·복제(ZFS·Forgejo)를 각각 최적화해 외부 의존 없는 확장 가능한 데이터 인프라 기반을 마련했습니다.

핵심 작업

- 메타 · 바이너리 → 오브젝트 스토리지 한 번 쓰고 key로 읽는(write-once, read-by-key) 패턴. SeaweedFS(S3 호환) · 오브젝트 볼륨 파일시스템 XFS
- Git 원본 → 파일시스템 잠금·부분수정·ref 갱신·index 재작성이 잦은 패턴. ZFS 위 bare 저장소 · 미러 자동화는 Forgejo Mirror에 위임
- 수집: 메타 · 바이너리 · git
- SeaweedFS S3 Haystack

임팩트

- 외부 의존 제거 — 레지스트리·API rate limit에 종속되지 않는 자체 미러 확보
- 재현 가능한 보존 — 수집 시점의 메타·바이너리·git을 재현 가능하게 장기 보관
- 대규모 수용 — 30억~60억 건 데이터를 2서버 · EC(6+2) · 58TB raw(증설 예정) 인프라로 수용하는 설계
- 확장 가능한 기반 — 작은 파일 대량 + git 갱신을 각각 적합한 저장소로 처리하는 인프라 기반 마련

웹 상세: https://portfolio-equ.pages.dev/items/object-storage-seaweedfs.html

07 · CDC · 2026

바이너리 로그 동기화 4.0 — 단순 리플레이를 CDC·멱등 UPSERT로 전환

mysqlbinlog 파이프로 파일을 통째 리플레이하던 고객사 동기화를, binlog 이벤트를 직접 파싱해 UPSERT로 변환·적용하는 Go 기반 CDC 구조로 재설계 — 재시도해도 깨지지 않는 동기화를 만든 작업

기간	소속	역할
2025.09 제안 · 2026.01–04 설계·구현·배포	래브라도랩스(LabradorLabs)	Data Engineer · CDC 전환 제안 및 설계·구현

Go · CDC · MySQL Binary Log · UPSERT · SHA256 · state.yaml

작업자의 메모

레거시 동기화의 단위는 binlog "파일"이었습니다. 파일 안의 이벤트 하나가 실패해도 파일 전체가 막히고, 어디까지 들어갔는지는 stderr 텍스트를 긁어 세는 수준이라 복구 위치를 특정할 수 없었습니다. 품질 개선 회의에서 CDC 구조 전환을 직접 제안한 이유입니다.

배경

레거시(v2/v3) Updater는 mysqlbinlog 유틸리티를 파이프로 실행해 binlog를 타겟 DB에 그대로 밀어넣는 단순 리플레이였습니다. SQL을 제어할 수 없으니 중복 키 에러가 나면 수동 개입이 필요했고, DEFINER 절·Virtual Column처럼 고객사 환경에서 깨지는 구문도 거를 수 없었습니다.

역할

2025년 9월 품질 개선 회의에서 Binlog Shipping의 파일 단위 한계를 제기하며 CDC 구조 전환을 제안했고, 2026년 1분기에 설계·구현을 맡아 4월 배포까지 진행했습니다. 비교 검토(Debezium·Maxwell·커스텀)부터 Go Updater의 파싱·변환·상태 관리 설계, SSD/HDD 환경별 v2 대비 성능 측정까지 문서로 남기며 작업했습니다.

핵심 작업

- binlog 직접 파싱 → SQL 변환 엔진을 Go로 구현 — INSERT/UPDATE를 INSERT ... ON DUPLICATE KEY UPDATE(UPSERT)로 변환해 재실행 시에도…
- 타겟 환경에 맞춘 SQL Sanitization — DEFINER 절 제거, GRANT 문 스킵, GENERATED ... VIRTUAL 컬럼은 information_schema 조회로 자…
- 무결성 검증을 MD5에서 SHA256으로 교체 — 서버가 보낸 평문 해시와 클라이언트가 복호화 후 계산한 해시를 대조해 전송 깨짐·키 오류를 즉시 감지
- Smart Polling 프로토콜 설계 — 마지막 수신 해시를 헤더로 보내 변경 없으면 304 Not Modified, 파일 생성 중이면 404 대신 202 Accepted로 "에러"와 "…

임팩트

- 정합성 — 멱등 UPSERT + 이벤트 단위 제어로 중복·누락 없이 재시도 가능, 사실상 exactly-once 적용
- 자동 복구 — 체크포인트 롤백이 내장되어 파일 손상·임포트 실패가 사람 개입 없이 다음 주기에 스스로 복구
- 처리 속도 — binlog 파일당 SSD 평균 96초 → 39초(2.56배), HDD 최대 5배. 데이터가 클수록 격차 확대

웹 상세: https://portfolio-equ.pages.dev/items/binlog-shipping-v4-cdc-upsert.html

08 · SECURITY · 2026

바이너리 로그 기반 데이터 동기화 암호화 전환

외부에서 제공되는 replica 연결을 허용하기 어려운 고객사 환경에 바이너리 로그 변경분을 전달하는 데이터 동기화 체계를 고도화했습니다. 난독화 수준이던 보호 방식을 정식 암호화로 전환해 전송 구간 보안을 강화했습니다.

기간	소속	역할
2026	래브라도랩스(LabradorLabs)	데이터 엔지니어링 리드 · 전환 주도

Binary Log · Encryption · Data Replication · Monitoring · Security

작업자의 메모

난독화는 “눈에 안 보이게” 만드는 수준이고 고객사 동기화 경로에서 필요한 것은 기밀성과 무결성을 설명할 수 있는 암호화였습니다. 그래서 AES-256-GCM 전환은 기능 개선보다 책임 경계를 분명히 하는 작업에 가까웠습니다.

배경

제품 데이터는 고객사 내부 DB가 초기 백업본을 보유하고, 운영 DB의 바이너리 로그 변경분을 받아 반영하는 경로로 동기화됩니다. 이 구조는 외부 replica 연결을 두기 어려운 보안 제약 때문에 필요했고 이전 버전에서는 전송 데이터를 스크램블링(난독화) 방식으로만 보호하고 있어 전송 구간 보안 수준을 더 끌어올릴 필요가 있었습니다.

역할

데이터 엔지니어링 리드로서 바이너리 로그 기반 데이터 동기화 체계의 암호화 전환과 모니터링 구축을 기획부터 설계·구현·운영까지 단독으로 수행했습니다. 보호 방식 전환의 설계 방향을 잡고 고객사 환경으로 나가는 데이터의 보안과 가시성을 함께 끌어올렸습니다.

핵심 작업

- 전송 암호화는 표준 알고리즘 AES-256-GCM을 채택해 기밀성과 무결성(Tag 검증)을 동시에 보장
- 암·복호화 키는 코드에 두지 않고 빌드 시 임베딩하며, DB 접속 계정 정보도 암호화해 자격 증명 노출을 차단
- 단순히 binlog 전체를 그대로 적용하던 방식에서, binlog→SQL 변환(Custom Parser) + upsert 변환으로 바꿔 중복 적용 에러를 제거
- 적용 실패 시 트랜잭션 재시도 메커니즘으로 일시적 오류에 견디도록 보강

임팩트

- 전송 보안 강화 — AES-256-GCM 표준 암호화로 기밀성과 무결성을 함께 보장(난독화 대비 보안 수준 향상)
- 중복 적용 제거 — binlog→SQL upsert 변환과 재시도로 중복 적용 에러를 없애 동기화 정합성 향상
- 가시성 향상 — 고객사 다운로드 모니터링으로 전송 상태를 실시간 확인
- 운영 자산화 — 모니터링 설계·아키텍처·런북·배포 시나리오를 문서로 정리

웹 상세: https://portfolio-equ.pages.dev/items/binlog-shipping-encryption.html

09 · CI/CD · 2026

CI/CD 개발 완료 루틴 Bootstrap Kit

여러 Bitbucket repo에 Jira 브랜치, Bitbucket Pipelines, Claude 기반 AI PR 리뷰, Slack 알림, Docker 이미지 push 루틴을 같은 방식으로 적용하기 위한 중앙 bootstrap kit 작업

기간	소속	역할
2026	래브라도랩스(LabradorLabs)	설계 · 구현 · 문서화

`Bitbucket Pipelines` `Jira` `Claude PR Review` `Slack` `Docker` `Full-SHA Pinning` `Secret Scan`

작업자의 메모

이 작업은 “AI 리뷰 기능을 붙였다”가 아니라 “Jira에서 시작해 Docker image push와 배포 완료 신호까지 이어지는 개발 완료 루틴을 kit로 만든 것”에 가깝습니다. 실제 기준 소스는 repo standards 문서가 아니라 bootstrap kit입니다.

배경

저장소가 늘어날수록 개발 완료 흐름이 repo별로 파편화됐습니다. Jira 브랜치 생성, 브랜치 푸시 테스트, PR 자동 생성, Claude 기반 AI 리뷰, Slack 알림, Docker image tag/push 방식이 저장소마다 달라지면 운영자가 결과를 같은 기준으로 해석하기 어렵습니다.

역할

공통 개발 완료 루틴을 설계하고, Jira 브랜치·Bitbucket Pipelines·Claude PR Review·Slack 알림·Docker build/push가 이어지는 중앙 bootstrap kit 구조를 구현했습니다. README와 beginner guide, central-runner, test-strategy 문서를 정리해 새 repo도 profile 선택 후 같은 방식으로 bootstrap하고 검증할 수 있게 만들었습니다.

핵심 작업

- 중앙 실행형 구조: target repo에는 `scripts/devflow.sh`, pre-push hook, env 파일, `bitbucket-pipelines.yml`, PR template 같은 얇은 파일만 배치
- 언어별 profile: `java-spring`, `go`, `python`, `js`, `mixed` profile을 제공해 repo 유형별 테스트 진입점을 표준화
- mixed profile: 변경 경로를 component와 매칭해 필요한 Dockerfile만 테스트하고, 매칭되지 않는 변경은 기본 실패로 처리
- 브랜치 푸시 중심 흐름: branch check 통과 후 PR 생성과 AI 리뷰까지 이어지도록 구성

임팩트

- 표준화 — repo별로 복제되던 CI/CD 습관을 중앙 bootstrap kit와 profile 기준으로 통합
- 품질 게이트 — branch check, PR 자동 생성, AI PR review, main check, Docker push를 하나의 개발 완료 흐름으로 연결
- 보안성 — secret scan-before-AI, full-SHA pinning, kit repo allowlist로 실행 경계를 명확히 함
- 운영성 — Slack 실패 요약, PR URL, pipeline artifact를 남겨 리뷰·디버깅·배포 추적 시간을 줄임

웹 상세: https://portfolio-equ.pages.dev/items/ai-cicd-review-kit.html