JaeJun Lee
https://www.jx2lee.kr
Github: https://github.com/jx2lee
Email : dev.jaejun.lee.1991@gmail.com

ABOUT ME

가상자산 거래소와 클라우드 서비스 기업에서 데이터 플랫폼 구축부터 운영, 실시간/배치 파이프라인 설계까지 폭넓은 경험을 가진 데이터 엔지니어입니다. 수학 전공자로서 컴공 지식을 스스로 채우며, 본질에 대한 깊은 이해와 새로운 기술에 대한 끊임없는 호기심을 바탕으로 복잡한 문제를 해결해요. AWS, GCP, Kubernetes, Kafka, Airflow, BigQuery 등 데이터 인프라를 다룰 수 있고, 오픈소스 기여와 문서화를 통해 기술 생태계에 공헌하고 있어요. 데이터 무결성, 안정성, 그리고 "누구나 접근 가능한 신뢰 가능한 데이터"를 제공하는 것에 자신있어요.

개발자 경험 주도 데이터플랫폼 모노레포 도입 **2025.12 - 2026.02**

Description: 데이터플랫폼 내 여러 파이프라인과 라이브러리 코드를 단일 저장소(Monorepo)에서 효율적으로 관리하기 위한 환경을 구축했다. Python의 차세대 패키지 관리 도구인 'uv'를 활용하여 워크스페이스를 구성하고, 공통 라이브러리를 'symlink'로 연결하여 중복을 제거했으며 모든 CI/CD 파이프라인은 GitLab을 통해 통합 관리하여 개발 생산성과 코드 일관성을 높였다.

- **Problem**
  - 여러 개의 분산된 레포지토리로 인한 파편화된 의존성 관리의 어려움
  - 공통 라이브러리 변경 시, 이를 사용하는 모든 프로젝트에 일일이 버전을 업데이트하고 배포해야 하는 번거로움
  - 프로젝트 간 코드 일관성 유지가 어렵고, 신규 프로젝트 셋업 비용 증가
  - 전체 플랫폼의 변경 사항을 한눈에 파악하기 어렵고, 통합 테스트의 부재
- **Action**
  - Python 가상 환경 및 패키지 관리를 위해 'uv' 도입, 빠른 속도와 'pip' 호환성으로 개발 환경 통일
  - Monorepo 내에 'libs', 'pipelines', 'services' 등 디렉토리 구조 표준화
  - 공유 라이브러리는 'uv'의 워크스페이스 기능을 활용하고 'symlink'로 각 프로젝트에 연결하여 중복 없이 참조하도록 구성
  - GitLab CI/CD 파이프라인 설계: 'changes' 키워드를 활용하여 변경된 디렉토리(프로젝트)만 선택적으로 빌드/테스트/배포
  - 레포지토리 전체의 코드 품질을 관리하기 위해 'pre-commit' 훅과 통합 린트(lint) 및 포매팅(formatting) 적용
- **Result**
  - 공통 라이브러리 업데이트 및 전파 시간 80% 단축
  - 'uv' 도입으로 로컬 개발 환경 구성 및 CI/CD 파이프라인의 패키지 설치 속도 50% 이상 개선
  - 신규 파이프라인 프로젝트 생성 시 보일러플레이트 자동화로 초기 설정 시간 단축
  - Monorepo 구조 하에서 코드 변경의 영향 범위를 명확히 파악 가능해져 배포 안정성 증대
- **Impact**
  - 플랫폼 전체의 기술적 통일성을 확보하고, "하나의 팀"처럼 협업하는 문화의 기반을 마련
  - 코드 재사용성이 극대화되고, 개발자들이 핵심 비즈니스 로직에 더 집중할 수 있는 환경 제공
  - 전사적으로 파편화된 Python 프로젝트들의 관리 표준을 제시하는 선례를 남김
  - 신규 입사자가 플랫폼의 전체 코드 베이스를 빠르게 파악하고 적응할 수 있도록 기여
- **Self-Reflection**
  - Monorepo는 단순히 코드를 합치는 것이 아니라, 도구(uv, CI/CD), 규칙, 문화가 함께 가야 성공할 수 있음을 체감
  - 'uv'와 같은 최신 도구를 검토하고 도입하며 기술 트렌드를 실무에 적용하는 경험을 쌓음

- GitLab CI/CD의 고급 기능(e.g., dynamic child pipelines)을 활용하여 복잡한 Monorepo의 빌드 시스템을 효율적으로 설계하는 역량을 키움
- 대규모 코드 베이스를 관리하면서 발생하는 장단점을 깊이 이해하고, 트레이드오프를 고려한 아키텍처 설계의 중요성을 배움
- 모노레포 영감은 Airflow의 파이썬 패키지 의존성 관리 어려움에서 비롯되었으며, '패키지 지옥'에서 벗어나고자 하는 의지가 컸음
- 플랫폼이라는 하나의 놀이터를 제공하는 관점에서, 레포지토리가 분리되어 있는 구조는 개발자의 사용자 경험을 저해한다고 판단했음

Data Quality Gate 개발 with Runtime/Semantic **2025.06 - 2025.08**

Description: (IN BITHUMB) 데이터 파이프라인 전반에 걸쳐 데이터 품질을 보장하기 위한 Data Quality Gate를 구축했다. Ingestion 시점의 'Runtime' 검증과 Transformation 시점의 'Semantic' 검증으로 나누어 데이터 신뢰도를 높이는 것을 목표로 했으며, Great Expectations 0.8.x 버전을 활용하여 설정 파일 기반의 자동화된 데이터 품질 파이프라인을 구현했다.

- **Problem**
  - 데이터 파이프라인에 유입되는 Null, 잘못된 형식의 데이터로 인한 후반 프로세스 장애 발생
  - 데이터 품질 이슈 발생 시 원인 파악 및 수정에 많은 시간 소요
  - BI 대시보드 및 분석 결과의 신뢰도 하락으로 데이터 기반 의사결정의 걸림돌
  - 수동으로 데이터 품질을 검증하는 데 드는 비효율적인 리소스
- **Action**
  - Great Expectations(0.8.x) 도입하여 데이터 품질 검증 자동화 환경 구축
  - 'Runtime' 검증: Ingestion 단계에서 데이터 형식, Null 여부 등 기본적인 정합성 체크
  - 'Semantic' 검증: Transformation 단계에서 비즈니스 로직 기반의 통계적 분포, 관계 무결성 등 의미적 품질 체크
  - 설정 파일(YAML) 기반으로 검증 규칙(Expectation)을 정의하고, CLI를 통해 파이프라인에 통합
  - 품질 검증 실패 시 Airflow Task 실패 처리 및 슬랙 알림 연동으로 즉각적인 대응 체계 마련
- **Result**
  - 데이터 품질 이슈로 인한 파이프라인 실패율 30% 감소
  - 품질 검증 리포트(Data Docs) 자동 생성으로 데이터 프로파일링 및 현황 파악 용이성 증대
  - 데이터 오류 발생 시 원인 추적 시간 단축 (MTTR 개선)
  - 신규 데이터 파이프라인 구축 시 품질 검증 절차 표준화 및 적용 시간 단축
- **Impact**
  - "신뢰할 수 있는 데이터" 제공이라는 데이터팀의 핵심 가치 실현
  - 데이터 분석가 및 현업 사용자의 데이터 신뢰도 향상 및 데이터 활용도 증진
  - 품질 게이트 도입으로 데이터 거버넌스 체계의 기술적 기반 마련
  - 자동화된 품질 검증 문화를 팀에 정착시키는 계기 마련
- **Self-Reflection**
  - 데이터 품질은 파이프라인의 특정 지점이 아닌, End-to-End 관점에서 관리되어야 함을 체감
  - Great Expectations 같은 도구를 활용하여 "기대를 코드로 관리"하는(Expectations as Code) 경험 확보
  - 'Runtime'과 'Semantic'으로 품질 검증의 책임을 분리함으로써 복잡한 파이프라인을 더 체계적으로 관리할 수 있음을 배움

- 단순히 도구를 도입하는 것을 넘어, 조직의 데이터 문화에 맞는 검증 프로세스를 정립하는 것이 중요함을 깨달음

데이터 플랫폼 아키텍처의 재설계_Software Architecture Document 를 중심으로 2025.09 - 2025.11

Description: (IN BITHUMB) 데이터 플랫폼 전반의 복잡도가 증가하면서 시스템 구조 · 운영 방식 · 의사결정 이유를 명확히 설명하기 어려운 상황이었다. 이를 해결하기 위해 CMU SEI의 Software Architecture Document(SAD) 템플릿을 기반으로 데이터 플랫폼 전체의 아키텍처를 재설계하고, Architecture Decision Record(ADR)를 도입하여 기술 선택과 운영 기준을 문서화했다. 모든 문서는 마크다운 형태로 작성되었으며 mkdocs & Gitlab CI/CD 와 S3 static web hosting 으로 제공한다.

- **Problem**
  - 방대한 구성요소에 비해 산발적으로 흩어져 있는 문서로 신규 온보딩 지연
  - "왜 Redshift 인가? 왜 운영정책은 이러지?" 의 의사결정이 쌓이지 않아 반복적인 논의 발생
  - 운영 관리문서 부재, 조직 간 용어 불일치
- **Action**
  - SEI SAD 템플릿 기반 아키텍처 재설계(CnC/Deployment 전통 + Pipeline/Management 등의 데이터 특성을 반영한 커스텀 뷰)
  - With ADR(Architecture Decision Record): 변경 이력 관리 및 리뷰 문화 활성화를 도입, "왜?" 의 창고 구성
  - 문서 빌드 파이프라인 개발: mkdocs 로 정적 문서 빌드(Merge 혹은 MR 이벤트CI/CD), 코드 수정에 따른 문서 미변경 시 알람 및 수정사항 반영 제안
  - 운영 런북(온보딩가이드): 플랫폼 운영(트러블슈팅) 문서 작성 및 문화 공유, 문서는 마크다운 !
- **Result**
  - 신규 엔지니어 온보딩 기간 단축 및 문서 창고(WAREHOURSE) 로 MTTR 개선
  - 플랫폼 문서 공유로 불필요한 커뮤니케이션 감소
  - 구성요소/파이프라인/운영이 어루어져 설명가능한 플랫폼 제공
- **Impact**
  - 코드에 매몰되어 있던 나를 설계 철학/운영/품질속성 기반 아키텍처로 넓은 시야를 갖게된 계기
  - 팀 간 책임 경계를 명확히 하여 운영 혼선/중복제거(DO NOT REPEAT YOURSELF) 구현
  - "감"이 아닌 "근거 기반(ADR)"으로 진행
- **Self-Reflection**
  - 기술 문서화는 단순 기록이 아니라, "조직의 사고 모델을 통합하는 작업"
  - 코드 · 운영 · 의사결정의 관계를 구조적으로 관리해야함. IF DO NOT, 기하급수적으로 증가하는 혼란
  - 플랫폼의 문서는 "일관성" 을 유지하기 위한 수단
  - 플랫폼 운영에 더 깊이 있는 곳(Quality Gate, Schema Evolution ETC)에 밀착시키는 방향으로 확장 기대

DATA API 2025.07 - 2025.10

Description: (IN BITHUMB) 거래소 서비스에서 사용자별 최근 3년 거래금액 등 고비용 집계 데이터를 실시간에 가깝게 활용하고자 DW의 컴퓨팅 파워를 활용해 집계 데이터를 생성, Reverse ETL을 거쳐 RDB에 저장한 후 FastAPI 기반 API 서버로 제공했다. Airflow로 배치를 운영하고, Redshift에서 집계 후 MySQL로 적재하여 서비스팀에서 집계 -¿ 조회 가능한 형태의 API 레이어를 구축했다.

- **Problem**

- 고비용 집계 쿼리를 OLTP에서 수행할 경우 서비스 지연 또는 Incident 높은 발생 위험
- DW(Redshift)에 직접 API 제공하기 버거운 보안/동시성/사용자 경험
- 거래소 간접 서비스에 제공하기 부족한 레이어

- **Action**
  - Airflow 기반 집계 파이프라인 개발: 서비스별 SLA 를 만족하도록 파이프라인 개발, 모니터링 환경구성
  - Reverse ETL 구조 설계: 간단한 조회만 OLTP 로 제공한다 의 방향, 모든 집계는 OLAP
  - API 개발: 비동기 워커(Unvicorn) 와 ASG 로 인한 프로세스 매니저(Gunicorn) 활용, 외부 캐시대신 로컬캐시
  - 모노레포 구성: 다양한 서비스 API 코드들이 하나의 Repository 에 관리, 의존하지 않도록 구현(CD 파이프라인 개별)
  - 이벤트 찍어내기 용도로 사용가능한 인하우스 툴 개발(Django/Celery)
- **Result**
  - DW 집계 SLA 99.9%
  - 타 서비스 API 대비 40% 낮은 스펙으로도 서비스 제공 가능
- **Impact**
  - 기존에는 불가능했던 "DW 기반 데이터의 실시간 API 제공"이라는 새로운 데이터 제품을 만들었음.
  - 단순 ETL 생산 조직에서 벗어나, 데이터 제품(Data Product)을 설계 · 제공하는 서비스형 데이터 플랫폼 역량 확보
  - 무거운 연산은 DW 로 위임, 거래소서비스 트래픽에 영향을 주지 않도록 분리함으로써 운영 안정성 향상됨.
- **Self-Reflection**
  - "DW → 서빙DB → API"로 이어지는 end-to-end 제품 구조를 직접 설계하면서 데이터 파이프라인과 애플리케이션 레이어 연결에 대한 실전 경험을 쌓음
  - 서비스팀과 협업하여 SLA/SLO를 정의하는 과정에서 "데이터 엔지니어의 결과물이 어떻게 실제 사용자 경험으로 이어지는가"를 명확히 이해함
  - 운영 자동화와 장애 대응 체계(Runbook)의 중요성을 다시 한 번 배웠고 이후 다른 프로젝트에도 동일한 운영 철학을 적용하는 계기가 됨

CDC Pipeline with Debezium **2024.05 - 2024.08**

Description: AWS RDS의 변경사항을 실시간으로 확보하기 위해 Debezium 기반 CDC(Change Data Capture) 파이프라인을 구축했다. Debezium Source Connector가 RDS binlog를 읽어 MSK로 발행하고, Kafka Connect Sink Connector 가 BigQuery로 실시간 적재한다. CDC 특성상 스냅샷 부하, binlog retention, 커넥터 재시작 시 오프셋 복구 등 운영 난도가 높은 영역을 메트릭 · 알림을 기반으로 운영할 수 있도록 정비했다.

- **Problem**
  - 분석 · 서비스 팀에서 "변경 이력" 기반 데이터가 필요했지만, 기존 DW 파이프라인으로는 삭제/갱신 이벤트를 포함한 history 를 얻기 어려움
  - log_ 테이블은 비즈니스 로직과 스키마가 달라 실제 변경 이벤트와 불일치하는 경우가 많았음
  - CDC는 지속 운영이 핵심인데, 메트릭 · 지연 · 에러를 관측할 체계 부재
- **Action**
  - 안정적 Snapshot/Lock 운영 정책 수립 (DBA 협업): writer 에 직접 붙는 형태로 운영되었기에 실 서비스에 영향이 없도록 옵션 조정(snapshot.mode/locking.mode)
  - Kafka Connect EKS 배포 자동화: Helm · Argo 기반 GitOps 환경 구성, Connector 설정을 CaaC(Configuration As s Code) 형태로 관리

- Observability: AMP(Prometheus) -¿AMG(Grafana) 로 JMX 설정값을 통해 변경 데이터가 흐르는 지표(이벤트발생¡-¿적재까지의 duration) 를 관측할 수 있는 환경 구성
- Connector 복구 방안 체계: Promotion 등으로 binlog 포지션이 변경된 경우를 대비한 운영 스크립트 및 문서작성

- **Result**
  - 한 달 기준 99.9% 적재 성공률 유지(인프라 장애 제외)
  - Snapshot/Lock 관련 DB 성능 이슈 0건
  - CDC 이벤트 기반 모델을 활용한 서비스 · 분석 팀의 신규 활용 케이스 증가
  - 운영 지표 시각화로 장애 감지 속도 단축
- **Impact**
  - 기존 log 기반 파이프라인에서는 얻을 수 없던 "행 단위 변경 기록"을 안정적으로 자체 수집할 수 있게 됨
  - Kafka Connect → BigQuery 흐름이 정착되며, 이후 여러 소스 시스템이 같은 패턴으로 확장 가능한 CDC 플랫폼 표준을 만듦
  - 대규모 스트리밍 시스템의 안정 운영 역량 확보
- **Self-Reflection**
  - CDC는 단순히 "데이터를 옮기는 기술"이 아니라 운영 난이도가 높은 분산 시스템임을 깊이 체감
  - 특히 retentionoffsetsnapshotlock 등 실전에서 자주 발생하는 문제를 직접 겪고 해결하며 운영 역량 확보
  - Kafka Connect/Debezium은 문서만 읽어서는 이해하기 어려운 도구였는데, 장애/성능/설정 문제를 직접 해결하며 "코드 레벨 이해를 갖춘 DE"로 성장하는 계기
  - 이후 다른 플랫폼 도구를 도입 · 살펴볼 때도 동일한 원칙(관측성복구성운영성)을 깊이 있게 살핌

Migration to Kafka Connect **2023.10 - 2023.12**

Description: 기존 Kinesis + Lambda 기반 실시간 적재 파이프라인을 MSK(Kafka) + Kafka Connect(BigQuery Sink) 기반으로 전환한 프로젝트이다. 거래소 회원 서비스의 EDA(Event-Driven Architecture) 전환과 함께 서버 로그 및 다양한 이벤트를 안정적으로 수집 · 적재하기 위한 표준화된 실시간 파이프라인을 구축하는 것이 목표였다. Connect 클러스터는 Helm Chart로 구성하고 ArgoCD 기반 GitOps로 배포하여 확장성과 운영성을 모두 확보했다.

- **Problem**
  - 기존 Kinesis 기반 파이프라인은 EDA 전환으로 곧 deprecate 예정이었고, 실시간 이벤트를 Kafka로 일원화 요건
  - Lambda 기반 BigQuery 적재는 스케일링과 오류 재처리에서 운영 복잡도가 높았고, Incident 상황에서 메시지 누락 여부를 확인하기 어려움
  - 서버 로그의 신뢰성 있는 적재율(SLO) 보장 체계가 없었으며, 인프라 장애 시 재처리 기준도 명확하지 않았음
  - MSK → BigQuery 파이프라인을 구축하는 방식이 여러 가지 있었지만 운영 복잡도 · 성숙도 · 확장성 측면에서 명확한 기준이 없었음
- **Action**
  - Kafka Connect 로 전환: SinkConnector(BigQuery) 채택(메시지 재처리, schema auto-create 등 지원, Source/Sink 구성 분리의 유연성)
  - EKS 기반 Connect Cluster 운영: Confluent official 저장소를 커스터마이징, Gitops
  - Observability: end-to-end 처리 지표, Grafana 로 Connect cluster 모니터링(w/ Alert)
- **Result**
  - 한 달 운영 기준 99.9% 이상의 BigQuery 적재 성공률 달성 (인프라 문제 제외)

- Connect 기반 파이프라인으로 전환 후 신규 이벤트 파이프라인 구축 속도 단축
- EDA 전환 서비스들과의 통합이 쉬워져 조직 전체의 데이터 흐름 단일화됨

- **Impact**
  - Kafka Connect를 통해 실시간 수집 파이프라인을 표준화하고 다양한 소스/타깃 시스템으로 확장 가능한 데이터 플랫폼 인터페이스 제공
  - Kinesis → Kafka로의 전환 작업은 전체 데이터 파이프라인의 유지보수 비용 감소
  - 로그 · 이벤트 수집의 신뢰성을 확보함으로써 이후 CDC, 실시간 분석, 피드백 모델링 등 조직의 데이터 활용 기반 강화

- **Self-Reflection**
  - Kafka Connect는 "구성하면 끝"이 아니라 메트릭 · 재처리 · 버전 호환성 · 스키마 일관성 등 운영 난도가 높은 도구임을 체감
  - 특히 DLQ 전략 부재, Sink 재시도 정책, Connector 재배포 타이밍 같은 실전 운영 문제를 몸으로 겪으며 스트리밍 파이프라인 운영의 핵심 포인트 deepdive
  - EKS/Helm/ArgoCD 기반 배포 자동화를 경험하면서 플랫폼 엔지니어로서의 역량(SRE/DevOps 기반 사고)이 크게 확장

dbt CI/CD Pipeline Migration **2023.07 - 2023.10**

Description: 기존 Bamboo + Airflow 기반의 dbt 배포 파이프라인은 낮은 사용성, 빌드 결과까지 도달하는 긴 흐름(stash → bamboo → airflow) 등으로 개발자 경험(DX)이 좋지 않았다. GitHub Enterprise 도입을 계기로 CI/CD 체계를 GitHub Actions 기반으로 재설계했으며, Slim CI / Full Build / Reviewdog / Jira 연동 등 데이터팀 중심의 자동화를 구축하여 운영성과 일관성을 확보했다.

- **Problem**
  - Bamboo 기반 파이프라인은 사용자 경험이 낮고, 빌드/테스트 결과를 확인하기까지 많은 홉 존재
  - Airflow DAG 기반 배포 방식은 변경 이력이 분산되어, 실패 원인 파악이 어려움
  - 개발자들이 변경된 모델에 대한 영향도(상속 관계-upstream/downstream) 파악 없이 전체 빌드를 반복, 비효율 발생
  - GitHub Enterprise 도입 이후 기존 CI 체계를 재설계할 필요가 있음

- **Action**
  - GitHub Actions 기반 CI/CD 전환: dev/prod 브랜치별 워크플로우 분리
  - Slim CI(`dbt build --select modified`) 및 Full Build 전략 수립
  - Reviewdog, SQLFluff 등 lint 환경 도입하여 코드 품질 자동 검증
  - Jira 연동 액션 활용 → PR 생성/리뷰/머지 시 자동 태깅 및 이슈 트래킹 자동화
  - EKS 기반 self-hosted runner 설계, 빌드 환경 표준화 및 실행 일관성 확보
  - S3 캐싱 전략 도입하여 빌드 속도 최적화

- **Result**
  - 기존 Bamboo 대비 빌드 속도 40% 개선
  - Slim CI 도입으로 변경 모델 중심 빌드가 정착, 전체 빌드 비용 20% 절감
  - 개발자 PR 리뷰 사이클 단축, dbt 모델 품질 및 배포 일관성 향상
  - CI/CD 파이프라인 장애율 감소, self-hosted runner 기반으로 안정적 실행 환경 확보

- **Impact**

- dbt 빌드/테스트/배포 전체 파이프라인을 자동화해 개발자 경험(DX)을 크게 개선
- GitHub Actions 도입으로 통합된 코드 리뷰/검증/배포 흐름 확보 → 데이터팀 생산성 향상
- 팀 단위의 공통 워크플로우(Standardized Workflow Repository) 구축으로 새로운 프로젝트 확장 용이
- 린트 및 품질 기준 자동 검증으로, 스키마 · 쿼리 품질 향상

- **Self-Reflection**
  - CI/CD 도구는 "빌드를 위한 시스템" 보다 과정 전체의 사용성 · 속도 · 가시성을 결정한다는 점을 실감
  - dbt 의존성 그래프 기반의 빌드 전략(Slim CI)은 데이터 제품 개발자의 생산성을 극적으로 높인다는 사실을 배움
  - GitHub Actions + EKS Runner + Lint + Jira 연동 경험은 이후 다른 파이프라인에도 확장 가능(e.g storypoint auto-fill w/ airflow)
  - CI/CD 파이프라인이 갖춰야 할 표준(재현성, 가시성, 자동화)을 깊이 이해하는 계기가 되었음

Self-Serve Data 환경 제공 **2023.07 - 2025.10**

Description: 사내 크루가 데이터 조직 도움 없이 데이터를 찾고, 적재하고, 변환하고, 탐색할 수 있는 Self-Serve Data Platform 을 구축한 프로젝트이다. DataHub(데이터 카탈로그), Airbyte(외부 데이터 파이프라인), MageAI(로우코드 데이터 워크플로우)를 Helm Chart + ArgoCD 기반으로 배포하여, 데이터셀 의존도를 낮추고 내부 사용자 중심의 데이터 접근성을 크게 향상시켰다.

- **Problem**
  - 내부 여러 조직이 필요한 데이터를 얻기 위해 매번 데이터셀에 요청하며 병목 발생
  - RDB / DW / 외부 API 등 데이터 위치를 파악하기 어렵고 메타데이터 관리가 없어 재사용성이 낮았음
  - 외부 데이터(거래소 API, CMC 등) 직접 구현 시 신뢰성 및 유지보수성이 떨어짐
  - 비전문가도 사용할 수 있는 직관적이고 self-serve 친화적인 환경 부재
- **Action**
  - DataHub 구축:
    - Airflow 스크립트로 RDB information schema ingestion 자동화
    - dbt/BigQuery는 recipe yaml + ingestion CLI 기반 메타데이터 수집
    - Helm + ArgoCD 배포, RBAC 및 Tag 정책 구성
  - Airbyte 도입: Python CDK 기반 custom connector 개발
    - 커넥터 저장소 구성 및 semantic versioning 도입
    - GitHub Actions → ECR 자동 빌드/배포
    - OAuth 기능 부재 해결을 위해 oauth2-proxy 연동
  - MageAI 도입: Third-Generation Workflow Management
    - 외부 기관 보고용 정제 작업 및 워크플로우 자동화
    - 공식 이미지 PostgreSQL client 누락 → 직접 PR 제출 및 merge 경험
    - Helm chart 수정하여 shared-volume 기반 운영 구조 구성
  - Self-Serve 구조 확립: 데이터 적재(EL), 탐색(Browse), 메타데이터(Document), 변환(Transform)을 단일 연결
- **Result**
  - 보안팀이 DataHub Tag 기능으로 개인정보 레이블링 직접 수행 → 데이터셀 요청량 감소
  - 외부 거래소 캔들 데이터 수집 자동화로 신뢰성 향상 및 데이터 일관성 확보
  - MageAI 기반 보고 자동화로 수동작업 0건
  - 내부 임직원이 스스로 데이터 위치를 파악하고 활용하는 self-serve 문화 정착

- **Impact**
  - 데이터 조직의 역할이 "요청 처리" 중심에서 "플랫폼 제공" 중심으로 전환되는 기반을 마련
  - DataHub, Airbyte, MageAI 등 OSS 운영 경험을 통해 플랫폼 엔지니어링 역량 강화
  - 팀 간 데이터 활용 생태계 확장: 개별 조직이 스스로 분석 가능한 구조 제공
  - Helm/ArgoCD 기반 배포 표준화로 이후 신규 데이터 제품 도입 시 운영 일관성 확보
- **Self-Reflection**
  - Self-Serve 플랫폼 구축은 "도구 제공"이 아니라 사용자 workflow 를 쉽게 만들수 있는 일임을 이해함
  - 성숙한 OSS부터 초기 단계 도구까지 운영하며 내부 구조 & 제약을 깊게 분석할 수 있었음
  - 인증 문제(OAuth), Helm chart 수정, 커넥터 개발 등 실전 운영 이슈를 해결하며 운영 역량 강화

Appendix: Diagrams

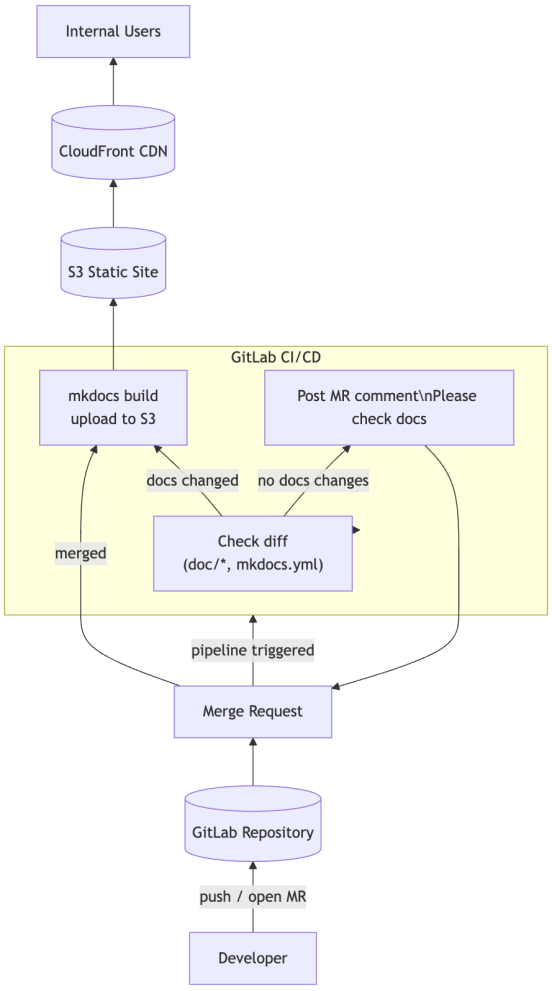

Figure 1: SAD CI/CD Pipeline Overview

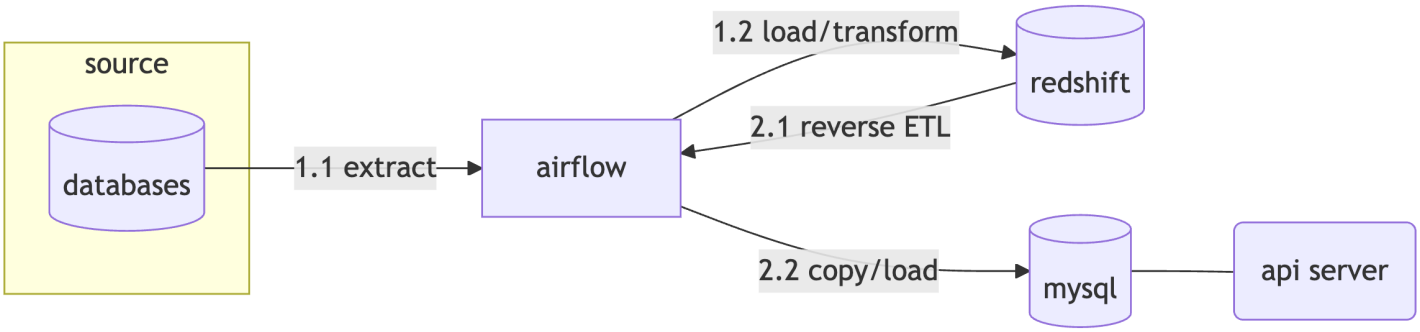

Figure 2: Data API Diagram

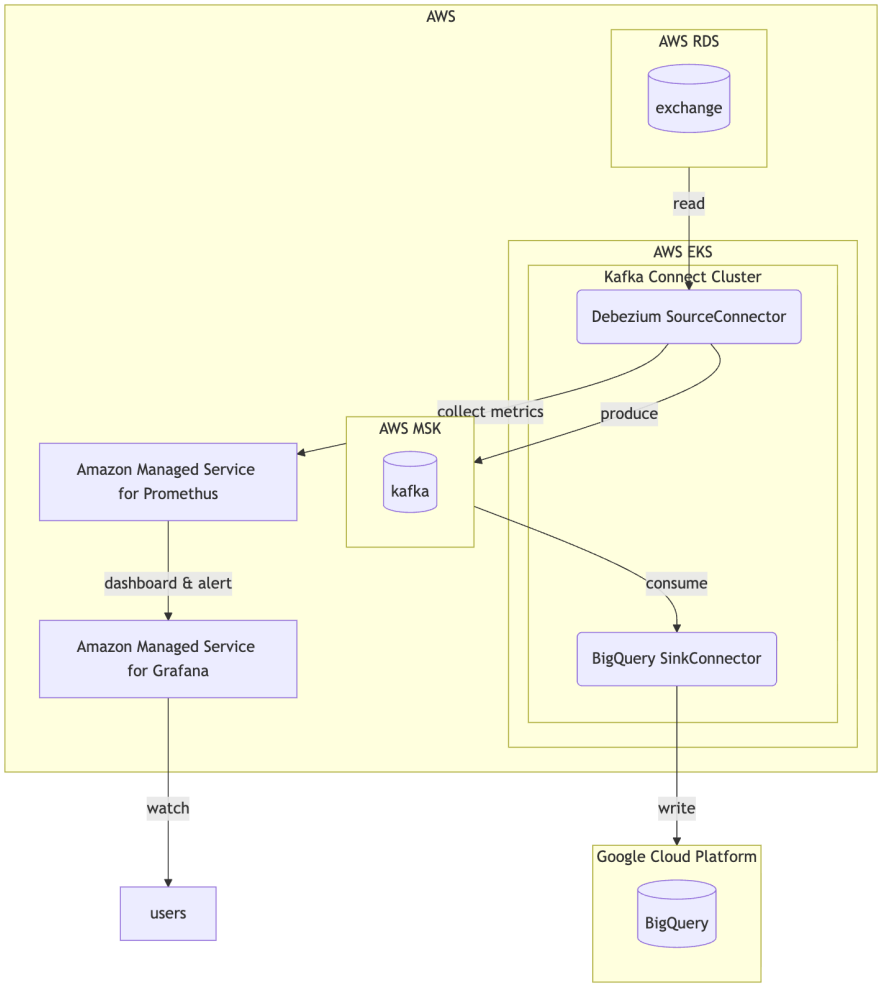

Figure 3: CDC Pipeline Overview

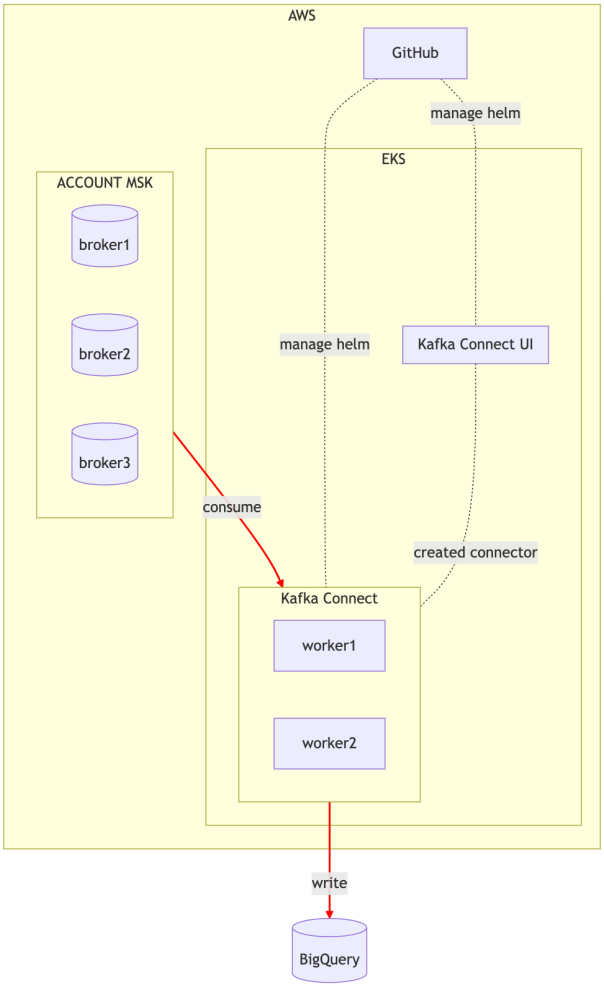

Figure 4: Kafka Connect Overview

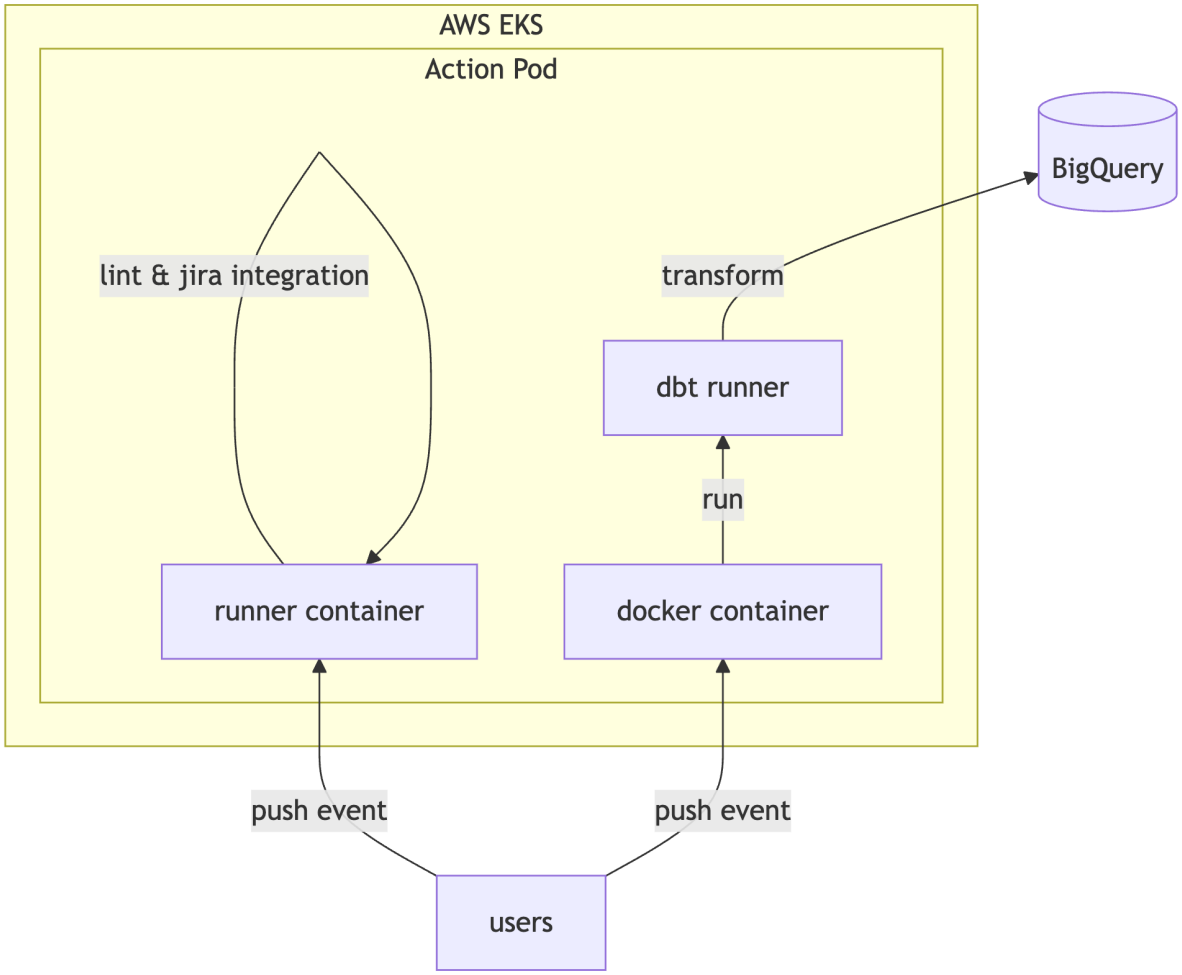

Figure 5: dbt CICD pipeline Overview