경 력 기 술 서

이동욱

CI/CD · 인프라 엔지니어

Email whddlsk994@gmail.com
Phone 010-6330-2599

경력 요약 SUMMARY

회사	썬더소프트코리아
부서 / 직급	System Solution Lab(Multimedia) / 연구원
재직 기간	2023-09 ~ 재직 중
담당 업무	고객사 Camera part의 CI/CD 파이프라인 운영·자동화 및 Linux 서버 구축·장애 대응
자격증	Naver Cloud Platform Professional (2023-08) · AWS Certified Solutions Architect – Associate (2026-07)

핵심 성과 KEY RESULTS

파이프라인 소요 시간
12h → 3h
▼ 75% 단축 · 무인화

TDD 테스트 시간
102 → 73분
▼ 28% 단축

신규 AP CI 온보딩
무중단
칩셋 라인업 전반 온보딩

분석 자동화
신규 구축
SW 재사용율 파이프라인

보유 기술 SKILLS

CI/CD	Quickbuild, Jenkins
Scripting	Shell Script, Python, Node.js
Test / Debug	ADB, 정적분석(Coverity), 코드 중복분석(SourcererCC)
OS / Infra	Linux
DB	SQLite, MySQL
협업	Jira, Confluence, Git
Cloud	AWS (SAA 자격 보유), Naver Cloud (Professional 자격 보유)

주요 수행 프로젝트 PROJECTS

01	**테스트 베드 OS 이미지 업데이트 & 빌드 산출물 Sync 자동화**	12시간 → 3시간 (75% 단축) · 무인화
02	**HAL CI 테스트 시간 개선**	TDD 102분 → 73분 (28% 단축)
03	**HAL CI 테스트 자동화 프레임워크 2.0 전환 & 릴레이 서버 구축**	Node.js 릴레이 서버 직접 설계·구축
04	**신규 AP CI 온보딩 (HAL CI + Daily CI)**	칩셋 라인업 전반 무중단 온보딩
05	**SW 재사용율 측정 시스템 구축**	분석 파이프라인 신규 구축
06	**CI 운영·개선 업무 — 정적분석 · 환경 셋업 · 대시보드 (지속)**	주간 파이프라인 무중단 운영 외

프로젝트 상세 PROJECT DETAILS

01 테스트 베드 OS 이미지 업데이트 & 빌드 산출물 Sync 자동화

기간	2024-10 ~ 2025-01 (약 2.5개월) · 기여도 100%
성과 (KPI)	파이프라인 소요 시간 **12시간 → 평균 3시간 (75% 단축)** · 개발자 수동 개입 **완전 제거 (무인 자동화)**
역할·전략	단독 구축 · 병목 원인 분석 후 발주 조직에 테스트 구조 개선을 제안·관철
기술 스택	Quickbuild, Shell Script, Android Debug Bridge(ADB)

무엇을, 왜

- 기존에는 개발자가 Camera HAL Layer TDD 패치를 검증하기 전, ① 새 OS 이미지를 테스트 베드 서버에 직접 내려받아 시료에 adb로 OS 업데이트(Fusing)하고, ② 빌드 산출물(`.so`/`.a`)을 직접 patch로 만들어 HAL 브랜치에 merge하는 **수동 작업을 반복**하고 있었습니다.
- 이 반복 작업을 사람의 개입이 필요 없는 **무인 파이프라인**으로 전환하는 것이 목표였습니다.

어떻게 해결했는가

- 매일 정해진 시각에 최신 OS 이미지 존재 여부를 자동 조회 → 테스트 베드 서버로 자동 반입 → 시료 Fusing을 자동 수행하도록 구성했습니다.
- 빌드 산출물을 patch로 자동 생성해 TDD를 통해 검증하고, **Pass 시 develop 브랜치에 자동 merge**되도록 했습니다.
- **안전장치 설계** — TDD Fail 시 해당 이미지를 무효로 판단하고 직전 유효 이미지로 **자동 Rollback**합니다.
- **병렬화 제안** — 순차 처리(A→B→C→D)의 원인이던 '모든 프로젝트 시료에 대해 일괄 테스트 진행 step'이 병목임을 분석하고, 해당 step 제거와 프로젝트별 병렬 수행을 **발주 조직에 건의해 채택**시켜 시간을 단축했습니다.

02 HAL CI 테스트 시간 개선

기간	2025-01 ~ 2025-03 (약 2개월) · 기여도 70%
성과 (KPI)	TDD 평균 테스트 시간 **102분 → 73분 (약 28% 단축)** · 요청 적체로 인한 merge 대기 병목 완화
역할·전략	기존 시스템의 비효율을 평소에 인지하고 개선 아이디어를 직접 제안·구현
기술 스택	Quickbuild, Shell Script, Linux

무엇을, 왜

- 개발자 patch가 develop 브랜치에 merge되려면 HAL CI의 TDD를 통과해 verify +1을 받아야 하는데, TDD가 평균 102분 소요되고 동시 병렬 처리 건수가 제한되어 요청이 몰리면 **대기 병목**이 발생했습니다.
- 시료 수 제약으로 '병렬 수 증설'이 불가능한 상황 → **테스트 자체의 소요 시간 단축**으로 접근했습니다.

어떻게 해결했는가

- **Fail-Stop 도입** — 기존에는 Fail이 나도 전체 TC를 끝까지 수행했습니다. 개발자의 목적은 Fail TC 수정이므로 full TC 수행이 불필요하다고 판단, **Fail 발생 시 즉시 중단**하도록 변경했습니다.
- **Heap leak 검사 최적화** — 기존에는 leak 검사 off 1회 + on 1회, 총 2회 full TC를 반복했습니다. 이를 **1회차부터 leak 검사를 켜서 누수 TC만 list-up**하고 2회차는 해당 TC만 재검사하도록 변경해, 반복 수행 TC 수 자체를 줄였습니다.

03 HAL CI 테스트 자동화 프레임워크 2.0 전환 & 릴레이 서버 구축

기간	2025-10 ~ 2025-12
성과 (KPI)	테스트 자동화 프레임워크 1.0 → 2.0 전환 완료, 기존 파이프라인 **무중단 정상 운영 유지**
역할·전략	API 사용 스크립트 전수 조사 → 리팩토링, 전환에 필요한 릴레이 서버 직접 설계·구축
기술 스택	Quickbuild, Node.js, Express, Redis, Shell Script

무엇을, 왜

- HAL CI의 기반인 테스트 자동화 프레임워크가 1.0 → 2.0으로 버전업되면서 ① 서버들의 프레임워크 업그레이드, ② API 변경에 따른 테스트 스크립트 리팩토링, ③ 변경된 호출·응답 흐름을 중계할 릴레이 서버 구축이 함께 필요해졌습니다.

어떻게 해결했는가

- **스크립트 전수 조사·리팩토링** — 구버전(1.0) API를 사용 중이던 테스트 스크립트를 전수 조사하고, 2.0에서 변경된 API를 사용하도록 리팩토링했습니다.
- **릴레이 서버 직접 구축** — 2.0은 테스트 진행 상태·결과를 callback URL로 전달하는 방식으로 바뀌어 `Quickbuild → 릴레이 → 프레임워크 2.0 → 릴레이 → Quickbuild` 구조가 필요했습니다. 이 릴레이 서버를 **Node.js + Express로 직접 설계·구축**하고, 영구 저장이 필요 없는 진행 중 데이터 특성에 맞춰 **Redis**로 상태를 관리했습니다.

04 신규 AP CI 온보딩 (HAL CI + Daily CI)

기간	2024-05 ~ 2026-02 (지속 수행) · 기여도 100%
성과 (KPI)	신규 AP **칩셋 라인업 전반**을 운영 중인 CI에 **무중단 온보딩**, 전 파이프라인 정상 동작 검증
역할·전략	신규 칩셋 출시 주기에 맞춰 반복 가능한 표준 온보딩 프로세스로 수행
기술 스택	Quickbuild, Jenkins, ADB, Python, Shell Script, SQLite, Jira, Confluence

무엇을, 왜 / 어떻게

- AP(Application Processor) 신규 칩셋이 출시될 때마다 운영 중인 Camera CI에 해당 AP를 추가해 자동 테스트가 돌도록 만들어야 합니다. **카메라 칩셋 라인업 전반의 CI 온보딩을 반복 수행**했습니다.
- CI 시스템에 신규 프로젝트 **Build Enable** → 테스트 파이프라인에 AP Enable 및 정상화 순으로 진행했습니다.
- **Panic 유발 testcase 필터링** — 신규 AP 환경에서 시료에 Panic을 일으키는 TC를 식별·제외해 파이프라인 안정성을 확보했습니다.
- 첫 담당 프로젝트에서 Jira·Confluence 기반 업무·문서 관리 프로세스를 익힌 뒤, 동일 패턴을 표준화해 반복 온보딩했습니다.

05 SW 재사용율 측정 시스템 구축

기간	2025-10 ~ 2025-11 (약 1개월) · 기여도 100%
성과 (KPI)	Android 버전 간 중복 코드 재사용률 **측정 자동화 파이프라인 신규 구축** (수동 분석 대비 반복 가능·정합성 확보)
역할·전략	단독 구축 · 파라미터 기반 재사용 가능한 분석 프레임워크 설계
기술 스택	Quickbuild, Shell Script, Python, SourcererCC

무엇을, 왜 / 어떻게

- Android 새 버전 출시에 따라 이전 버전 대비 중복 코드 사용량(예: `android15/platform` ↔ `android16/platform`) 측정이 필요했습니다.
- **target / origin branch를 파라미터로 입력**받아 SourcererCC 라이브러리로 중복 코드를 분석하는 파이프라인을 구축, 신규 버전마다 동일 절차로 반복 측정할 수 있게 했습니다.

06 CI 운영·개선 업무 — 정적분석 · 환경 셋업 · 대시보드 (지속)

2024-09 ~ 재직 중 (지속 수행) · 주간 정적분석 파이프라인 **무중단 운영** · Quickbuild, Jenkins, Coverity, Shell Script, Linux, React, Node.js

- **주간 정적분석 파이프라인 운영** — Jenkins 스케줄러(매주 목) → Quickbuild 빌드 → Coverity 정적분석 리포트로 이어지는 주간 파이프라인을 무중단 운영. 신규 프로젝트 편입 시 분석 자동 적용.
- **신규 워크스테이션/노드 CI 환경 셋업** — 신규 입고 서버·노드에 동일한 CI 운영 환경 구축. **init 스크립트 리팩토링**(누락 라이브러리 보충·경로 보정)으로 재현 가능한 셋업 표준화 (2025-05 ~ 2026-01, 건별).
- **상시 개선** — 테스트 산출물에 `coverage.info`(코드 커버리지) 포함 개선 · 사내 디버깅 도구 활성화(WebDAV 자동 다운로드 스크립트, Quickbuild Step 연동) · Daily CI 서버 정기 보안 점검 위반사항 조치 · Daily CI Dashboard(React / Node.js) 유지보수.
