Portfolio

김다정 · Cloud / DevOps Engineer.

하이브리드 클라우드와 K8s 기반 인프라를 학습하고 구축합니다

온프레미스와 클라우드를 결합한 하이브리드 인프라 운영에 깊은 관심을 갖고 있습니다.
6개월간의 클라우드 MSP 부트캠프와 팀 프로젝트를 통해 Terraform 기반 IaC, Kubernetes, CI/CD 자동화, 통합 모니터링까지 인프라 전 생애주기를 직접 구축해봤습니다.

오늘의 설계를 의심하고, 내일은 한 발짝 더 앞서갑니다

스스로에게 던지는 엄격한 피드백이 성장의 가장 큰 동력이라고 믿습니다. 이슈를 해결할 때마다 '왜'라는 질문으로 문제의 뿌리까지 파고들고, 거기서 얻은 답을 다음 설계에 반영합니다.

동시에 혼자 끌어안기보다 동료의 지혜를 빌려 정확하게 문제를 푸는 편입니다. 팀에 실질적인 기여를 하는 동료, 그리고 함께 성장하는 엔지니어가 되는 것이 목표입니다.

About

- 이름: 김다정
- 이메일: dajeong248@naver.com
- GitHub: https://github.com/990402DAJEONGKIM

Skills

- **Cloud & IaC:** AWS, GCP, Terraform, Kubernetes
- **DevOps / CI/CD:** GitHub Actions, ArgoCD, Docker, Rundeck
- **Networking:** DMZ, Port Forwarding, HAProxy, etcd, High Availability / Disaster Recovery Architecture
- **Observability:** Prometheus, Grafana, Loki
- **Scripting & OS:** Python, Shell Script, Linux
- **Automation:** RPA (MS Power Automate), SAP, ERP, MES, Microsoft Excel
- **Languages:** English (OPIc IH - conversational), Spanish (basic conversational)
- **Team Collaboration:** Communicated clearly with teams across multiple departments and followed standard procedures
- **Problem Solving:** Investigated and resolved issues independently before asking for help
- **Documentation**: Carefully Created and maintained accurate technical documents

하이브리드 환경 기반 중소형 병원 정보 관리 시스템

2026. 05 ~ present (진행 중, 3인 팀) 기여도 40%

GitHub | Docs

개요

3인 프로젝트로 입원실 · 수술실을 보유한 중소형 병원을 대상으로, 온프레미스와 AWS를 연계한 하이브리드 보안 아키텍처를 설계 · 구축했습니다. ISMS-P · 개인정보보호법 · 의료법 기준에 따라 민감 의료정보는 온프레미스에 저장하고 비민감 데이터는 AWS에 분리 배치하는 데이터 등급 기반 구조를 적용했으며, 보안 관제는 Wazuh SIEM으로 온프레미스와 클라우드 로그를 통합 수집하고 분석하였습니다. 또한 AWS 리전 전체 장애 시에도 서비스 연속성을 확보할 수 있도록 GCP기반 Cloud Standby DR 구조를 함께 구현하였습니다.

역할

- AWS 인프라 설계·구축
- 웹 서비스 개발(환자용 예약 포털 및 관리자 대시보드 프론트엔드·백엔드 개발)
- IaC(Terraform 으로 AWS 리소스 코드화, CI/CD 파이프라인 구성)
- 팀장을 맡아 팀 운영 전반을 지원, 아키텍처 설계도 작성을 통해 설계 방향 수립, 팀 공동작업을 위한 산출물 문서 양식 표준화

Skills

- Cloud & Infra : AWS (VPC, ECS, RDS, ALB, S3, Route 53, WAF, AWS Backup, Auto Scaling, ACM, ECR, KMS)
- IaC & DevOps : Terraform, GitHub Actions (CI/CD)
- Database : RDS (PostgreSQL)

인프라 구축 상세

하이브리드 인프라 아키텍처

GCP 환경은 AWS의 RDS와 실시간 동기화 되며, AWS 장애 시 환자 예약 서비스만 제공하는 경량 DR 구조로 운영됩니다. 세 환경에 설치된 Wazuh Agent 가 모든 로그를 수집 · 분석하며, 탐지된 보안 이벤트는 Slack 알림으로 관리자에게 즉시 전달됩니다.

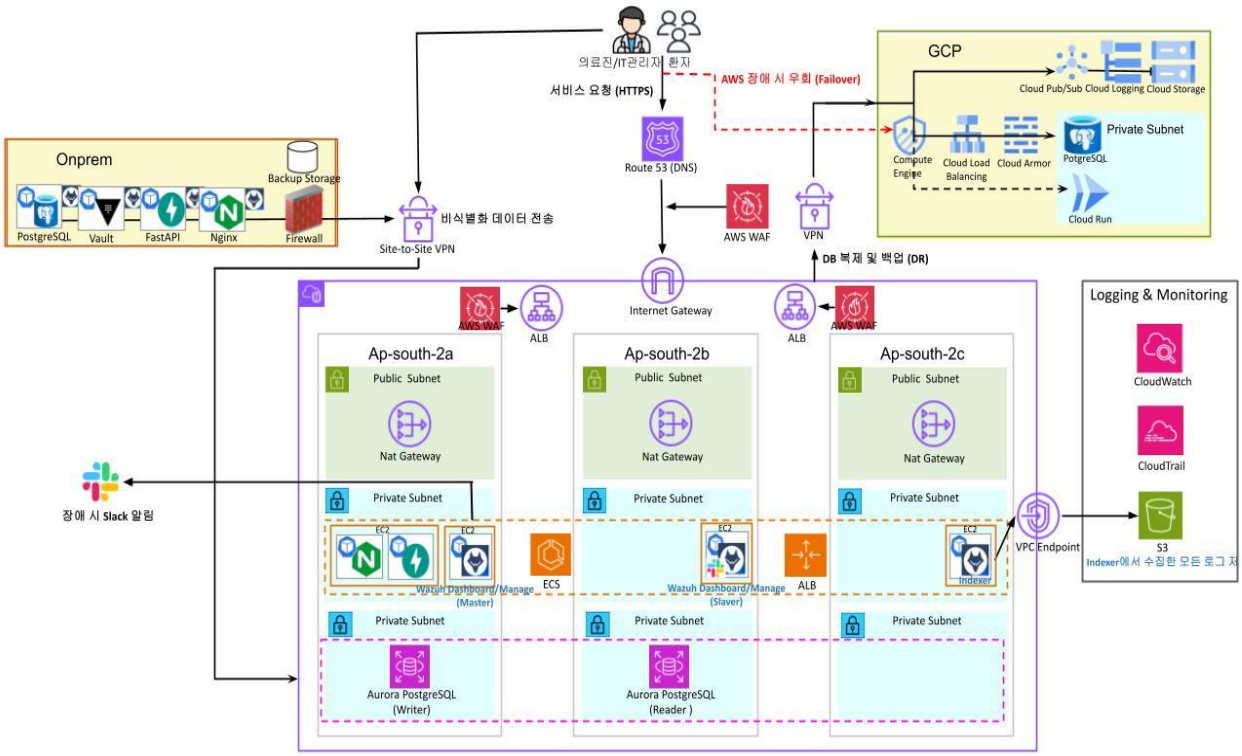

그림 1. 온프레미스(VMware) ↔ Site to Site VPN ↔ AWS VPC 전체 구조도

① 네트워크 설계

- 3개 AZ(ap-south-2a/2b/2c)에 걸쳐 VPC를 구성하여 단일 AZ 장애 시에도 서비스가 중단되지 않도록 설계했습니다.
- 각 AZ는 Public/Private Subnet으로 분리하여 외부에 노출되는 리소스(NAT Gateway)와 내부 리소스(ECS, Aurora)의 네트워크 경계를 명확히 구분했습니다.
- 환자 포털은 Public ALB, 의료진 포털은 Internal ALB로 분리하여 의료진 포털은 허용된 병원 네트워크에서만 접근 가능하도록 구성했습니다.

② 예약 서비스 아키텍처

- DB는 Aurora PostgreSQL Writer/Reader로 분리하여 읽기 부하를 분산하고, GCP PostgreSQL과 실시간 동기화해 DR 전환 시 데이터 손실을 최소화했습니다.
- ECS Task 내에 NGINX와 FastAPI를 Sidecar 패턴으로 구성하여 NGINX→FastAPI 통신이 외부 네트워크를 거치지 않고 localhost로 처리되도록 했습니다. API Key는 NGINX가 환경변수로 주입해 프론트엔드 코드에 노출되지 않습니다.
- Task 수 자동 확장과 EC2 Auto Scaling을 이중으로 구성하여 트래픽 급증 시 Task 먼저 증가하고, EC2 용량이 부족해지면 서버도 자동으로 추가되도록 설계했습니다.

③ 보안/ISMS-P 적용

- 환자 포털에는 SQLi·XSS 등 웹 공격 차단 WAF를, 의료진 포털에는 허용 IP 외 전체 차단하는 WAF를 각각 배치하여 외부 공격과 비인가 접근을 분리 차단했습니다..
- 인증 보안을 강화하기 위해 Access/Refresh Token을 분리 발급하고, Refresh Token은 SHA256 해시값만 DB에 저장하는 Rotation 방식을 적용했습니다.
- 웹에서 모든 API 요청의 요청자·IP·액션·응답 코드를 자동으로 수집했습니다. 진료기록 조회의 경우 미들웨어 기록 외에 RECORD_VIEW 액션을 별도 저장해, 개인정보 유출 민원 발생 시 누가 언제 어떤 환자 기록을 조회했는지 추적할 수 있도록 했습니다.
- DB 비밀번호·JWT 시크릿·API Key는 Secrets Manager에 보관하고 컨테이너 실행 시 환경변수로 주입해 코드 유출 시에도 시크릿이 노출되지 않도록 했습니다.

④ DB 설계

- PostgreSQL 는 벤더 종속 없이 오픈소스 방식으로 온프레미스·AWS·GCP 세 환경에서 동일하게 동작하고, 의료 데이터 특성상 필요한 ACID 트랜잭션과 감사 로그를 오픈소스로 충족할 수 있어 선택했습니다.
- Aurora PostgreSQL을 Writer/Reader로 분리해 쓰기·읽기 부하를 분산했습니다.
- 온프레미스 PostgreSQL에는 민감 의료정보 원본을 보관하고, AWS Aurora에는 비식별화된 예약 데이터만 저장하는 데이터 등급 기반 분리 구조를 적용했습니다.

주요 트러블슈팅

[1] 온프레미스 ↔ RDS 스키마 불일치로 인한 연동 오류

AS-IS

온프레미스 DB 스키마를 사전에 파악하지 않고 AWS RDS sync_* 테이블 관련 테이블을 설계하면서 컬럼명·데이터 타입·VARCHAR 길이 불일치로 pglogical 복제가 동작하지 않았습니다.

TO-BE

온프레미스 스키마를 덤프·분석한 뒤 패치 스크립트로 AWS 테이블을 일괄 수정하고, 이후 스키마 변경 시 온프레미스 구조를 먼저 확인하는 협업 프로세스를 정립했습니다.

[2] RDS 시크릿 로테이션 후 앱 미반영 문제

AS-IS

ECS에서 컨테이너 실행 시 DB 비밀번호를 환경변수로 주입하는 방식은 컨테이너 기동 시 1회만 값을 읽어 고정되기 때문에, 시크릿 로테이션 후 DB 연결 오류가 발생했습니다. 또한 관리자용 DB 계정을 ECS에서 그대로 사용하고 있어 보안상 문제가 있었습니다.

TO-BE

ECS → RDS 전용 DB 계정을 분리 생성하고, Secrets Manager로 관리했습니다. Secrets Manager는 90일마다 자동 로테이션되도록 설정하고, 로테이션 완료 이벤트를 EventBridge로 감지해 ECS 서비스 재배포를 자동 트리거하는 방식으로 개선하여 수동 개입 없이 새 DB접속 정보가 반영되도록 했습니다.

프로젝트 결과

- Terraform 으로 AWS 핵심 리소스 15 종, 총 231 개 리소스를 코드화, 환경 간 인프라 일관성 확보
- ISMS-P 핵심 통제항목 11 개 이행
- 비용 최적화:
  - ECS Fargate 대신 EC2 기반 컨테이너 실행 환경 구성으로 컴퓨팅 비용 절감
  - EC2 와 ECS Task 모두 평소 최소 1 개만 운영하고 부하 시 최대 4 개까지 자동 확장하여 유휴 비용 최소화
  - 오픈소스 PostgreSQL 채택으로 라이선스 비용 절감, Wazuh 서버용 ALB 와 의료진용 ALB 를 통합하여 ALB 비용 절감

향후 계획

- 환자 증상 기반 AI 진료과 추천 기능 확장
- Datadog 도입으로 온프레미스·AWS·GCP 전 환경의 시스템 상태 · 로그 · 서비스별 요청 흐름을 단일 대시보드에서 통합 모니터링하는 체계로 고도화

VMware & K8s 기반 배관 수리 예약 시스템

2026. 03. 13 ~ 2026. 03. 25 (13일, 1인 개인 프로젝트)

GitHub | Docs

개요

3-Tier 아키텍처와 고가용성(HA), CI/CD, 통합 관제까지 포함하는 온프레미스 인프라를 1인으로 구축한 프로젝트입니다. K8s 기반의 Web/WAS 계층과 독립 VM으로 격리된 DB 계층을 결합한 아키텍처를 설계하고 구축했습니다.

역할

- 3-Tier 인프라 설계 및 구축
- HA 구성 및 자동 복구 체계 수립
- CI/CD 파이프라인 및 통합 모니터링 시스템 구축
- 웹 서비스 개발

Skills

- **Cloud & Infra** : Ubuntu VMware, Kubernetes, Ingress, NFS, HAProxy, Keepalived
- **IaC & DevOps** : GitHub Actions (Self-hosted Runner), Helm, Docker Hub
- **Database** : MySQL, ProxySQL, Orchestrator
- **Monitoring** : Prometheus, Grafana, Loki, Alertmanager, Slack API

아키텍처 및 구축 상세

3-Tier 아키텍처

Web/WAS 계층은 Kubernetes로 구성하여 유연한 확장성과 자가 복구 능력을 확보하고, DB는 독립 VM으로 격리하여 보안 및 안정성을 확보했습니다. Bastion 서버를 관문으로 두어 외부망과 내부 전용망을 논리적으로 분리했습니다.

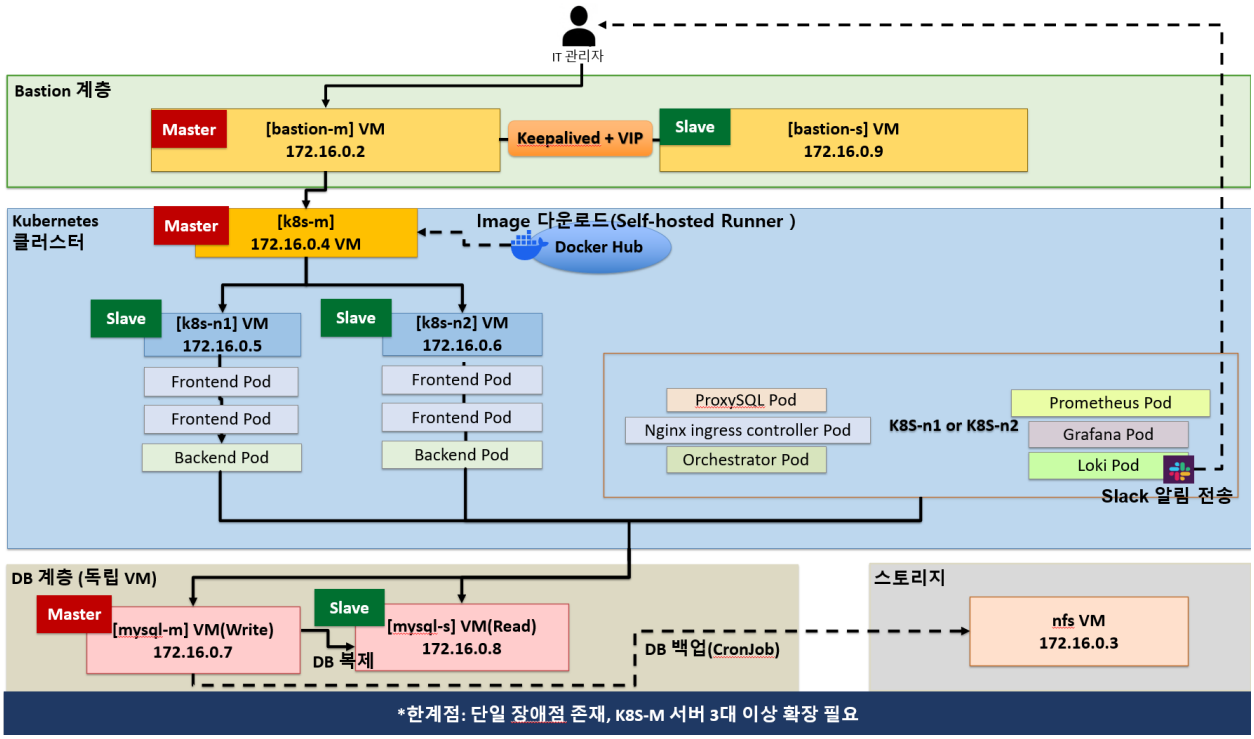

그림 2. Bastion → Ingress → K8s(Web/WAS) → ProxySQL → MySQL HA 구조

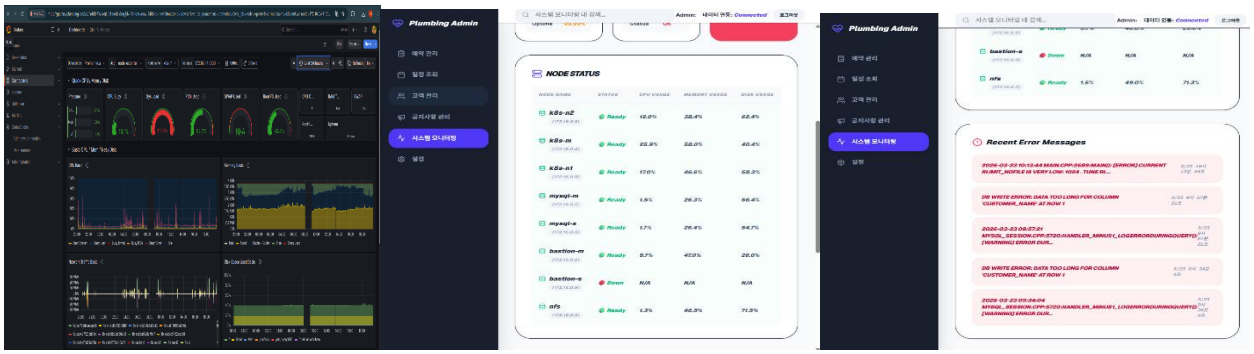

그림 3. Prometheus + Loki + Grafana 통합 대시보드

① 장애 감지 속도 (5분 → 15초)

AS-IS

PromQL에서 CPU·메모리 지표를 5분 평균으로 계산하여 서버 다운 시에도 과거 데이터를 바탕으로 평균값을 계속 반환해 서버가 살아있다고 오판하는 문제가 있었습니다. 결과적으로 장애 인지까지 최대 5분이 소요됐습니다.

TO-BE

15초 단위로 갱신되는 up 메트릭을 병렬로 추가 조회하여, up == 0인 경우 5분 평균값을 무시하고 즉시 Down 상태로 덮어쓰는 로직을 구현했습니다. 장애 인지 시간을 **5분에서 15초**로 단축했습니다

② 외부 종속성 제거 (Self-hosted Runner)

AS-IS

핵심 인프라 컴포넌트(Orchestrator)의 이미지가 공식 Docker Hub에 없는 외부 레지스트리에만 존재하여, 외부 저장소 장애 시 배포가 불가능한 구조였습니다.

TO-BE

GitHub에서 소스코드를 직접 클론하여 이미지를 빌드한 뒤 개인 Docker Hub 레지스트리에 미러링하는 Self-hosted 구조로 개선했습니다. 외부 저장소 장애와 무관하게 배포 독립성을 확보했습니다.

③ 다중 파드 상태 동기화

AS-IS

사이트 점검 모드 상태를 Node.js 앱 메모리에 저장하여, K8s 로드밸런싱 환경에서 파드마다 상태가 달라지는 불일치 문제가 발생했습니다. 관리자가 점검 모드를 켜도 일부 파드는 여전히 예약을 받는 누수가 발생했습니다.

TO-BE

상태 저장소를 앱 메모리에서 MySQL system_settings 테이블로 이관했습니다. 모든 파드가 DB를 공통 상태 저장소로 참조하도록 구성하여, 파드 수에 관계없이 점검 모드 ON/OFF가 즉시 전체 동기화되도록 했습니다.

④ 비전문가 단독 운영 환경을 고려한 통합 관제 대시보드

AS-IS

실제 사이트 운영자가 Grafana · Loki 등 전문 모니터링 도구에 익숙하지 않아 직접 접속하거나 쿼리를 작성하는 것이 현실적으로 어려웠습니다. 또한 장애 발생 시 여러 도구를 오가며 상황을 파악해야 해 비전문가 단독 운영 환경에서 신속한 대응이 어려웠습니다.

TO-BE

Prometheus·Grafana·Loki 데이터를 백엔드에서 PromQL로 가공하여 관리자 웹 페이지 하나에서 서버 상태·에러 로그·DB 현황을 통합 조회할 수 있도록 구현했습니다. Slack 알림에 관리자 대시보드 바로가기 버튼을 추가해 모바일에서도 즉각 대응이 가능하도록 했습니다.

프로젝트 결과

- **서비스 신뢰성:** DB 전 구간 장애 시에도 ProxySQL 캐싱을 통해 무중단 조회 서비스 유지
- **재해 복구:** 인프라 전체를 코드로 관리, CI/CD 파이프라인으로 즉시 동일 상태 재구축 가능
- **내부망 보안:** Self-hosted Runner 내부망 구축으로 안전한 배포 환경 확보
- **통합 관제:** 여러 서버에 분산된 로그를 NFS 에 통합 저장하고, 실시간 분석을 통해 장애를 즉시 인지할 수 있는 커스텀 모니터링 뷰 구현

회고

- MySQL + Orchestrator + ProxySQL 조합의 자동 페일오버 구현에 어려움이 있었음. 이후 개인 프로젝트에서 PostgreSQL + Patroni + etcd 조합으로 10~15 초 이내 **PostgreSQL Standby 서버가 Primary 서버로** 자동 승격 구현에 성공함
- K8s 마스터 노드 단일 구성으로 SPOF 존재. 외부 로드밸런서 기반 마스터 서버를 3 대로 확장하여 개선 예정
- 파드 간 통신 제약 부재로 보안 취약점 존재 NetworkPolicy Ingress·Egress 적용으로 필요한 통신만 허용하도록 개선 예정

AWS Lambda Serverless 날씨 정보 앱

2026. 01. 29 ~ 2026. 01. 30 (2일, 4인 팀) 기여도 30%

GitHub | Docs

개요

서버리스 기반의 데이터 흐름을 학습하고, 지역별 오늘의 날씨 API 데이터를 웹 인터페이스에 시각화하는 4인 팀 프로젝트입니다. AWS Lambda, S3, API Gateway를 활용한 백엔드와 HTML/CSS/JS 기반 프론트엔드를 연동했습니다.

역할

- S3 ↔ API 데이터 바인딩 및 UI 개발
- Fetch API 기반 비동기 통신 로직 구현

Skills

- Cloud & Infra : AWS Lambda, Amazon S3, API Gateway
- IaC & DevOps : GitHub Actions

프로젝트 결과

- S3 → Lambda → Web 흐름을 구현하며 클라우드 리소스가 실제 서비스 화면에 어떻게 연결되는지 학습
- 브랜치 충돌 경험을 통해 main pull → 로컬 충돌 해결 → PR 순서 체득

그림 4. 오늘의 날씨 앱 화면

MS Power Automate 기반 전사 업무 프로세스 자동화(RPA)

2024. 04 ~ 2025. 07 (15개월, 3인 팀) | DN오토모티브 DX팀 | 기여도 70%

회사 내부 프로젝트

개요

반복적이고 복잡한 업무를 자동화하여 비부가가치 업무 시간을 단축하고 업무 효율성을 극대화한 전사 RPA 프로젝트입니다. DN오토모티브 DX팀 2명, 외부 RPA 개발자 1명과 협업하여 총 44건 중 12건을 전담 설계/개발했습니다.

역할

- 솔루션 구축: 1 차/2 차 프로젝트 총 44 건 중 12 건 전담 설계 및 개발
- RPA 연동 환경 구축: ERP, SAP, MES, 사내 보안 프로그램, Excel 등 업무 시스템과 RPA 연동
- 최적화: 시스템 안정화 모니터링, 성능 최적화, 프로세스 변경에 따른 유지보수
- 요구사항 분석 및 프로젝트 관리: 2 차 프로젝트에서 전사 업무 현황 조사를 통해 자동화 과제 도출 및 개발 우선순위 선정, 프로젝트 일정 수립

Skills

- **Tools** : Microsoft Power Automate (Desktop/Cloud), ERP, SAP, MES 등 사내 전용 웹시스템, Microsoft Excel
- **Environment** : Windows Server, Microsoft 365, 사내 전용 메일 서버

구축 상세

1. 해외 무역 Part - 납가 및 환율 정보 작성 업무

AS-IS

매일 아침 외부 사이트에서 납가·환율 정보를 수동으로 수집하고 당월 변동사항을 계산하여 담당자에게 메일을 발송하는 반복 업무가 존재했으며, RPA 자동화 이후에는 외부 사이트의 날짜 업데이트 지연·형식 불일치로 인한 데이터 오기입 리스크가 운영 중 발생했습니다.

TO-BE

납가·환율 정보 수집부터 변동사항 계산 및 메일 발송까지 전 과정을 자동화했습니다. 당일 날짜 기반 데이터 정합성 검증 로직을 추가하여 유효하지 않은 데이터 발생 시 수동 처리 안내 및 예외 리스트를 생성함으로써 업무 데이터의 정확도와 신뢰성을 확보했습니다.

2. 영업 Part - 출고의뢰서 작성 업무

AS-IS

지역별 출고 우선순위 결정부터 LOT 수량 집계, 배차 대수 작성, 담당자 메일 발송까지 전 과정을 수동으로 처리하여 매일 반복되는 업무 부담이 존재했습니다.

TO-BE

지역별 출고 우선순위 알고리즘과 LOT 수량 집계 로직을 구현하여 출고 리스트 파일 생성부터 메일 발송까지 전 과정을 자동화했습니다. 비정형 데이터 포함 시 워크플로우가 중단되는 문제는 예외 처리 로직으로 해결하여 비정형 데이터 발생 시에도 워크플로우가 중단 없이 지속되도록 했습니다.

3. 품질보증 Part - 검사 성적서 작성 업무

AS-IS

주 3회 업체별 검사 성적서 작성 시, 한 달 분량의 전체 시트를 눈으로 직접 확인하며 업체별 최신 전송 날짜를 찾아 수동으로 입력해야 했습니다. 데이터 누락·오기입 리스크와 함께 상당한 시간이 소요되는 반복 업무였습니다.

TO-BE

검사 성적서 전체 시트를 자동으로 읽어 조건값에 따라 업체별 최신 전송 날짜를 추출하고 당일 제품별 성적서에 자동 기재하도록 구현했습니다. 수동 입력 제거로 오기입 오류가 사라졌으며, 현업 담당자로부터 업무 효율이 크게 향상되었다는 직접적인 피드백을 받았습니다.

프로젝트 결과

1. 정량적 성과

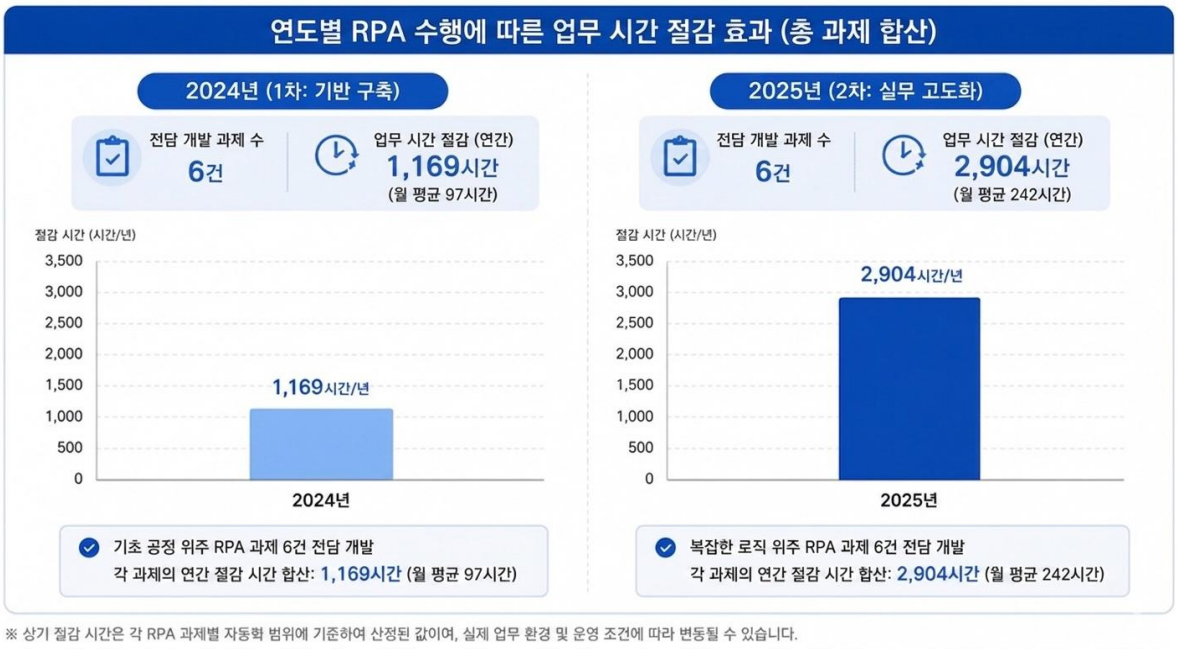

그림 5. 1차, 2차 프로젝트 업무 시간 절감 그래프

2. 운영 역량 및 마인드셋 형성

- **다각적 요구사항 분석 및 현업 소통**

자금무역, 생산관리 등 다양한 부서의 비즈니스 프로세스를 분석하여 사용자 니즈에 최적화된 요구사항을 도출하고 현업과 기술적으로 소통하는 역량 확보

- **체계적인 장애 대응 프로세스**

시스템 오류 발생 시 장애 공유 절차 및 단계별 대응 프로토콜을 숙지하여 현업의 혼선을 최소화하고 복구 시간을 단축하는 실무 대처 능력 배양

- **시스템 가용성 및 무중단 운영 마인드**

RPA 서버 및 스케줄링 상태를 상시 모니터링하며 서비스 무중단 운영의 중요성을 몸소 깨달았으며, 장애 복구 후 Double-check를 통한 재발 방지 습관 형성

- **자기주도적 문제 해결 마인드셋**

예상치 못한 장애 상황에서도 침착한 상황 판단과 투명한 소통이 해결의 핵심임을 체득