전유진

SERVER ENGINEER / PLATFORM ENGINEER

네트워크의 흐름을 이해하고,
운영 가능한 서버와 플랫폼을 구축합니다.

Linux 서버와 네트워크 기본기에서 출발해 Docker와 Kubernetes 기반의 서비스 운영 환경을 구축해왔습니다. 온프레미스 클러스터 구성, 서비스 트래픽 연결, 관측성, GitOps, AWS 인프라 설계까지 직접 경험하며 장애가 발생했을 때 시스템의 흐름을 따라 원인을 좁히는 역량을 길렀습니다.

CORE	NETWORK	OPERATIONS
Linux	TCP/IP	Terraform · Argo CD
Kubernetes	Calico · Ingress	Prometheus · Grafana
AWS	Istio · OpenVPN	Loki · Tempo
Docker	WebSocket	GitHub Actions

ujin5712@gmail.com

PROFILE

서버에서 플랫폼까지,
서비스의 전체 흐름을 봅니다.

무인이동체공학과 컴퓨터공학을 복수전공하며 애플리케이션, 운영체제, 네트워크와 데이터베이스의 기초를 쌓았습니다. 이후 Linux와 컨테이너 환경에서 직접 서비스를 구축하며 관심을 Server·Platform Engineering으로 확장했습니다. 특정 도구를 나열하기보다 요청이 들어와 처리되고 저장되는 경로를 이해하고, 장애 시 어느 계층에서 문제가 생겼는지 설명할 수 있는 엔지니어를 지향합니다.

다룰 수 있는 기술

SERVER · CONTAINER	Linux · Ubuntu · Rocky Linux · Docker · Docker Swarm · Kubernetes · Kubespray · Ansible
CLOUD · IaC	AWS · EKS · EC2 · VPC · ECR · S3 · SNS/SQS · Terraform
NETWORK · SECURITY	TCP/IP · Calico · MetalLB · NGINX Ingress · Istio · OpenVPN · iptables · NAT · WebSocket · SSL/TLS
CI/CD · GITOPS	Argo CD · GitHub Actions · Jenkins · Harbor · ECR · SonarQube · Git
OBSERVABILITY	Prometheus · Grafana · Loki · Tempo · Alloy · OpenTelemetry · ADOT · Kiali
PROGRAMMING · DATA	Python · C/C++ · Java · JavaScript · Bash/Shell · PostgreSQL · MariaDB · MongoDB · Redis · Kafka

대표 프로젝트

01	**DAI RUN**	온프레미스 Kubernetes 운영부터 AWS EKS 이전 설계까지
02	**Kubernetes 이커머스**	클러스터·네트워크·관측 환경 구축과 장애 분석
03	**Docker Swarm 3-Tier**	분산 배포, CI/CD, Load Balancing과 Observability
04	**자율주행 안내 로봇**	웹·WebSocket·ROS2·AI 모듈을 연결한 시스템 통합

PROJECT 01

AI 러닝 종합 플랫폼 DAI RUN

2026.06.30 - 2026.08.26 | 4명 | 팀장 · DevOps / Infra / Observability
서비스 보러가기

위치, 날씨·대기질, 러닝 기록과 사용자 선호를 이용해 당일 러닝 거리와 강도, 코스를 추천하는 AI 기반 러닝 플랫폼입니다. 초기 Docker 기반 서비스를 MSA와 Kubernetes 환경으로 전환하고, 온프레미스 운영 환경을 안정화한 뒤 AWS EKS 기반 아키텍처로 확장했습니다.

Kubernetes	AWS EKS	Terraform	Argo CD	Istio Ambient	Prometheus
Grafana	Loki	Tempo	KEDA	PostgreSQL	SNS/SQS

Kubernetes 기반 MSA 운영 환경

- 12개 MSA를 Kubernetes에 배포하고 애플리케이션, 데이터베이스, Kafka, 검색 및 관측성 워크로드를 Namespace와 노드 역할에 따라 분리했습니다.
- Calico 기반 Pod 네트워크와 MetalLB, Istio Gateway를 연결해 외부 요청이 Gateway > Service > Pod로 전달되는 구조를 구성했습니다.
- Istio Ambient Mode의 ztunnel을 적용해 Sidecar 없이 서비스 간 트래픽을 관리하고, Kiali로 서비스 토폴로지와 통신 상태를 확인했습니다.
- PostgreSQL, Redis, MongoDB, Kafka 등 상태 저장 워크로드의 배치와 스토리지를 고려해 애플리케이션 계층과 데이터 계층을 구분했습니다.

Metric · Log · Trace 통합 관측

- Prometheus와 Grafana로 클러스터 및 애플리케이션 지표를 수집하고 Loki와 Tempo를 연결해 로그와 Trace를 함께 조회할 수 있도록 구성했습니다.
- 장애가 발생하면 단순히 Pod를 재시작하지 않고 Gateway > Service > Pod > DB 순서로 요청 경로를 확인해 문제 구간을 좁혔습니다.
- 서비스별 HPA를 적용하고 k6 부하 테스트를 통해 CPU·Memory 사용량 증가에 따라 Pod 수가 확장되는 과정을 검증했습니다.

운영 관점

기술을 많이 사용하는 것보다 서비스 요구사항과 운영 환경에 맞는 기술을 선택하는 데 집중했습니다. 관측 도구 역시 대시보드를 만드는 목적이 아니라, 장애 시 어느 계층을 먼저 확인할지 판단할 수 있도록 연결했습니다.

GitOps와 AWS 이전까지 연결한 운영 구조

Argo CD 기반 GitOps

- 애플리케이션 배포 구성을 Git에서 관리하고 Argo CD가 클러스터의 실제 상태를 동기화하도록 구성했습니다.
- 배포 정의와 실행 환경을 분리해 변경 이력을 추적하고, 선언 상태와 실제 상태의 차이를 확인할 수 있도록 했습니다.
- 워크로드 특성에 따라 HPA, KEDA, Karpenter의 적용 범위를 비교하며 Pod와 Node 확장을 분리해 설계했습니다.

AWS EKS 이전 및 하이브리드 인프라

- 10.10.0.0/16 메인 VPC와 10.20.0.0/16 CI/CD VPC를 분리하고 Public, Application, Data Subnet으로 역할을 구분했습니다.
- NAT Gateway 의존도를 줄이기 위해 S3 Gateway Endpoint와 ECR·EC2·STS·EKS Auth Interface Endpoint를 사용하는 Private Network 구조를 설계했습니다.
- VPC, Subnet, EKS, VPC Endpoint 등 주요 인프라를 Terraform 코드로 관리했습니다.
- Harbor는 ECR, MinIO는 S3, Kafka 기반 이벤트 흐름은 SNS/SQS와 DLQ로 전환하는 방안을 설계했습니다.
- Site-to-Site VPN을 이용해 온프레미스와 AWS를 연결하는 하이브리드 네트워크를 구성했습니다.

트러블슈팅 · ETCD 디스크 장애

Kubernetes 클러스터 운영 중 ETCD 데이터가 저장된 디스크에 장애가 발생해 kubectl 명령이 정상적으로 동작하지 않았습니다. 보유한 마지막 Snapshot은 약 이틀 전의 상태였기 때문에 그대로 복원하면 팀원들의 최근 작업을 잃을 수 있었습니다.

1 · 원인 확인	Pod나 Service를 수정하기 전에 API Server와 ETCD의 연결을 추적했습니다. ETCD 디스크가 원인이었지만 실행 중인 Endpoint에는 아직 접근할 수 있음을 확인했습니다.
2 · 상태 보존	재시작보다 현재 상태 보존을 우선해 실행 중인 ETCD에서 최신 Snapshot을 새로 확보하고 상태를 검증했습니다.
3 · 이관과 검증	정상 디스크를 사용하는 다른 노드로 데이터를 이관한 뒤 API Server 응답과 Node·Pod 등 리소스 조회까지 검증했습니다.

결과	이틀치 작업을 잃지 않고 클러스터를 정상화했습니다. 장애 대응에서는 빠른 재시작보다 현재 남아 있는 복구 가능성을 먼저 판단해야 한다는 점을 배웠습니다.

PROJECT 02

Kubernetes 기반 헬스용품 이커머스 플랫폼

2026.05.26 - 2026.06.12 | 4명 | 팀장 · Kubernetes Infrastructure / Network / Observability

온프레미스 환경에 MSA 헬스용품 이커머스 서비스를 배포하기 위해 Kubernetes 클러스터와 네트워크, CI/CD, 관측 환경을 구축했습니다. 제한된 VM 자원 안에서 서비스 배치와 외부 연결, 데이터 영속성, 장애 확인 방법을 함께 고려했습니다.

Kubernetes	Kubespray	Ansible	Calico	MetalLB	NGINX Ingress
Jenkins	Harbor	Prometheus	Grafana	Loki	Tempo

클러스터와 서비스 네트워크

- Ubuntu VM을 Control Plane과 Worker 역할로 나누고 Kubespray·Ansible 기반으로 Kubernetes 클러스터를 구성했습니다.
- Calico VXLAN을 적용해 서로 다른 Node에 위치한 Pod가 통신할 수 있도록 구성했습니다.
- LoadBalancer 구현이 없는 온프레미스 환경에서 MetalLB로 외부 IP를 할당하고 NGINX Ingress가 Host·Path에 따라 요청을 분기하도록 했습니다.
- Pod CIDR, Service, Ingress와 외부 IP의 관계를 확인하며 사용자 요청이 애플리케이션까지 전달되는 전체 경로를 점검했습니다.

배포와 관측 환경

- Jenkins > Harbor > Kubernetes로 이어지는 이미지 빌드 및 배포 흐름을 구성했습니다.
- Prometheus, Grafana, Loki, Tempo, Alloy를 연결해 Node·Pod·애플리케이션 상태를 한곳에서 확인할 수 있도록 했습니다.
- Metrics Server, Node, Network, ETCD 관련 장애를 Event와 Log, 구성 요소 상태, 통신 경로 순으로 확인했습니다.

장애 확인 흐름

01	상태	get으로 Pod·Node·Service의 현재 상태 확인
02	이벤트	describe에서 Scheduling·Mount·Probe·CNI 관련 Event 확인
03	실행	Container Log와 Process, Resource 상태 확인
04	통신	Endpoint·Service Selector·Ingress·Node Network 순서로 점검

결과

Calico, MetalLB, NGINX Ingress의 역할과 연결 관계를 실제 트래픽 흐름으로 이해했습니다. Cluster, Network, Observability 장애를 계층별로 분석하며 Kubernetes 트러블슈팅 역량을 높였습니다.

PROJECT 03

Docker Swarm 기반 3-Tier 웹 서비스

2026.04.27 - 2026.05.15 | 4명 | 팀장 · Infrastructure / CI/CD / Monitoring

기존 웹 서비스를 Docker Image로 만들고 4대의 Ubuntu VM으로 구성한 Docker Swarm 환경에 분산 배포했습니다. Frontend, Backend, Database를 3-Tier로 분리하고 서비스 이중화, Load Balancing, 자동 배포와 모니터링을 적용했습니다.

Ubuntu	Docker	Docker Swarm	Nginx	GitHub Actions	SonarQube
Harbor	Prometheus	Grafana	Loki	Tempo	FastAPI

다중 노드 배포 구조

- 4대의 Ubuntu VM으로 1 Manager·3 Worker Docker Swarm Cluster를 구성했습니다.
- Frontend와 Backend를 각각 2개 Replica로 배포하고 Nginx Reverse Proxy와 Swarm Load Balancing을 통해 요청을 분산했습니다.
- web-net, db-net, monitor-net Overlay Network를 역할별로 분리해 서비스가 필요한 네트워크에만 참여하도록 구성했습니다.
- MariaDB, MongoDB, Redis를 별도 노드에 배치하고 Volume을 이용해 컨테이너 재생성 후에도 데이터를 유지했습니다.
- Health Check와 Rolling Update·Rollback 정책을 적용해 배포 중 서비스 중단 위험을 줄였습니다.

CI/CD와 Observability

CODE	>	QUALITY	>	IMAGE	>	DEPLOY
GitHub		SonarQube		Build · Harbor		Swarm Update

- GitHub Actions > SonarQube > Docker Build > Harbor > Docker Swarm으로 이어지는 자동 배포 Pipeline을 구성했습니다.
- Prometheus, Grafana, Loki, Tempo로 Host·Container·Application의 Metric, Log, Trace를 확인했습니다.

결과

기존 단일 웹 서비스를 4-Node 분산 구조로 전환하고 Replica와 Load Balancing을 적용했습니다. 컨테이너 실행뿐 아니라 네트워크, 데이터 영속성, 배포 안정성과 관측성을 함께 고려하는 운영 관점을 익혔습니다.

PROJECT 04

시각장애인을 위한 자율주행 경로 안내 로봇

2024.09 - 2024.12 | 3명 | Web · WebSocket Communication · Traffic Light Detection

사용자가 웹에서 목적지를 지정하면 경로 데이터를 로봇에 전달하고, 로봇이 횡단보도 도착 여부와 신호등 상태를 확인하며 안전하게 주행하도록 만든 프로젝트입니다. 웹 입력부터 실시간 통신, ROS2 Topic, 신호등 인식 결과와 보행 판단까지 여러 모듈을 하나의 데이터 흐름으로 연결했습니다.

ROS2 Foxy	JavaScript	WebSocket	roslibjs	Tmap API	C++
Python	YOLOv8	OpenCV	HSV	Moving Avg.	CAN

웹에서 로봇까지 이어지는 데이터 흐름

- 시각장애인 사용자와 보호자가 출발지와 목적지를 입력하고 검색 결과를 확인할 수 있는 웹 화면을 구현했습니다.
- Tmap API에서 보행 경로와 좌표 정보를 받아 로봇이 사용할 수 있는 데이터로 정리했습니다.
- roslibjs와 ROS2 Web Bridge를 이용해 브라우저에서 생성한 출발지·목적지 및 경로 좌표를 ROS2 Topic으로 발행했습니다.
- WebSocket으로 웹과 로봇 사이의 데이터를 실시간 전달하고, 수신한 경로가 GPS 기반 경로 추종 로직에 이어지도록 구성했습니다.
- 웹, 지도 API, ROS2, Jetson, 주행 제어 모듈의 데이터 형식과 전달 순서를 맞추며 전체 시스템을 통합했습니다.

핵심 기여

웹 페이지 구현 자체보다 사용자의 입력이 네트워크를 거쳐 로봇의 실제 움직임으로 이어지게 하는 데 집중했습니다. 좌표 형식, Topic, 연결 순서가 맞지 않으면 전체 시스템이 동작하지 않았기 때문에 인터페이스를 정의하고 끝단까지 데이터가 전달되는지 반복해서 검증했습니다.

신호등 인식 및 안전한 보행 판단

- Tmap API에서 받은 횡단보도 위치와 로봇의 현재 GPS 좌표를 비교해 횡단보도에 도착한 경우에만 신호등 판단 로직이 동작하도록 했습니다.
- YOLOv8 Object Detection으로 신호등을 탐지하고, 탐지된 신호등 영역만 Crop해 HSV Filter의 입력으로 사용했습니다.
- Crop 이미지에서 Red·Green 색상 범위를 판별하고, 최근 판단 결과에 이동평균 필터를 적용해 평균값이 0.5를 초과할 때만 Green으로 인정했습니다.
- Red를 확인한 경우 red_flag를 1로 설정하고, 이후 Green으로 전환되면서 이동평균 조건까지 충족한 경우에만 GO 판단을 내리도록 구성했습니다.
- 처음 인식한 신호가 이미 Green이면 남은 시간이 짧을 수 있으므로 출발하지 않고, Red에서 Green으로 바뀐 시점에만 보행을 시작하도록 해 판단 안정성을 높였습니다.

수상

제13회 공학교육인증 창의설계경진대회 대상 · 2024 소프트웨어융합대학 학술제 우수상

PROJECT 05

OpenVPN 기반 폐쇄망 접속 환경

2026.02.26 - 2026.03.30 | 개인 네트워크 프로젝트

외부 Client가 인터넷에 직접 노출되지 않은 내부 Server에 안전하게 접근하도록 OpenVPN 기반 암호화 터널을 구축했습니다. Linux - Linux와 Linux - Windows 환경을 구성하며 인증, 라우팅, 패킷 포워딩, 방화벽까지 직접 설정했습니다.

OpenVPN	Rocky Linux	Windows	VirtualBox	iptables	TCP/IP
SSL/TLS	X.509	NAT	Routing		

구성 및 구현

- VirtualBox에서 외부 Client, VPN Server, 폐쇄망 Server 역할의 가상 네트워크를 구성했습니다.
- VPN Server와 Client 설정을 작성하고 인증서, 공개키·개인키 기반의 상호 인증 환경을 구축했습니다.
- VPN 접속자에게 가상 IP를 할당하고 Client 요청이 VPN Server를 거쳐 내부망 Server까지 전달되도록 Routing과 IP Forwarding을 설정했습니다.
- iptables의 FORWARD 및 NAT 규칙을 구성해 허용된 VPN 트래픽만 내부망으로 전달되도록 했습니다.
- Linux Client뿐 아니라 Windows Client에서도 VPN 연결과 내부망 접근을 검증했습니다.

문제 해결 · 연결됐지만 내부망에는 닿지 않는 문제

VPN 연결이 성공해도 내부 Server에 접근할 수 없는 경우가 있었습니다. 인증 성공 여부만 보는 대신 Client의 Routing Table, VPN Interface, Server의 IP Forwarding, iptables 정책과 내부 Server의 응답 경로를 차례로 확인했습니다.

CLIENT	>	VPN TUNNEL	>	FORWARD / NAT	>	PRIVATE SERVER
Route 확인		가상 IP·인증		iptables·IP Forward		응답 경로 확인

배운 점

터널 생성과 End-to-End 통신은 별개의 단계이며 요청과 응답 패킷의 경로가 모두 완성돼야 통신이 성립합니다. VPN을 인증 > 터널 > 라우팅 > 방화벽 > 응답 경로의 흐름으로 이해하게 됐습니다.

PROJECT 06

Java Socket 기반 SMTP 메일 클라이언트

2024.09.27 - 2024.11.21 | 4명 | SMTP Client 설계·구현

외부 메일 라이브러리 없이 Java Socket과 SSLSocket으로 Gmail SMTP Server에 직접 연결해 메일을 전송했습니다. RFC 5321의 명령과 응답 코드를 기준으로 SMTP Session을 구현하며 애플리케이션 계층 프로토콜의 실제 동작을 확인했습니다.

Java	Socket	SSLSocket	SMTP/ESMTP	SSL/TLS	Base64
Gmail SMTP	RFC 5321				

프로토콜 흐름 직접 구현

연결	인증	전송	종료
SSLSocket 220 확인	HELO · AUTH LOGIN 250 · 235 확인	MAIL FROM · RCPT TO · DATA 354 확인	QUIT 221 확인

- smtp.gmail.com:465에 SSLSocket으로 연결하고 Server의 220 응답을 확인해 세션을 시작했습니다.
- AUTH LOGIN에서 사용자 정보를 Base64로 인코딩하고 235 응답으로 인증 성공을 확인했습니다.
- MAIL FROM, RCPT TO, DATA 순서로 발신자·수신자·메일 본문을 전달했습니다.
- 220, 250, 354, 235, 221 등 단계별 Status Code를 검증해 요청 성공 여부와 세션 상태를 처리했습니다.

트러블슈팅 · SSL/TLS 적용 이후 인증 실패

일반 Socket 통신에서 확인한 흐름을 그대로 적용했지만 보안 인증 단계에서 통신이 반복적으로 실패했습니다. 다른 라이브러리로 우회하지 않고 RFC 문서와 Server 응답 코드를 확인하고, Telnet을 이용한 평문 통신과 SSL/TLS 연결의 차이를 비교했습니다. 보안 적용 이후의 인증 명령과 통신 순서를 규격에 맞게 수정해 메일 송신에 성공했습니다.

배운 점

TCP 연결 위에서 애플리케이션 계층 프로토콜이 명령과 상태 코드로 Session을 관리하는 방식을 이해했습니다. 오류 메시지를 현상으로만 보지 않고 공식 규격과 실제 응답을 비교해 원인을 찾는 습관을 길렀습니다.

OTHER PROJECTS

애플리케이션과 데이터 계층의 이해

군자동 음식점 DB 관리 프로그램

2024.03 - 2024.06 | MSSQL · C++ · MFC · ODBC

1,000건 이상의 음식점 리뷰 데이터를 대상으로 ER 모델링부터 MSSQL 구축, C++ MFC와 ODBC를 이용한 조회·관리 화면까지 구현했습니다.

부동산 시세·생활 인프라 DB

2026.03.31 - 2026.04.13 | MariaDB · MongoDB

법정동 코드를 기준으로 부동산 시세와 생활 인프라 데이터를 연결하고 데이터 특성에 따라 관계형 DB와 문서형 DB를 구분했습니다.

상권 추천 서비스 '상추'

2026.04.14 - 2026.04.24 | Python · FastAPI · MariaDB · MongoDB

서울시 상권 데이터를 분석해 조건에 맞는 정보를 제공하는 서비스의 백엔드 API와 데이터 연결을 구현했습니다.

EDUCATION & QUALIFICATIONS

기반을 넓히고, 직접 구축하며 확인했습니다.

학력	세종대학교 · 무인이동체공학 / 컴퓨터공학 복수전공
교육	더존비즈온 Cloud DX Academy 6기 · 2026.02.25 - 2026.08.26
자격	정보처리기사 · SQLD · 네트워크관리사 2급 · 리눅스마스터 2급
어학	TOEIC 920 · TOEIC Speaking 120

서비스가 정상적으로 배포되는 데서 끝내지 않고,
장애가 발생했을 때 원인을 확인하고 복구할 수 있는 운영을 만들겠습니다.

CONTACT

ujin5712@gmail.com