



DEVOPS / CLOUD ENGINEER

김준우 Joon Woo Kim

사람과 환경에 대한 관심에서 시작해,  
제작 시스템을 거쳐 인프라와 자동화를 만들고 있습니다.

joon@junu.dev

junu.dev

github.com/Vivacissimo

wiki.junu.dev

Portfolio · resume.junu.dev

## 01 INTRODUCE

현재는 **Cloud Native Infrastructure**와 **DevOps**에 관심을 두고, Kubernetes 기반 서비스 운영과 CI/CD, IaC, Observability를 중심으로 경험을 넓히고 있습니다.

사회복지에서 출발해 VFX 분야로 옮긴 뒤, 결과물보다 그것이 만들어지는 **환경과 시스템**에 더 큰 흥미를 느꼈습니다. Render Farm과 Linux 제작 환경을 운영하며 반복 작업을 자동화했고, 관심사는 **Pipeline Automation** → **IT Infrastructure** → **Cloud / DevOps**로 이어졌습니다.

반복 가능하고, 관찰 가능하며, 운영할 수 있는 시스템을 만드는 엔지니어를 지향합니다.

## 02 EXPERIENCE

**2026.03 — 2026.10** **KT Cloud TECH UP · 클라우드 네이티브 과정 수료**  
팀 프로젝트 대상 · 우수 수료자

**2024.01 — 2026.03** **Cocoavision · VFX본부 Pipeline / IT Infrastructure**  
Linux 제작 환경 · Render Farm · Pipeline 운영 및 자동화

**2022.09 — 2024.01** **Cocoavision · NewMedia본부 R&D팀**  
Unreal Engine · Blender 실시간 콘텐츠 · Previs Workflow R&D

## 03 SKILLS

|                              |                                        |
|------------------------------|----------------------------------------|
| Cloud                        | AWS                                    |
| Container & Orchestration    | Docker · Kubernetes · Helm             |
| CI/CD & GitOps               | Jenkins · GitHub Actions · ArgoCD      |
| Service Mesh & Observability | Istio · Prometheus · Grafana · Kiali   |
| IaC & Configuration          | Terraform · Ansible                    |
| Systems & Network            | Linux · Nginx · Cloudflare · PXE · Git |
| Scripting                    | Python                                 |

# MoongCheap · AI 역경매 공동구매 플랫폼

AWS EKS Multi-AZ 위의 FE · BE · AI 서비스를 공통 Chart와 GitOps 규칙 하나로 배포하는 구조를 만들고, 운영 중 생긴 병목을 실측으로 해결했습니다.

상세: [resume.junu.dev/projects/moongcheap](https://resume.junu.dev/projects/moongcheap)

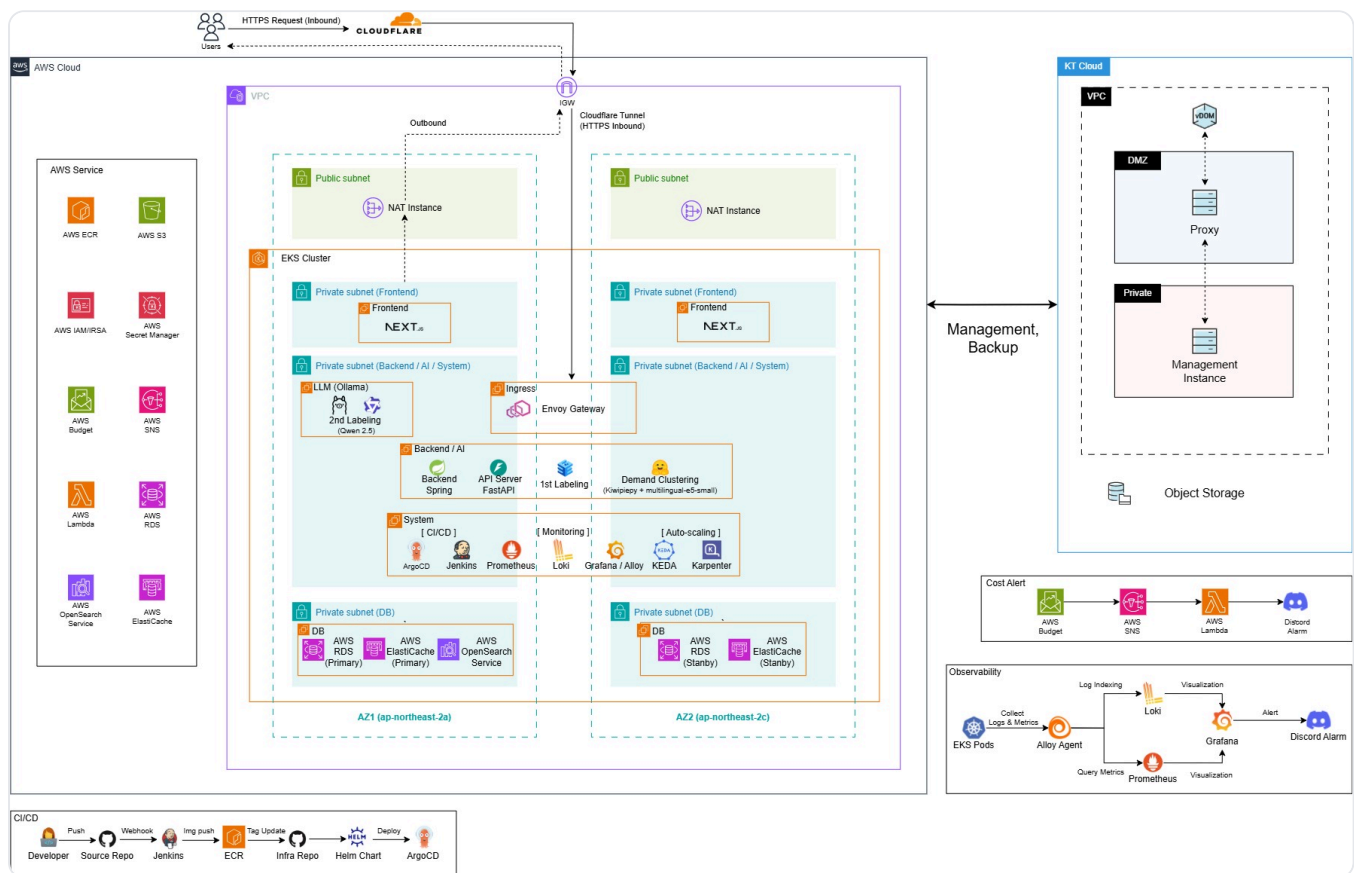
|                        |                     |                                |                                              |
|------------------------|---------------------|--------------------------------|----------------------------------------------|
| PERIOD<br>2026.08 - 10 | TEAM<br>Cloud 파트 5인 | ROLE<br>CI/CD · GitOps · AI 배포 | STACK<br>EKS · Jenkins · ECR · Helm · ArgoCD |
|------------------------|---------------------|--------------------------------|----------------------------------------------|

## Kubernetes 운영

## CI/CD · GitOps

## 장애 원인 분석

## AWS EKS



## 담당 파트

- Jenkins → ECR → GitOps Repo → ArgoCD ApplicationSet 흐름 설계, develop / prod / platform 별 Application 자동 생성
- FE · BE · AI가 **moongcheap-service** Chart 하나를 쓰고, 차이는 base → env → services → overrides 4단계 Values로만 관리
- Commit SHA 이미지 태그로 운영 중인 Pod가 어떤 커밋인지 바로 추적
- App Repo와 GitOps Repo를 분리하고, 서비스 팀이 넘길 항목(Port · Health Check · 환경 변수)을 정의해 배포 책임 경계를 정리
- AI 팀의 CPU · GPU · Memory · 추론 API 요구사항을 받아 같은 GitOps 흐름으로 AI · LLM 서비스 배포

## 트러블슈팅

### 노드에 여유가 있는데 ArgoCD · Jenkins가 Pending

노드당 Pod 한도 17개로 **Too many pods** → VPC CNI Prefix Delegation · **maxPods: 110**을 인프라를 내리는 시간대 (평일 22시 이후)에 무중단 적용 **17 → 110개**

### 인프라를 켜 뒤 CI 서버가 1시간 반 넘게 안 열림

기동마다 플러그인 100개 재설치 → **initializeOnce: true** 한 줄 **96분 → 2.6분**

### 백엔드 빌드 1회 45분

빌드 Pod 메모리 과다 요청으로 빌드마다 노드 4번 추가 → Pod 46개 실측으로 요청값 재산정 **45분 → 16분**

CI 기동

**97% 단축**

96분 이상 → 2.6분

BUILD

**64% 단축**

노드 추가 4회 → 1회

POD LIMIT

**6.5배**

노드 추가 비용 없이

## 회고

### 빌드 시간 단축을 리소스 조정에서 멈춤

요청값 조정으로 백엔드 빌드를 45분에서 16분으로 줄였지만, 캐시나 병렬화처럼 파이프라인 자체를 개선하는 단계까지는 가지 못했습니다.

**다시 한다면** 의존성과 이미지 레이어 캐시를 도입하고, 변경된 서비스만 빌드하도록 파이프라인을 나누겠습니다. 빌드 시간을 지표로 수집해 개선 효과를 계속 추적하겠습니다.

### 팀별 CI 파이프라인의 통일성 부족

CD는 공통 Helm Chart와 ApplicationSet 규칙으로 맞췄지만, CI는 Frontend · Backend · AI 팀마다 파이프라인 구성이 달라 단계와 규칙이 일관되지 않았습니다.

**다시 한다면** Build · Scan(Trivy · gitleaks) · Push · Tag 갱신 단계를 공통 Jenkins 템플릿(Shared Library)으로 만들고, 각 팀은 언어별 빌드 설정만 넘기도록 해 CI도 CD와 같은 수준으로 표준화하겠습니다.

# Hotel Booking · GitOps & Observability

02

특가 오픈 순간 트래픽이 몰리는 호텔 예약 MSA를 Terraform으로 만든 EKS에 올리고, push 한 번으로 dev/prod까지 이어지는 배포 흐름을 만들었습니다.

상세: [resume.junu.dev/projects/hotel-booking](https://resume.junu.dev/projects/hotel-booking)

| PERIOD  | TEAM | ROLE             | STACK                                            |
|---------|------|------------------|--------------------------------------------------|
| 2026.07 | 3인   | 팀장 · 인프라 & CI/CD | EKS · Terraform ·<br>Git Hub Actions ·<br>ArgoCD |

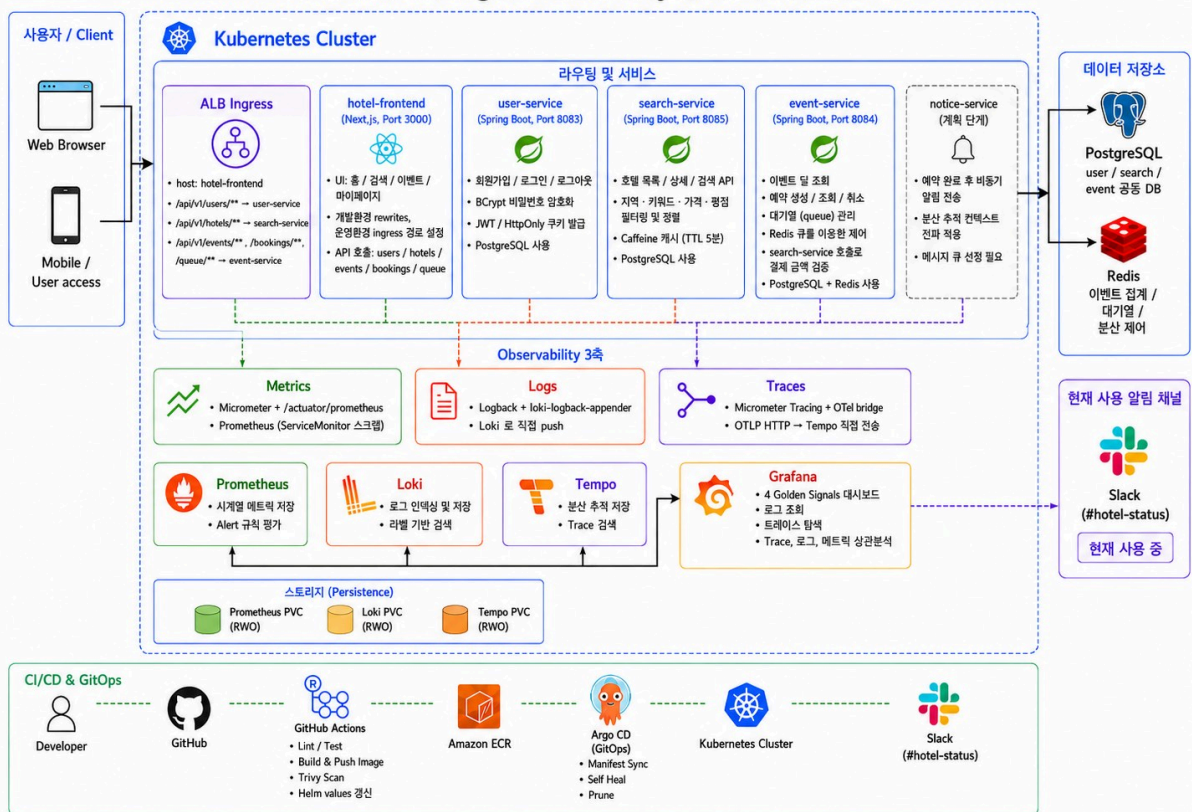
IaC (Terraform)

CI/CD · GitOps

모니터링 · 알림

AWS EKS

## Hotel Booking Observability 통합 시스템 구성도



### 담당 파트

- terraform-aws-modules/eks로 노드 그룹 · IRSA · Access Entry를 구성하고 ArgoCD도 Terraform Helm Release로 설치
- GitHub Actions에서 변경된 서비스만 Matrix 빌드 → Gradle 테스트 → Trivy HIGH/CRITICAL 차단 → ECR Push
- CI가 Helm values의 이미지 태그를 커밋하면 ArgoCD(App of Apps · prune · selfHeal)가 dev/prod에 자동 배포
- prod 파이프라인에 helm lint와 렌더링 검사를 넣어 dev 네임스페이스 · 브랜치 설정 유입을 차단

- 공통 Chart에 HPA(2-5) · 무중단 Rolling Update · ALB Ingress 경로 라우팅 적용

## 트러블슈팅

### ALB에서 로그인 후 401 반복

HTTP ALB에서 Secure 쿠키가 저장되지 않음 → 쿠키 속성을 환경 변수로 분리해 환경별 values로 주입

### 배포 직후 Pod 반복 재시작

초기화 전에 Liveness Probe가 실패 판정 → 공통값 `initialDelaySeconds` 45초 · `failureThreshold` 6

### 클러스터에서만 API 호출 실패

CORS 분산 · 로컬 주소 직접 호출 → CORS 공통화, 상대경로 호출, Ingress `/api/v1/*` 경로 라우팅

#### LOAD TEST

##### 오버셀 0건

K6 1,000명 · 재고 30

#### ERROR

##### 시스템 오류 0%

매진(409)은 정상 응답으로 분리

#### DEPLOY

##### 5개 서비스 · 2개 환경

push 한 번으로 자동 배포

## 회고

### 클러스터에서만 생기는 로그 · 트레이스 유실을 끝까지 좁히지 못함

로컬에서는 메트릭 · 로그 · 트레이스가 모두 연결됐지만, EKS에서는 일부 서비스 로그가 Loki로 전송되지 않고 일부 POST 요청의 Span이 Tempo에 남지 않은 채로 마무리했습니다.

**다시 한다면** 배포 후 검증 단계에 로그 · 트레이스 수집 여부 확인을 넣어, 환경 차이를 배포 시점에 바로 잡겠습니다.

### 버스트 트래픽의 응답시간 목표 미달

정합성은 지켰지만 1,000명이 동시에 몰리는 순간의 p95는 239ms로 목표를 넘었습니다. HPA는 CPU 기준이라 짧은 버스트에 대응하는 확장 전략은 검증하지 못했습니다.

**다시 한다면** 대기열 길이나 요청량 같은 지표 기반 스케일링과 특가 오픈 전 사전 확장을 함께 검토하겠습니다.

# Medical MSA · CI/CD & Service Mesh

03

의료 MSA의 CI/CD · GitOps 배포를 자동화하고, 심화 단계에서 Istio를 올려 트래픽 전환 · 장애 복구 · 내부 접근 차단을 직접 실험했습니다.

상세: [resume.junu.dev/projects/cloud-native](https://resume.junu.dev/projects/cloud-native)

| PERIOD       | TEAM | ROLE           | STACK                                    |
|--------------|------|----------------|------------------------------------------|
| 2026.05 - 06 | 4인   | CI/CD · K8s 운영 | GitHub Actions ·<br>ArgoCD · EKS · Istio |

Kubernetes 운영

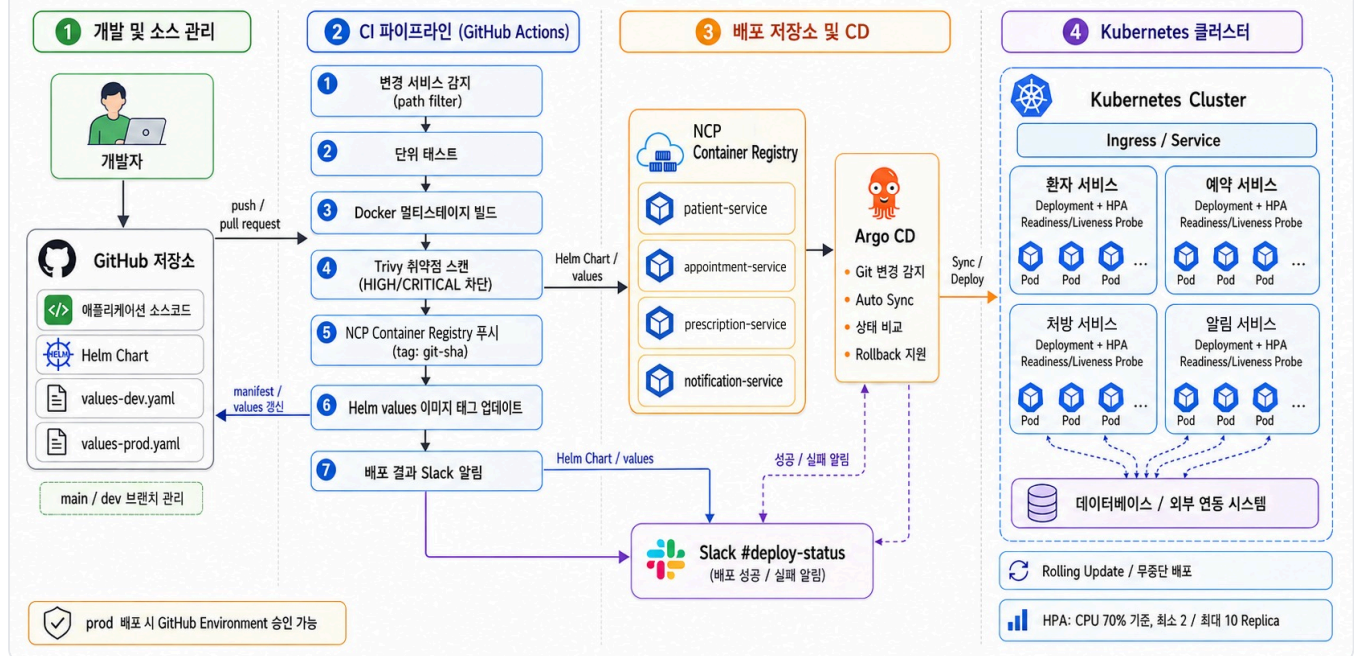
Service Mesh (Istio)

CI/CD

모니터링

## 쿠버네티스 기반 애플리케이션 배포 환경 구성도

CI: GitHub Actions / CD: Argo CD / Registry: NCP Container Registry



### 담당 파트

- Path Filter로 변경된 서비스만 Matrix 빌드, 테스트 실패 시 중단
- jdk → jre 멀티스테이지 · 비루트 실행 이미지, Trivy **exit-code 1**로 취약 이미지 레지스트리 유입 차단
- git-sha 태그 Push → Helm values 갱신 → ArgoCD **medical-dev / medical-prod** 분리 배포, 결과는 Slack으로 공유
- HPA(CPU 70%, 2-10) · Readiness/Liveness Probe · Rolling Update 구성, DB 장애 · Memory Leak 유도로 Probe 동작 검증
- 심화 단계: Prometheus로 K8s · Envoy 메트릭 수집, Grafana → Kiali → ArgoCD 장애 추적 Runbook 정리

### Canary · Blue/Green

VirtualService 가중치로 20 → 50 → 70 → 100% 단계 전환, 50:50 → 0:100 즉시 전환

### Pod 강제 종료 중 재시도

retries.attempts: 3 적용 후 `kubectl delete pod` → Envoy가 살아 있는 Pod로 재시도 **성공률 100%**

### Zero-Trust 접근 차단

mTLS STRICT · AuthorizationPolicy 상태에서 Mesh 밖 Pod의 직접 호출 → **RBAC: access denied**

### 빌드 아키텍처 차이로 배포 실패

QEMU · Docker Buildx 멀티 플랫폼 빌드로 해결

#### CI/CD

### 4개 서비스 자동 배포

Trivy 통과 이미지만 반영

#### RETRY

### 성공률 100%

Pod 강제 종료 중

#### CANARY

### 4단계 전환

20 → 50 → 70 → 100%

## 회고

### VPC · EKS 클러스터를 셀 스크립트로 생성

심화 단계에서 배포 파이프라인과 트래픽 제어에 집중하느라, VPC와 EKS 클러스터 같은 클라우드 자원은 Terraform 같은 코드가 아닌 셀 스크립트로 만들었습니다.

**다시 한다면** Terraform으로 클러스터 바깥 인프라까지 선언형으로 관리하겠습니다. 다음 프로젝트(Hotel Booking)에서 EKS와 ArgoCD를 Terraform으로 구성하며 이 부분을 이어서 해결했습니다.

### 리소스 산정과 권한 설정의 시행착오

비용 제한으로 레지스트리를 옮기고, 심화 단계에서는 프리티어 자원 제약과 IAM 사용자 권한 매핑 문제로 클러스터 생성과 접근 권한 확보가 반복해서 지연됐습니다.

**다시 한다면** 사이드카 비용을 포함한 리소스 총량과 비용을 먼저 산정하고, 팀원별 클러스터 접근 권한을 생성 시점에 코드로 함께 정의하겠습니다.

# Cocoavision · VFX Pipeline / Infrastructure

04

아티스트가 쓰는 DCC 환경과 Linux Render Farm을 운영하며, 반복되는 설치 · 설정 · 제출 작업을 자동화하고 제작이 멈추는 원인을 해결했습니다.

상세: [resume.junu.dev/projects/cocoavision](https://resume.junu.dev/projects/cocoavision)

| PERIOD            | TEAM                     | ROLE                            | STACK                                       |
|-------------------|--------------------------|---------------------------------|---------------------------------------------|
| 2022.09 – 2026.03 | VFX본부 · NewMedia<br>R&D팀 | Pipeline / IT<br>Infrastructure | Linux · Ansible · PXE ·<br>Python · Tractor |

IaC (Ansible)

Linux 서버 운영

Python 자동화 도구

장애 원인 분석

## Case 1 · Ansible Render Farm

- **문제** 수십 대 Render Node에 SSH로 접속해 패키지 · Mount · 팀별 Tool을 노드마다 수동 설정, 설정 편차 발생
- **해결** Inventory로 노드를 그룹화하고 반복 작업을 Playbook으로 선언해 일괄 적용
- **결과** 서버별 수동 작업을 재사용 가능한 운영 코드로 전환

## Case 2 · PXE 네트워크 설치

- **문제** USB로 장비마다 OS를 설치해, 장비가 늘수록 사람이 하는 단계가 증가
- **해결** PXE 부팅 · DHCP/HTTP 설치 파일 제공 · Kickstart 자동 설치를 연결
- **결과** 초기 세팅을 같은 절차로 반복할 수 있는 네트워크 설치로 전환

## Case 3 · Tractor 제출 도구

- **문제** 작업자가 렌더 제출 정보를 직접 구성하고 DCC마다 절차가 달라 누락 · 설정 차이 발생
- **해결** Houdini · Nuke용 Custom Submit Node를 Python으로 개발. 사용자 교육 대신 제출 규칙을 도구에 넣음
- **결과** DCC 안에서 일관된 제출 경로를 제공해 Submission Workflow 표준화

## 장애 대응 · Farm에서만 실패하는 렌더

- 로컬에선 열리지만 Farm 제출 후 반복 실패해 제작이 중단
- Tractor Log → DCC/Plugin 버전 → 환경 변수 → 파일 경로 → Mount 순으로 실행 환경 차이를 비교해 원인 특정
- 설정 수정 후 같은 유형이 다른 노드에서 반복되지 않도록 운영 절차와 자동화 대상에 반영

### FARM

#### 수십 대 노드

Linux Render Node 운영

### AUTOMATION

#### 3개 영역 자동화

Submit · Provisioning · Configuration

### WORK STYLE

#### Fix → Standardize → Automate

반복 이슈를 구조로 줄임

# junu.dev Home Infrastructure

05

Raspberry Pi에는 기술 위키를, Cloudflare Pages에는 웹 도구를 올려 서비스 특성에 맞게 배포 위치를 나눠 운영하고 있습니다.

상세: [resume.junu.dev/projects/personal-infra](https://resume.junu.dev/projects/personal-infra)

| PERIOD | TEAM    | ROLE                      | STACK                                                                                          |
|--------|---------|---------------------------|------------------------------------------------------------------------------------------------|
| 운영 중   | 개인 (1인) | Design · Deploy · Operate | Raspberry Pi · Docker<br>Compose · Cloudflare<br>Tunnel · Cloudflare<br>Pages · GitHub Actions |

CI/CD

Linux 서버 운영

네트워크 · DNS

컨테이너

이전 구조 · 웹 도구 Pi 직접 배포

GitHub Push

master 브랜치

Build Job

npm ci · lint · build

Artifact

dist 업로드

Pi Runner

Self-hosted · Label

Nginx

rsync --delete

Cloudflare

Tunnel

\*.junu.dev 공개

현재: [Cloudflare Pages](#)로 이전

## 구현 내용

이전 구조 · 웹 도구를 Pi에 직접 배포하던 시기에 구성

- Build Job이 lint · build를 통과한 결과물만 Artifact로 올리고 Deploy Job이 받아 반영하는 2단계 GitHub Actions 파이프라인 구성
- Pi에 Self-hosted Runner를 설치하고 **raspberrypi · deploy** · 서비스 이름 라벨로 배포 Job이 정해진 장비에서만 실행되게 구성
- rsync --delete**로 서비스 디렉터리를 빌드 결과와 똑같이 맞추고, **concurrency** 그룹으로 연속 push 시 마지막 변경만 배포

현재 구조

- 서버 처리가 필요 없는 웹 도구는 Cloudflare Pages로 이전하고, Pi에는 데이터베이스가 필요한 위키만 남겨 운영
- Cloudflare Tunnel로 인바운드 포트 없이 서브도메인별 서비스 공개
- Wiki.js · PostgreSQL을 Docker Compose로 분리하고 볼륨 · SQL Dump로 데이터 유지, 저장장치를 microSD에서 NVMe로 이전

## 운영 중인 서비스

### [wiki.junu.dev ↗](https://wiki.junu.dev) · 기술 위키

Wiki.js + PostgreSQL(Docker  
Compose)로 학습 문서를 정리하고 운영

### Personal Dev Tools · AI 에이전트로 개발 · Cloudflare Pages 배포

[maskly.junu.dev ↗](https://maskly.junu.dev) 레이어 마스크 · 브러시 · 알파 편집을 지원하는 웹  
이미지 편집기(React · Vite)

[transformer.junu.dev ↗](https://transformer.junu.dev) 이미지 · PDF · 텍스트 형식 변환을 서버 없이  
처리하는 브라우저 파일 변환기