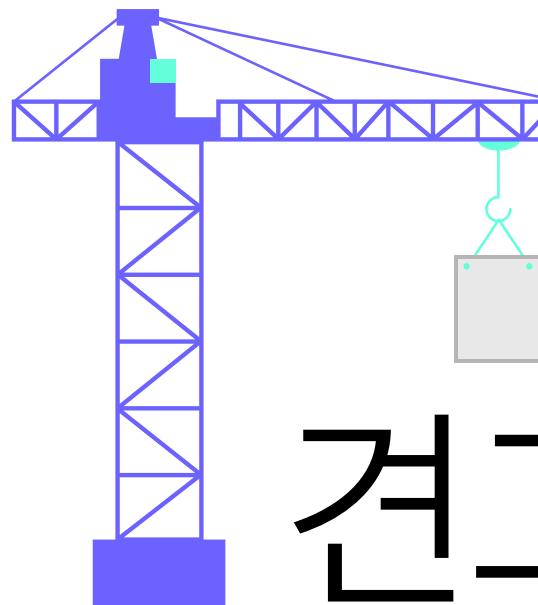


2025

FOUNDATION · CONSISTENCY · RELIABILITY



견고한 기초에서 시작해
높은 확장성으로 나아가다

CONTACT

PHONE : 010 6483 5892 | MAIL : rkddkwl059@naver.com

송진우

송진우 (1997.04.19) | <https://github.com/openSongce>

기본기가 튼튼 해야 서비스가 무너지지 않는다.

저는 **기반**을 다지고, 그 위에 **확장** 가능한 **서비스**를 올리는 **개발자**입니다.



송진우 Song-Jin-Woo

/ 1997.04.19

📞 010-6483-5892

✉️ rkddkwl059@naver.com

🌐 <https://github.com/openSongce>

학력 Education

2013.03 - 2016.02

경북고등학교 졸업

2016.03 - 2025.02

영남대학교 기계공학부 기계설계 졸업

복수전공 컴퓨터 공학 졸업

이력 Background

2023.03 - 2024.12

YEUNGNAM UNIVERSITY CONERGENCE S/W COURSE

2025.01 - 현재

삼성청년SW·AI아카데미(SSAFY) 13기

자격 Certifications

2025.03.15

OPIC 영어 INTERMEDIATE MID 1

2025.06.13

정보처리기사

어떤 역량을 가지고 있을까

| 스킬 및 역량

Java & Kotlin (Spring / Android)



MySQL · PostgreSQL



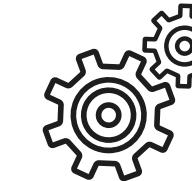
협업 및 코드리뷰



Jenkins · Docker · k8s · AWS



Prometheus · Grafana



CI/CD 구축 및 운영

Jenkins + Kaniko 기반 CI/CD 파이프라인 구축 경험

Docker 이미지를 자동 빌드·배포하고, k3s 클러스터에 Helm으로 롤링 업데이트 운영

NHCafe(AWS EC2에 수동 빌드·재실행 배포 경험),

BookgleBookgle(Compose 배포, AWS EC2), HelloWorld(k3s 이전/관찰성, AWS EC2)



백엔드 개발 경험

Spring Boot + JPA, REST/gRPC API 설계 및 구현

AWS S3 + Presigned URL 기반 미디어 업로드/다운로드 API

OpenAI DALL·E-3 연동, 캐리커처 생성 후 이미지 저장

사용자 일기(텍스트·이미지) 및 미디어 저장소 설계

BookgleBookgle(gRPC 동기화), HelloWorld(S3 + DALL·E-3 API, 일기/사진 저장,

MSA)



Observability 구축

Prometheus + Grafana 기반 지표 모니터링

Loki + Promtail 로그 수집 파이프라인

알람 규칙 기반 자동 탐지 및 대응 프로세스 단축

BookgleBookgle (기본 대시보드 구성 및 성능 지표 추적),

HelloWorld (Prometheus/Grafana 모니터링, Loki 로그 수집)

01. AI 아이디어 협업 마인드맵 서비스

o-O

DevOps/Backend

02. 임산부 케어 & 빅데이터 추천 서비스

HelloWorld

DevOps/Backend

03. 함께 읽고 토론하는 실시간 PDF 협업

BookgleBookgle

Android/Infra

04. 음성만으로 주문 완료하는 무인 카페 경험

NHCAFE

Android/AI/Infra

05. 지도로 계획하고 예산까지 챙기는 여행 플래너

RouteMap

Android

06. 장소 기반 소통을 담는 지도형 SNS

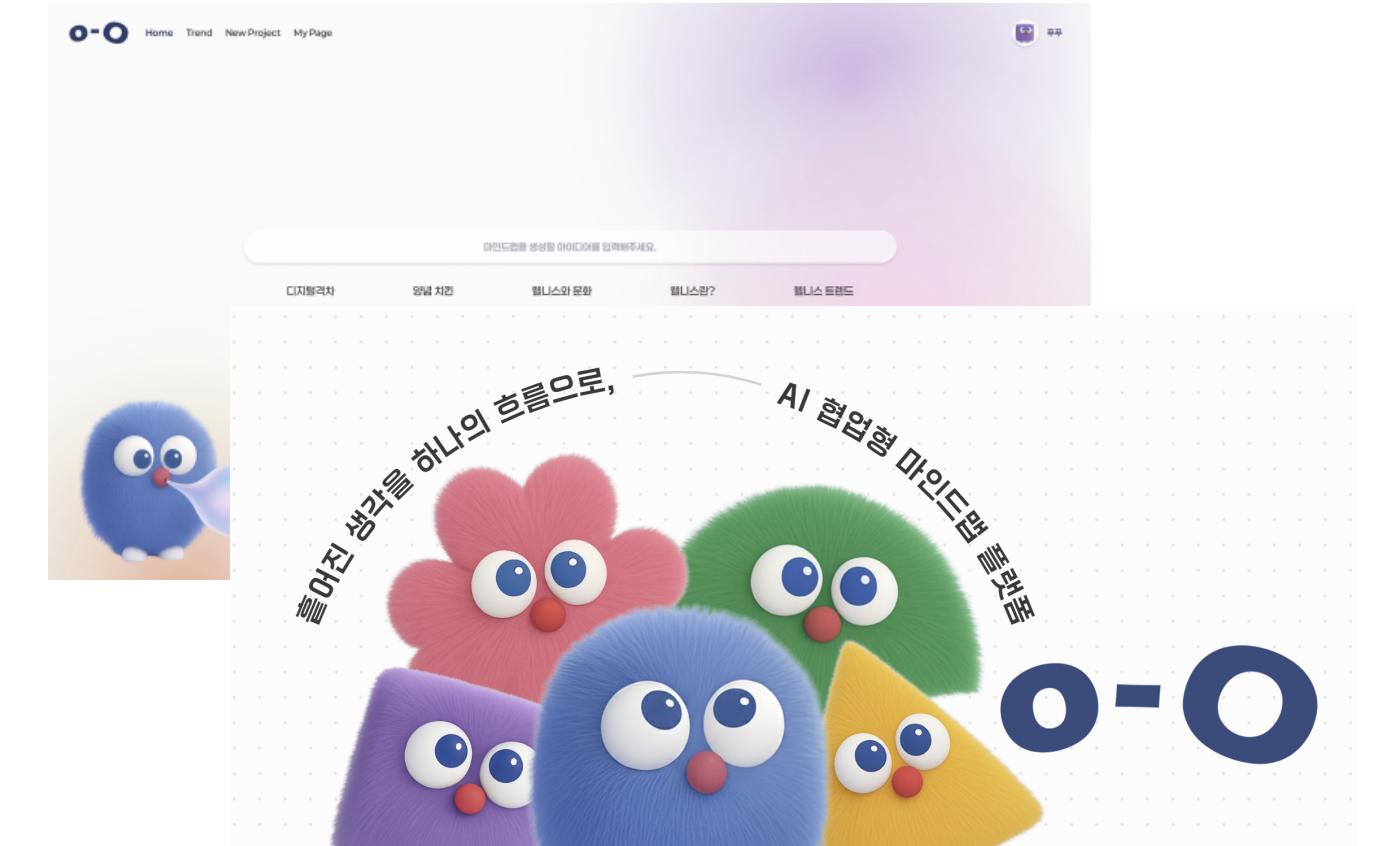
Broaf

Android

EKS·Kafka·Yjs 기반 실시간 동기화와 AI 파이프라인 운영 구조 설계

실시간 협업을 위한 MSA 아키텍처 설계 및 DevOps 자동화 구축

기관	삼성 청년 SW AI 아카데미
기간	2025.10.10 ~ 2025.11.20
기여도	30% 인프라(EKS, CloudFront) 구축, DevOps(배포·모니터링) 총괄, 게이트웨이·트렌드·마이페이지 서버 개발



아이디어를 연결하는 새로운 방식

기획배경

기존 팀 회의가 예비군·면접·일정 충돌로 **인원 변동이 잦아** 내용이 말로 전달되거나 **가독성 낮은** 회의록으로 남아 **협업 효율 저하**

기존 협업 도구는 기획 흐름을 한눈에 **파악하기 어려움**, 아이디어 확장 **기능 부족**
AI가 '대신'이 아닌 '도구'로 작동하는 **기획 플랫폼 필요**

기존 마인드맵 툴의 요약 중심 구조, 협업 부재, 확장성 부족 **문제가 존재**
 흩어진 아이디어를 **자동 구조화**하고 **흐름을 한눈에 볼 수 있는** **플랫폼 필요**

AI 아이디어 협업 마인드맵 서비스 # o-O

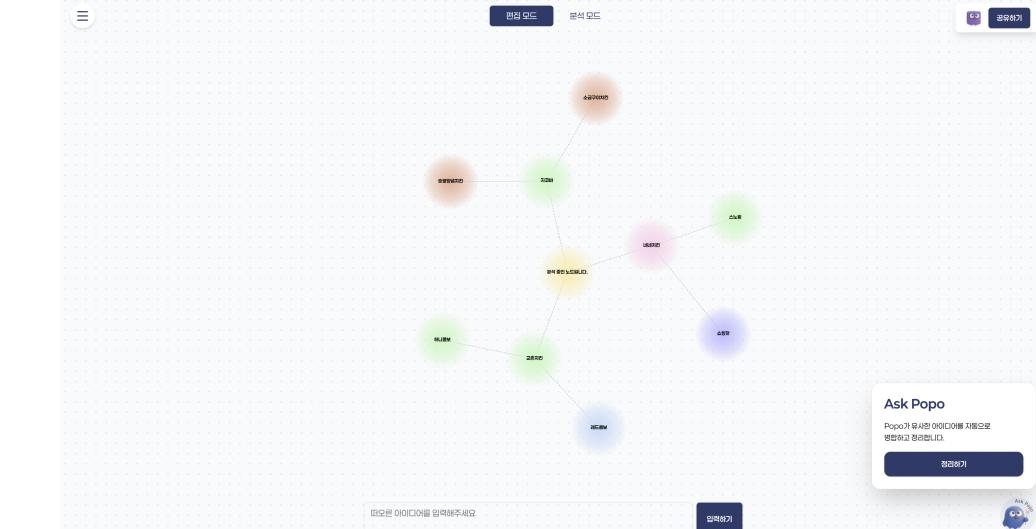
o-O는 아이디어를 구조화·확장하는 전 과정을 **AI**와 함께 수행하는 **협업형 마인드맵 플랫폼**
 말·자료·회의 대화 등 **다양한 입력을 자동으로 정리·연결·구조화**하여 시각적으로 표현
Yjs 기반 **CRDT**, **WebSocket**, **WebRTC** 음성 채팅, **Kafka** 기반 비동기 **AI** 처리, 멀티
 모달 **Llama** 모델 등 기술을 결합한 **실시간 협업 환경 구축**



트렌드

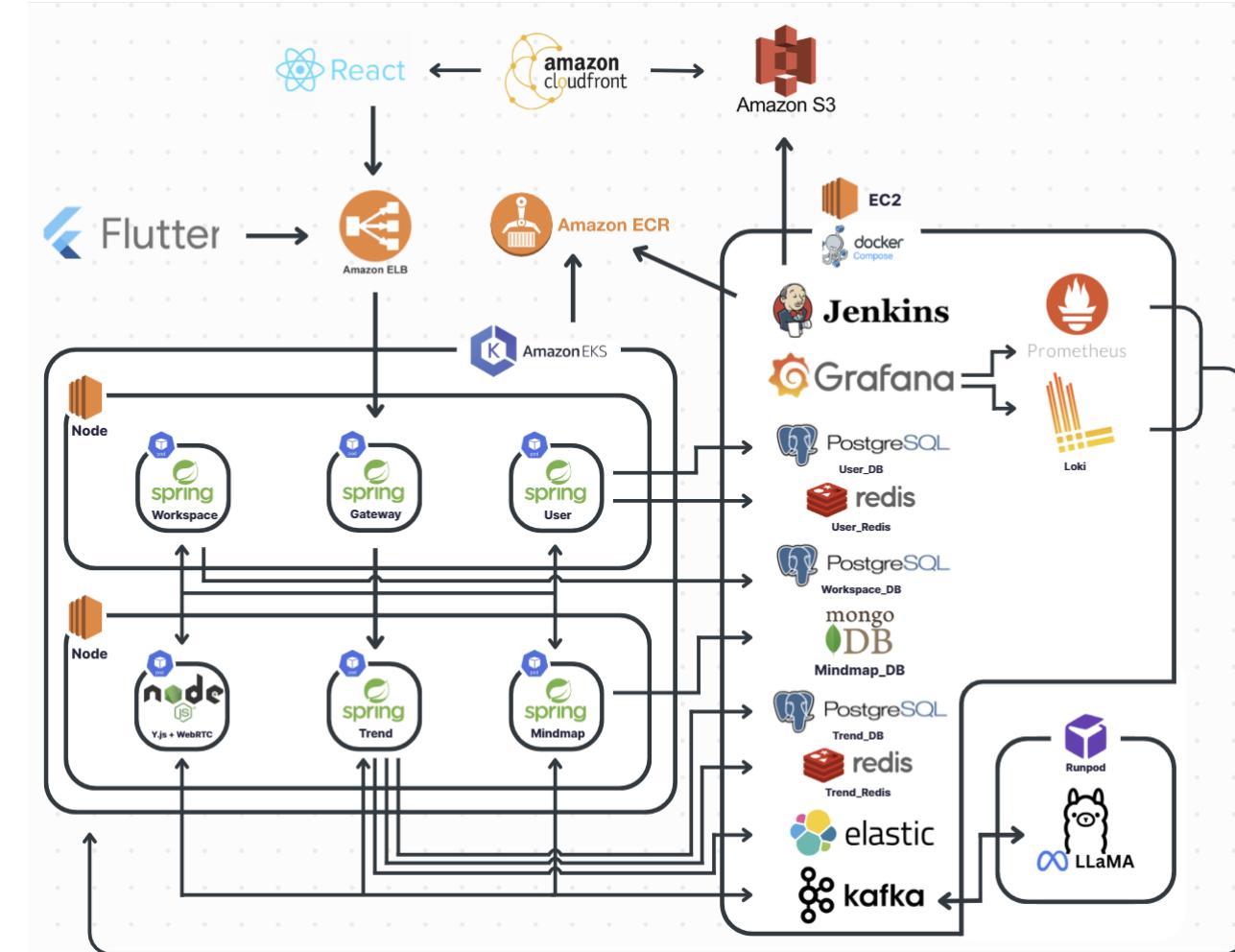


트렌드 노드



AI 정리하기

인프라·배포 총괄 및 핵심 백엔드 개발



CI/CD 구축 | 관측·모니터링 구성

Jenkins 기반 서비스별 개별 배포 파이프라인 구축

Docker·ECR·Helm 기반 자동 롤링 업데이트 운영

Prometheus + Grafana + Loki 기반 관측 환경 구성

인프라 구축 (EKS 기반 MSA 운영 환경 설계)

AWS EKS 기반 MSA 운영 환경 구성

서비스별 Namespace, Deployment, Service, HPA 구성

ACM 기반 HTTPS, ALB Ingress 라우팅 구성

S3 + CloudFront 기반 Web 정적 파일 배포 구성

Secret/ConfigMap 관리 및 운영 구조 설계

Backend Core 기능 개발

Gateway API 서버 전체 구현 (인증, 라우팅, 서비스 연결 포인트)

트렌드 서버 기능 개발

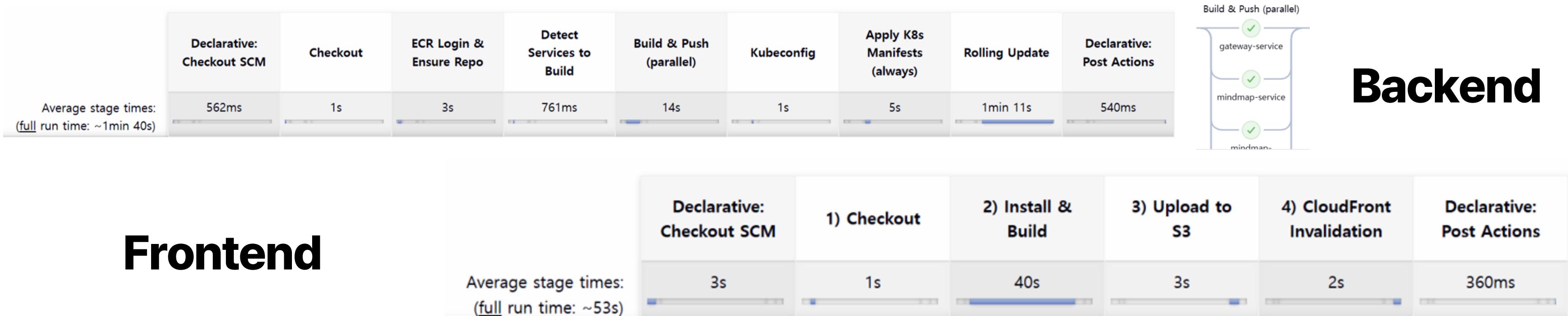
Redis ZSET 기반 실시간 트렌드 집계

Elasticsearch 기반 키워드 검색 및 인덱싱

マイ페이지·유저·워크스페이스 기능 개발 참여

Kafka 기반 비동기 메시지 처리 구조 설계

CI/CD 병목 해소 및 성능 개선 회고

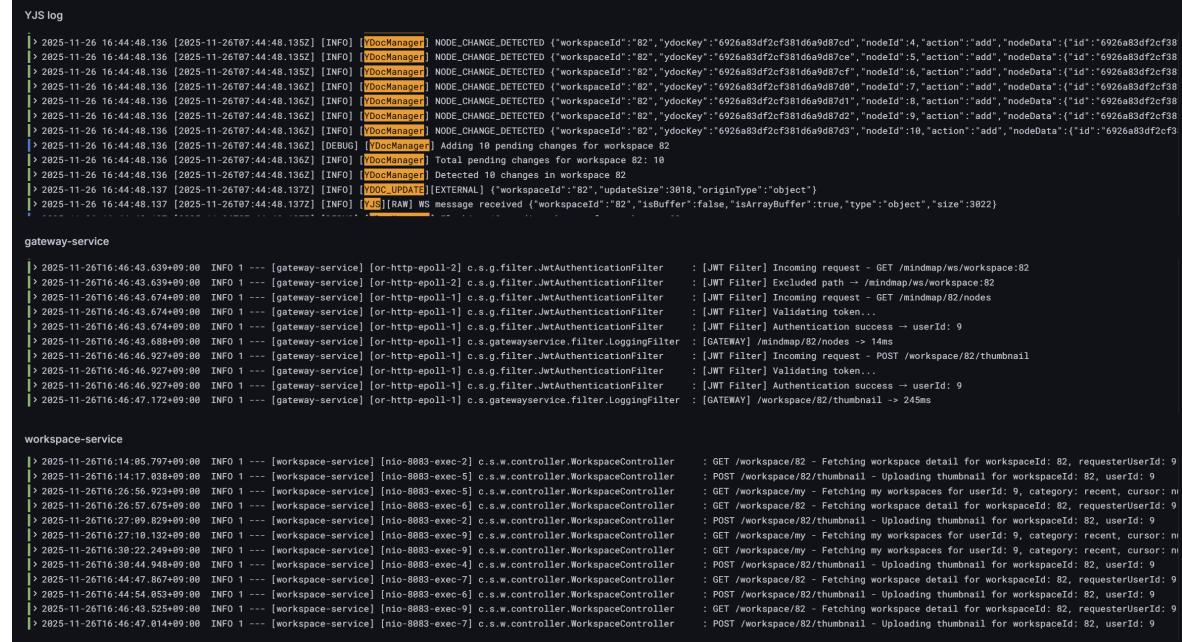


Frontend

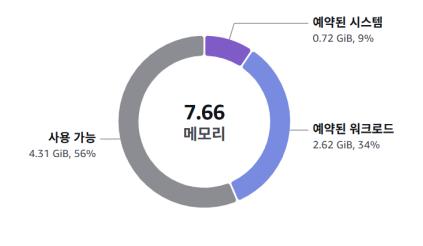
프로젝트 회고

이전 프로젝트에서는 여러 개의 백엔드 서비스를 순차적으로 빌드하여 전체 파이프라인 시간이 **평균 5분**, 빌드만 **약 2분 52초**가 소요되는 구조였다. 순차 빌드 방식은 서비스가 늘어날수록 배포 시간이 비례하여 증가하는 병목이었고, 운영 부담이 컸다. 이번 o-O 프로젝트에서는 **서비스별 병렬 빌드 구조(parallel stage)**를 도입하여 같은 시간 내 복수의 서비스를 동시에 처리할 수 있도록 개선했다. 그 결과 전체 파이프라인 평균 시간은 **약 1분 40초**, 빌드 시간은 **평균 14초**로 크게 단축되어 배포 효율성이 눈에 띄게 향상되었다. 향후에는 **롤링 업데이트 시간 자체를 줄이는 방향**(readiness probe 최적화, 이미지 사이즈 축소, Helm chart 개선, 카펜터 기반 빠른 노드 프로비저닝 등)으로도 확장해볼 수 있을 것으로 생각한다.

예산 제약 속 안정적 운영을 위한 인프라·스케일링 구조 개선 인사이트



NAME	READY	STATUS	RESTARTS	AGE
gateway-deploy-6f97678cb-knnvl	0/1	Pending	0	4m5s
mindmap-deploy-7b9778d876-hhxkv	0/1	Pending	0	4m5s
user-deploy-74b484d4d7-bh4nt	0/1	Pending	0	4m5s
workspace-deploy-867d9467f8-vzshv	0/1	Pending	0	4m5s



MSA 환경에서 서비스 개수만 늘리는 것보다, 노드 리소스·비용 한계를 먼저 고려한 인프라 설계의 중요성 체득

프로젝트 회고

t3a.large 1대를 기본 노드로 두고 몰릴 때 large 1대를 추가하는 구조 설계 → **CPU 여유 부족**으로 HPA가 Pod를 늘려도 전체 Pod가 **Pending**되는 **장애 경험**

서비스 수가 늘어나는 MSA에서는 서비스·인프라를 모두 EKS에 올리기보다, **EKS에는 애플리케이션만**, DB·관측성·Kafka 등은 **별도 EC2 + Docker Compose**로 분리하는 운영 구조가 더 현실적이라는 판단

로그·메트릭 확인을 위해 매번 AWS Console·kubectl에 의존하는 **구조의 한계**를 깨닫고, **Grafana**로 노드·서비스 상태를 한눈에 보는 관측 환경의 **필요성 체감**

개선점

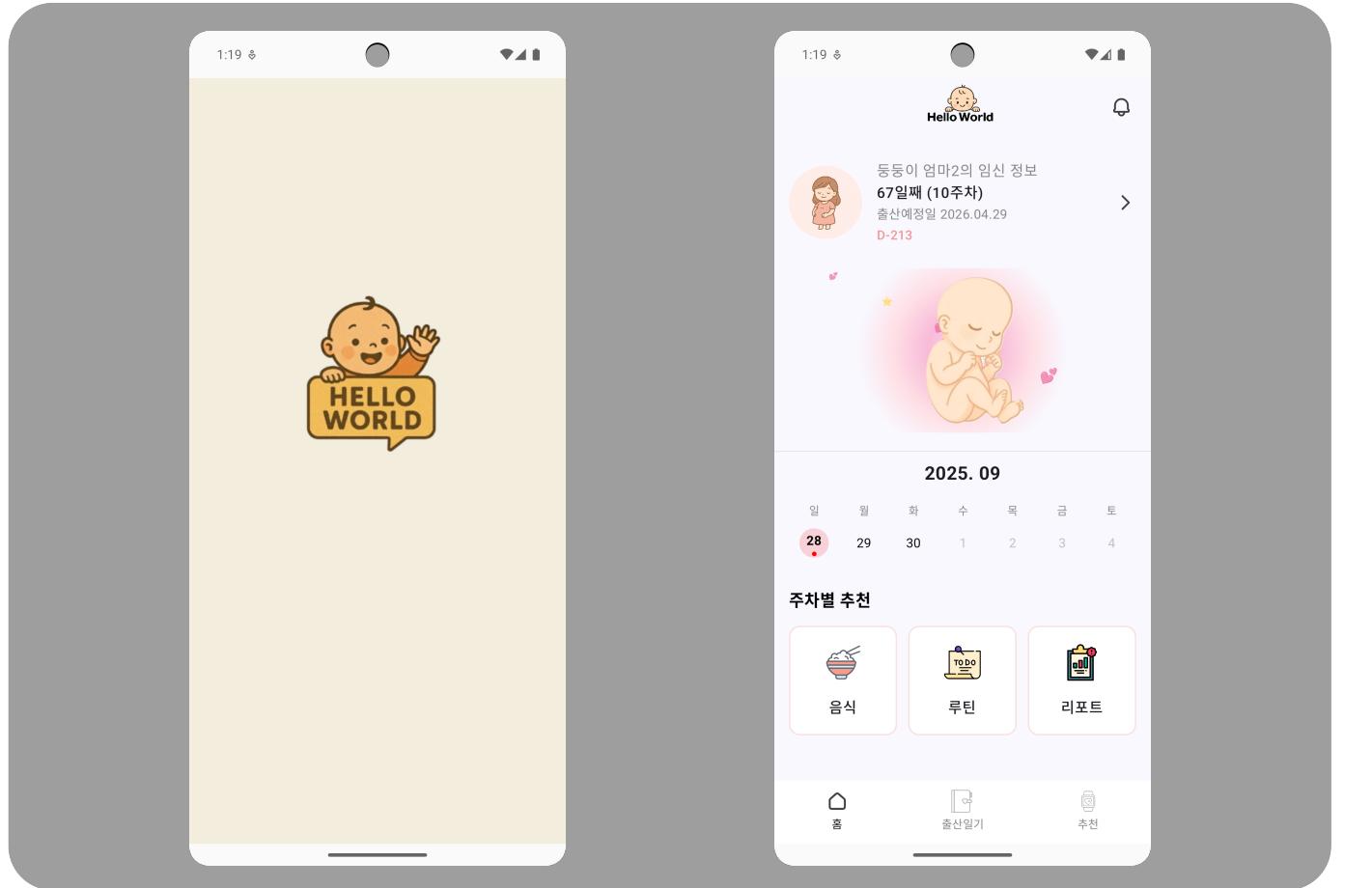
기본 노드 구성: t3a.large 1대 → **t3a.medium 2대로 분산**, CPU 사용률이 일정 기준 이상일 때 **Karpenter**가 medium 노드를 **1대 추가** 프로비저닝하도록 재설계 → Pending 없이 안정적인 자동 확장 달성

Grafana 대시보드에 노드·Pod CPU 사용량, 요청 수, 오류 로그를 통합해 배치 → MSA 환경에서도 **장애 구간을 한눈에 파악** 가능, AWS Console·명령어 기반 **로그 조회 시간 대폭 단축**

EKS에는 **애플리케이션 서비스만 배치하고**, DB·관측성 스택·Kafka 등은 **별도 EC2에서 Docker Compose로 운영** → 서비스 트래픽과 인프라 리소스를 분리해 **비용·안정성 모두 고려한 구조로 개선**

향후 과제: CPU 기반 HPA에 더해 요청 수·응답 시간 기반 스케일링(HPA/KEDA 조합), Spot 노드 활용 등 **비용 최적화까지 포함한 자동 확장 전략으로 고도화**

DevOps 자동화 기반 무중단 배포 체계 구축



기획배경

한국 산후우울증 유병률 **24%** 이상, 관리·지원 서비스 **부족**

실시간 모니터링 과 **예방 중심 관리** 필요성 **대두**

웨어러블·AI·모바일을 결합한 **맞춤형 케어 서비스**로 문제 해결을 **목표**

이전 프로젝트(NHCafe, BookgleBookgle)에서 **수동·중단 배포의 한계**를 경험 → **k3s** 기반

Rolling Update 구조로 **무중단 배포**를 완성

임산부 케어 & 빅데이터 추천 서비스 # HelloWorld

출산 이후 산모와 가족이 겪는 정신적 부담(산후우울증, 스트레스 등)을 완화하기 위해,
웨어러블 데이터 기반 맞춤형 케어 와 **가족 연동형 기록 서비스**를 제공하는 앱
빅데이터 기반 추천과 AI 연동으로 **개인화** 된 케어 경험을 지원

내 역할 삼성 청년 SW AI 아카데미 | 기간 : 25.08-25.09 | 기여도 : 20%

#Jenkins #Kaniko #k3s #Helm #Traefik #cert-manager #Prometheus #Grafana #Loki
#S3 #DALL-E-3

팀 구성: Android 2 · Backend 2 · Big Data 1 · **나(DevOps/Backend)**

CI/CD: Jenkins + Kaniko 자동 빌드/배포, 롤백

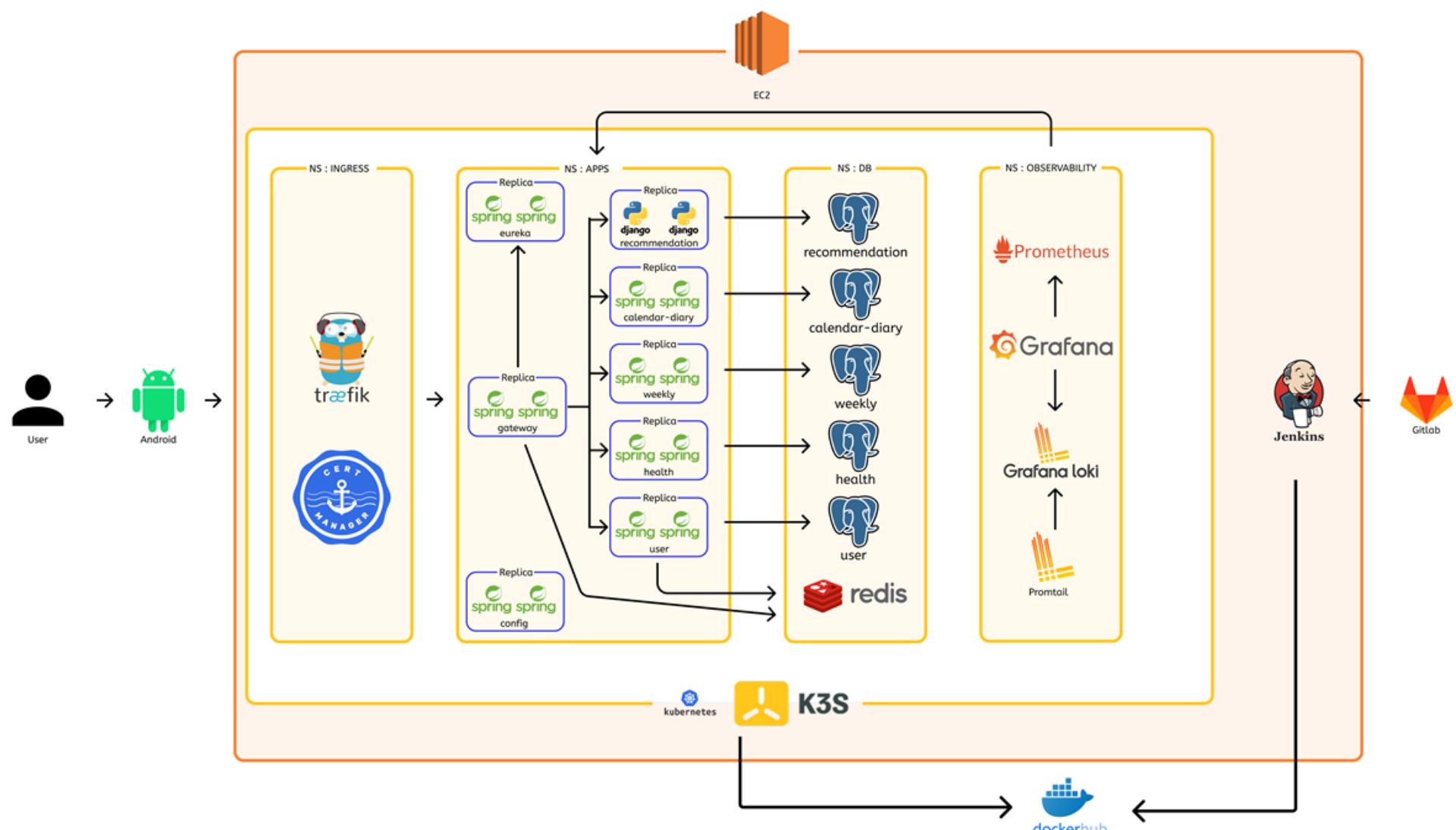
Cluster: k3s on EC2, Helm, Traefik + cert-manager

Observability: Prom / Grafana / Loki / Promtail

API: S3 Presigned URL, DALL-E-3 생성/저장

Impact: 배포시간 25%▽, 지연율 평균 10배 ▽

Declarative: Checkout SCM	Checkout	Kube debug	Detect changed (code/k8s/ingress)	Docker build & push (Kaniko)	Ensure namespace (optional)	Deploy to k3s (services)	Deploy to k3s (ingress)	Declarative: Post Actions	
Average stage times: (full run time: ~4min 52s)	16s	1s	1s	2s	2min 52s	2s	1min 19s	0ms	815ms



배포 파이프라인 한눈 요약

GitLab 푸시 → Ephemeral K8s Agent 생성

변경 감지(코드/매니페스트/Ingress) → 빌드·배포 타겟 최소화

Kaniko로 도커이미지 빌드/푸시 (<svc>-<gitSha>, <svc>-latest)

k3s(EC2)로 서비스 RollingUpdate + rollout status 확인

Ingress 변경 분만 별도 apply

결과는 GitLab Commit Status로 즉시 피드백

설계 이유

MSA: 서비스가 많아 → 변경된 서비스만 빌드/배포해야 시간·리스크↓

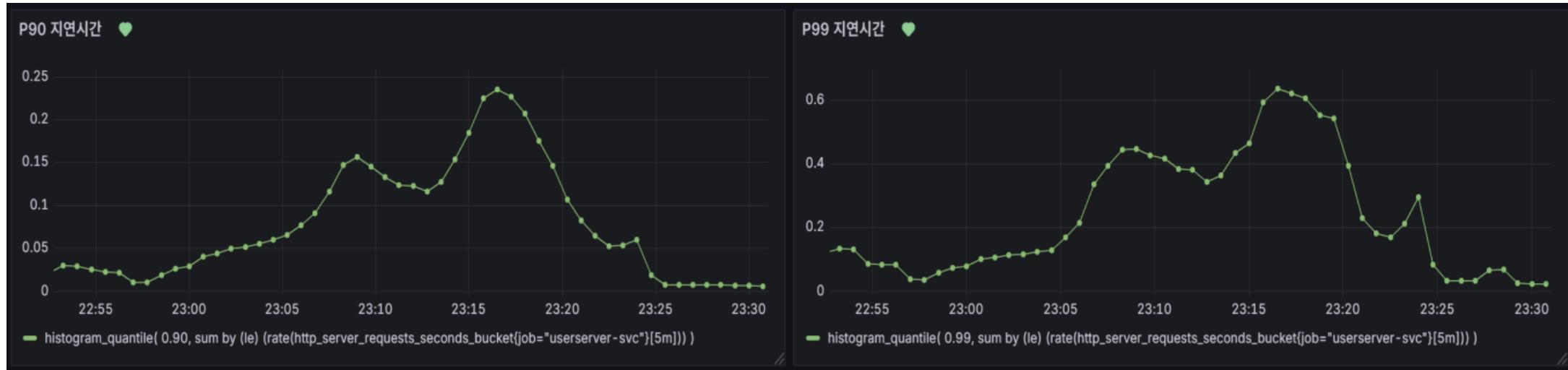
Kaniko: K8s 내부 daemonless 빌드(Docker 데몬 없이 안전/빠름)

k3s on EC2: 가벼운 K8s로 학습/운영 비용↓ + 표준 배포 경험 유지

Traefik + cert-manager: TLS 자동(Let's Encrypt)·헤더 정합 쉬움

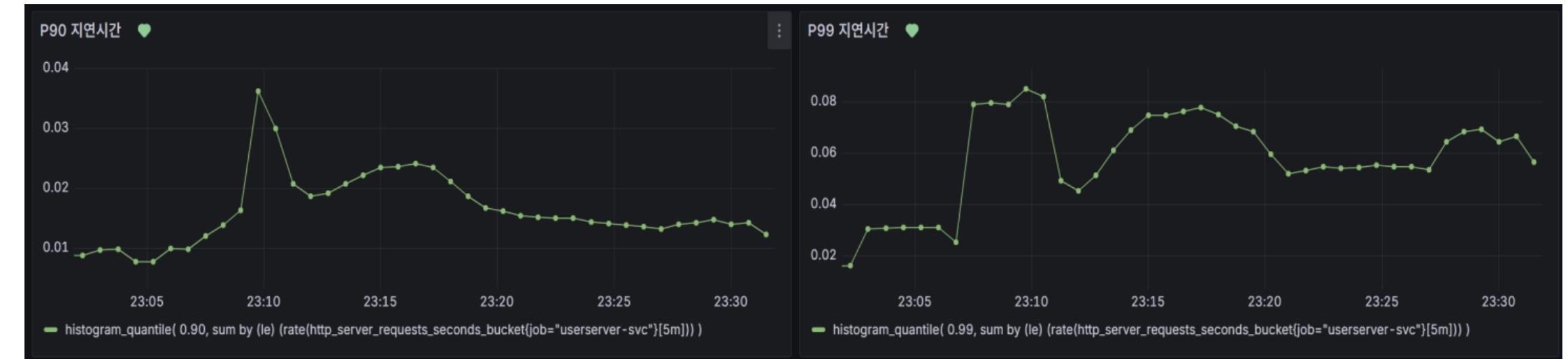
무중단/복원력: Readiness/Liveness, PDB/HPA/Anti-Affinity로 장애 전파 최소화

Observability: Prom/Grafana/Loki로 지표·로그를 하나의 대시보드에서 추적



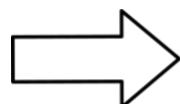
Before

After



지연율 단축 → 평균 10배 단축

배포 전략을 **replicas=3 → 2, maxSurge=1 / maxUnavailable=0**
로 조정해 롤링 중 메모리 압박 해소
스왑 활성화로 급격한 OOM을 완화, GC 스톰/스케줄링 지연 감소



P90: 피크 시 약 0.20~0.24s → 0.01~0.02s 수준으로 하락 (~ 10배 내외 개선)
P99: 피크 시 약 0.5~0.6s → 0.03~0.04s로 감소 (~ 12~15배 개선)
롤링 구간 이후 기저(베이스라인) 지연도 안정적으로 낮게 유지

OOM과의 이별: 메모리 2.5배 개선 & 롤링 실패 제거

Before

```
ubuntu@ip-...:~/yaml dir$ free -h
total        used        free      shared  buff/cache   available
Mem:       15Gi       14Gi     282Mi      99Mi       1.3Gi      1.1Gi
```

After

2.5배 △

```
ubuntu@ip-...:~/yaml dir$ free -h
total        used        free      shared  buff/cache   available
Mem:       15Gi       12Gi      1.6Gi      99Mi       1.7Gi      2.8Gi
Swap:      2.0Gi      1.3Gi    677Mi
```

Before

✓	Checkout SCM 15초
✓	Checkout 1.0초
✓	Kube debug 0.72초
✓	Detect changed (code/k8s/ingress) 1.5초
✓	Docker build & push (Kaniko) 2분 24초
✓	Ensure namespace (optional) 0.69초
✓	Deploy to k3s (services) 1분 18초
»	Deploy to k3s (ingress) 56ms
✓	Post Actions 0.12초

After

✓	Checkout SCM 16초
✓	Checkout 1.7초
✓	Kube debug 1.0초
✓	Detect changed (code/k8s/ingress) 2.3초
✓	Docker build & push (Kaniko) 2분 11초
✓	Ensure namespace (optional) 1.3초
✓	Deploy to k3s (services) 58초
»	Deploy to k3s (ingress) 56ms
✓	Post Actions 0.29초

25% ▽

트러블 슈팅

- **REPLICA 3 → 2로 조정:** 롤링 중 동시 유지해야 하는 파드가 많아 메모리 압박 → 노드(EC2) 멥통 사례 발생.
- REPLICAS=2, MAXSURGE=1 / MAXUNAVAILABLE=0로 바꿔 무중단은 유지하면서 메모리 여유 확보.
- **스왑 활성화:** 예기치 않은 피크 시 OOM 대신 완만한 성능 저하로 방어.
- **결과:** 사용 가능 메모리 **1.1 GiB → 2.8 GiB**, 롤링 시간(SERVICES) **1M 18S → 58S.**

생산성 증가 액션

- **변경 감지 기반 배포:** 코드/매니페스트/INGRESS 단위로 바뀐 서비스만 빌드·롤링 → 불필요 작업 제거.
- **KANIKO IN-CLUSTER 빌드:** 빌드 노드 관리 없이 일관된 속도/보안 확보.
- **결과:** 만약 적용하지 않았다면 항상 모든 서버를 재빌드하기 때문에 서버 하나 당 평균 2M의 시간이 증가되어 작업 효율이 낮아짐

핵심 회고

- **가용성은 복제 수만으로 보장되지 않는다.** REPLICA·ROLLING 전략·노드 자원이 함께 설계되어야 안전하다.
- **기본이 성능이다.** READINESS/LIVENESS, PDB/HPA, 리소스 한계선을 먼저 정리해야 배포 자동화의 이득이 제대로 나온다.
- **MSA는 선택과 집중이 성능.** 변경 감지로 작게/빨리 배포하는 게 전체 체감 속도를 크게 올린다.
- **병렬 빌드/롤링 도입(제한형):** 서비스 N개 동시 처리(예: 2~3개) + 동시성 가드(큐/세마포어)로 빌드 시간 추가 단축.

수동 배포의 한계를 넘어, 자동화로 성장한 DevOps 여정

실시간 PDF 협업 플랫폼

BookgleBookgle

삼성 청년 SW AI 아카데미 | 기간 : 25.07-25.08 | 기여도 : 20%

비대면 스터디가 늘어나며 여러 명이 동시에 PDF를 보고 토론할 수 있는 협업 플랫폼의 수요가 증가했습니다.

이를 해결하기 위해 PDF 동기화·주석 공유·AI 토론 기능을 제공하는 실시간 협업 서비스를 기획하였습니다.

운영 문제 및
개선 필요성

이전 프로젝트에서 **EC2 수동 배포** 과정에서 환경 충돌과 설정 오류 찾음

이번 프로젝트에서는 **Jenkins + Docker Compose**로 자동 배포를

도입했지만, 빌드 중 일시적인 **다운타임**(3~5분)이 발생하며 **무중단 배**

포의 필요성을 느낍

- 자동 배포 구현으로 배포 속도 약 4배 향상

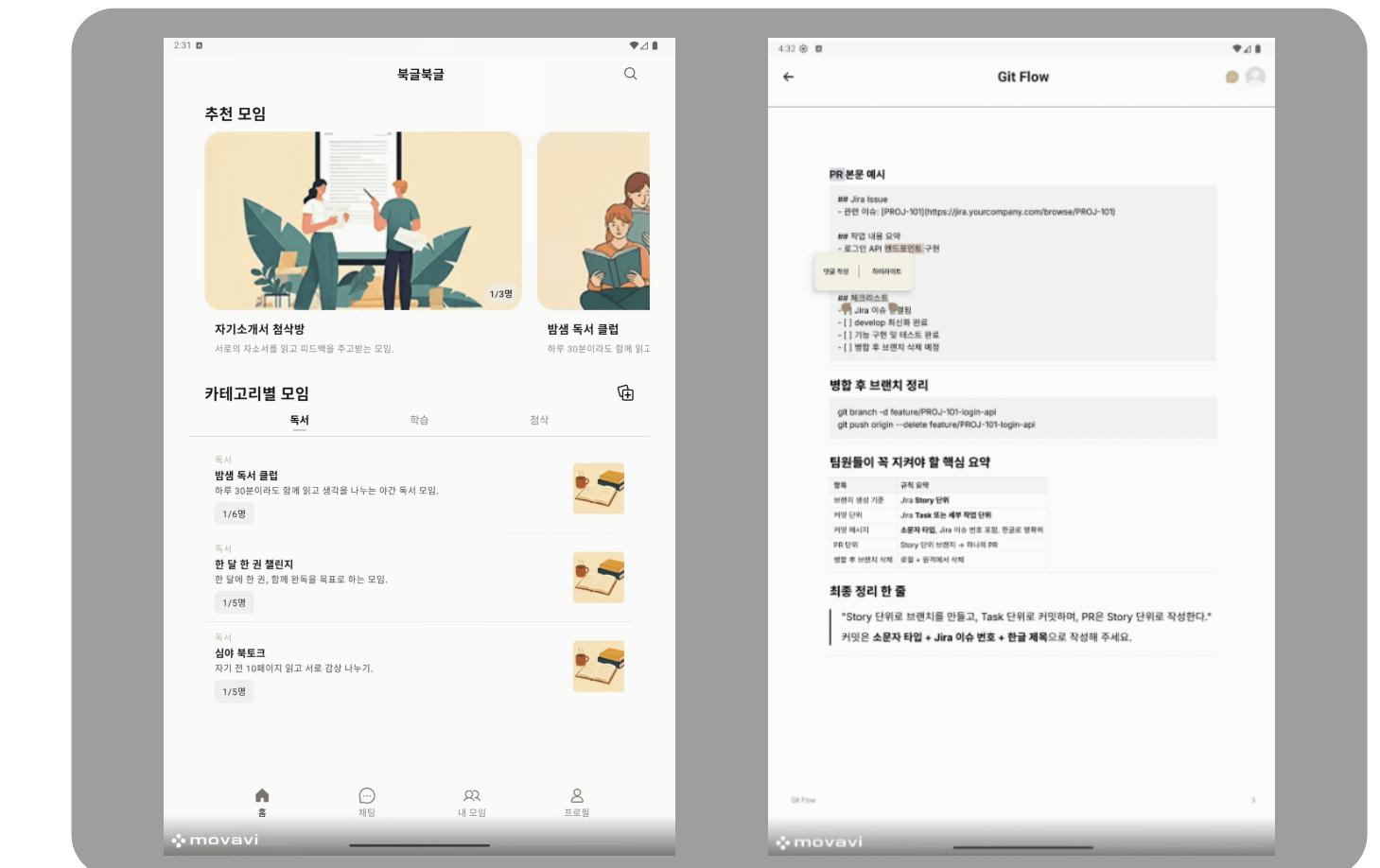
- 환경 차이 제거로 배포 실패율 대폭 감소

- Jenkins 도입하기 전 자동 배포의 필요성을 체감하고,

이후 무중단 배포 및 DevOps 환경 구축의 방향성을 잡음.

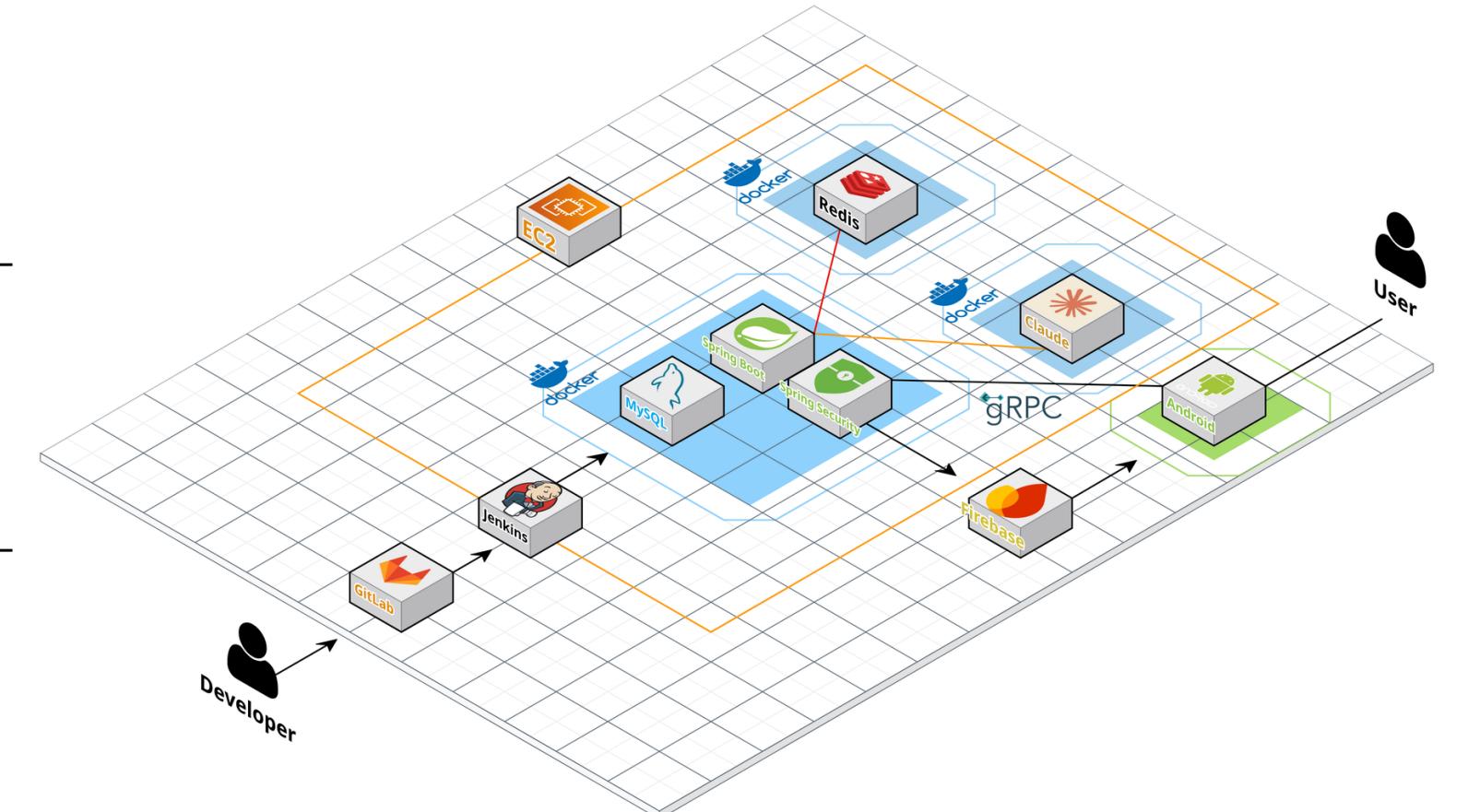
- gRPC 기반 서버 간 연동 구조 구현 및 통신 효율 향상

기술적 성과 요약



자동 배포 환경 구축 및 기술 설계

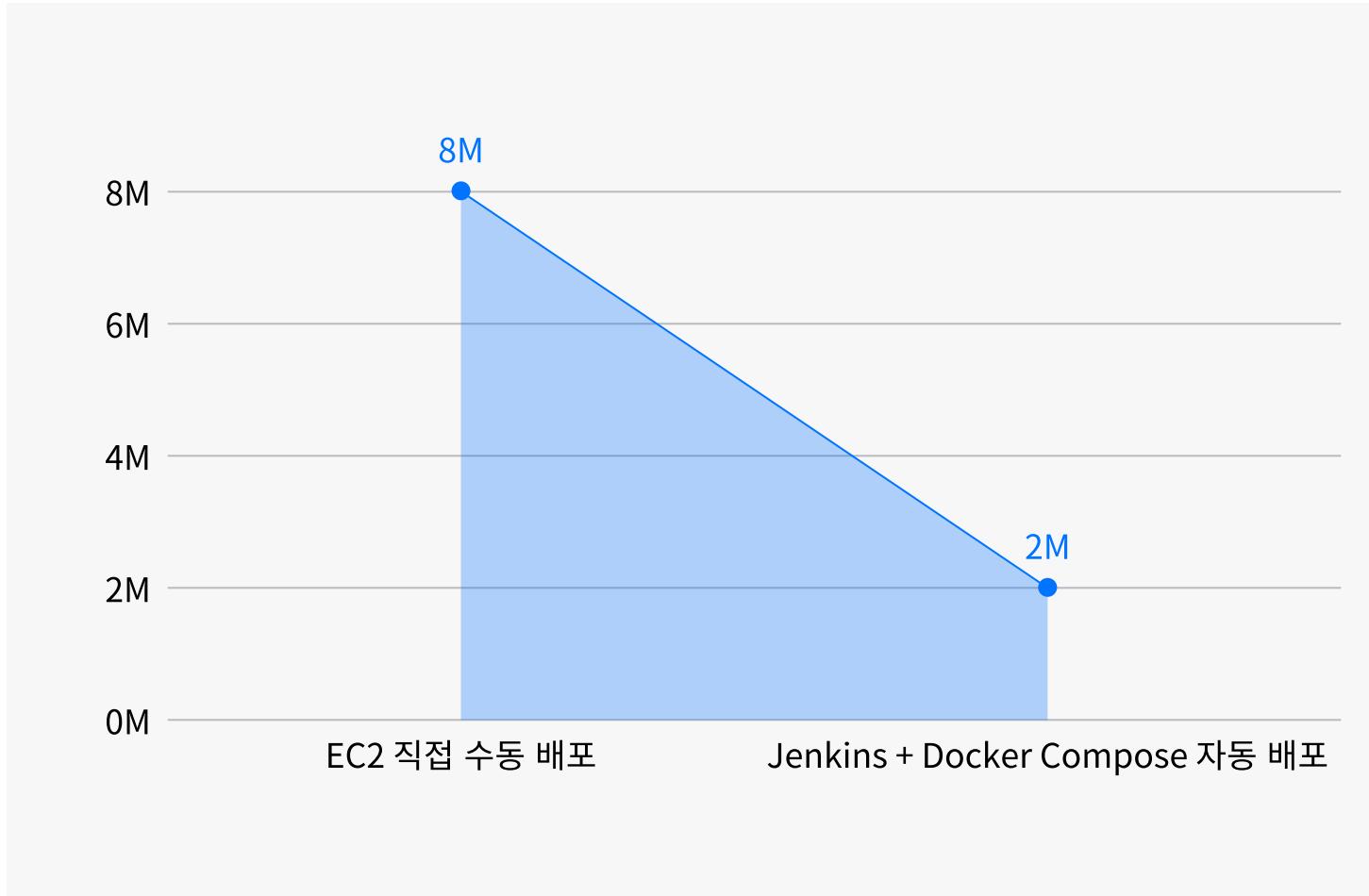
CI/CD 파이프라인	<ul style="list-style-type: none"> • GITLAB WEBHOOK으로 GITLAB 커밋 감지 • BUILD → PUSH → DEPLOY 자동화
DOCKER COMPOSE 배포 구조	<ul style="list-style-type: none"> • SPRING-APP, MYSQL, REDIS, PROMETHEUS, LOKI, GRAFANA 컨테이너로 통합 운영 • DOCKER COMPOSE로 환경 차이 제거 및 자동 배포 반복성 확보 • AI 서버는 별도 COMPOSE 환경에서 독립 운영, BGP-NETWORK로 gRPC 연동
한계 및 개선 포인트	<ul style="list-style-type: none"> • 빌드 중 컨테이너 재시작으로 일시적 다운타임 발생 • 다음 프로젝트(HELLOWORLD)에서 ROLLING UPDATE로 개선



프로젝트 결과 요약

Jenkins 기반 자동 배포 환경을 구축하여 **배포 효율성을 향상**시키고, **gRPC** 연동 구조를 통해 서비스 간 **통신 안정성을 확보**.
이 구조는 이후 프로젝트의 **무중단 배포(k3s)** 설계 **기반이 됨**.

이제 배포가 발목을 잡지 않습니다



문제해결

EC2 서버마다 직접 명령어를 입력해야 했던 **수동 배포** 방식을 **개선**.

생산성 증가

Jenkins 자동화 파이프라인을 구축하여 빌드–배포 과정을 **일원화**함.

의사소통 협상

Docker Compose 환경으로 서비스 간 **설정 차이**를 제거하고,
배포 소요 시간을 평균 8분 → 2분(**약 75% 단축**)으로 개선.

프로젝트 회고

EC2 수동 배포 시 명령어 입력과 설정 충돌로 반복적인 실패를 겪으며 **자동화의 필요성**을 느꼈습니다.

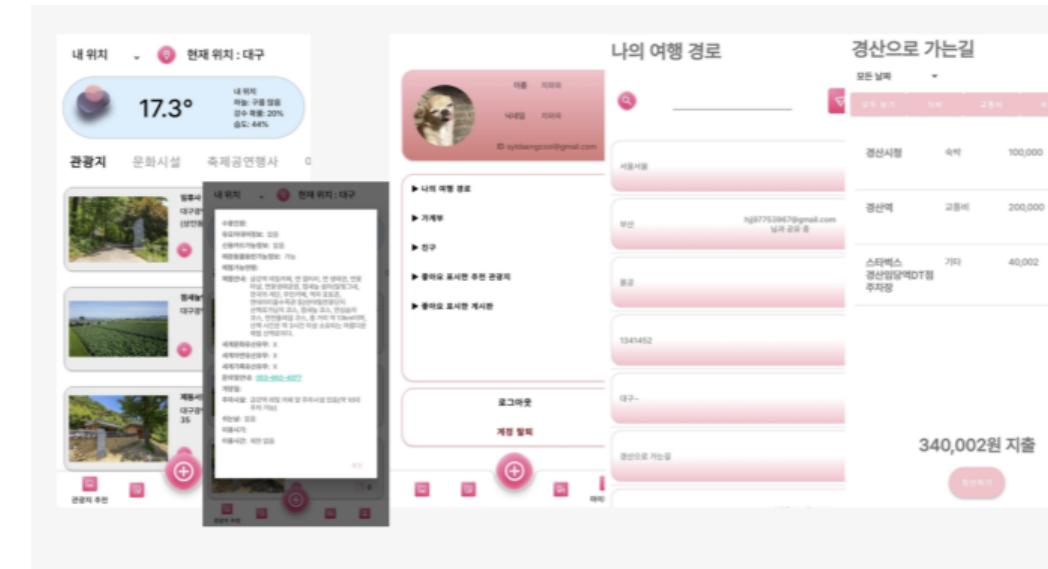
Jenkins 도입 후 배포 시간을 **8분 → 2분**으로 **단축**하고, 수동 작업의 번거로움을 해소했습니다.

다만 무중단 배포가 없어 푸시 시 서버가 재시작되는 문제를 겪으며 **Rolling Update의 필요성**을 인식했습니다.

또한 Android–Spring–AI 서버 간 통신을 **gRPC**로 통합해 **데이터 일관성과 전송 속도**를 개선했습니다.

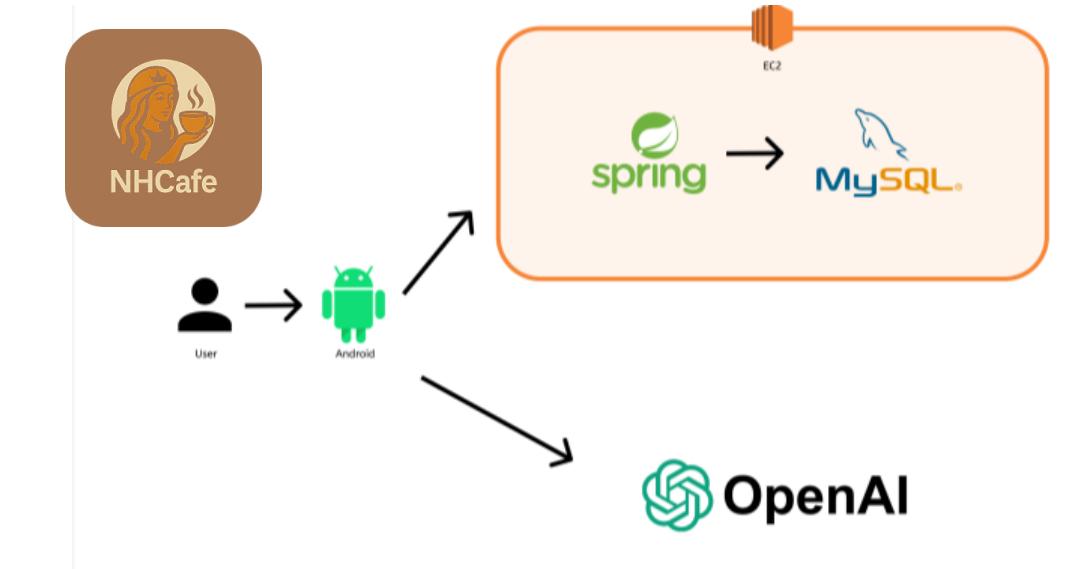
프로젝트 아카이브

수동 배포의 한계를 겪고, 실시간·지연·환경 불일치를 해결하는 과정을 통해 DevOps와 자동화의 가치를 체득했습니다.



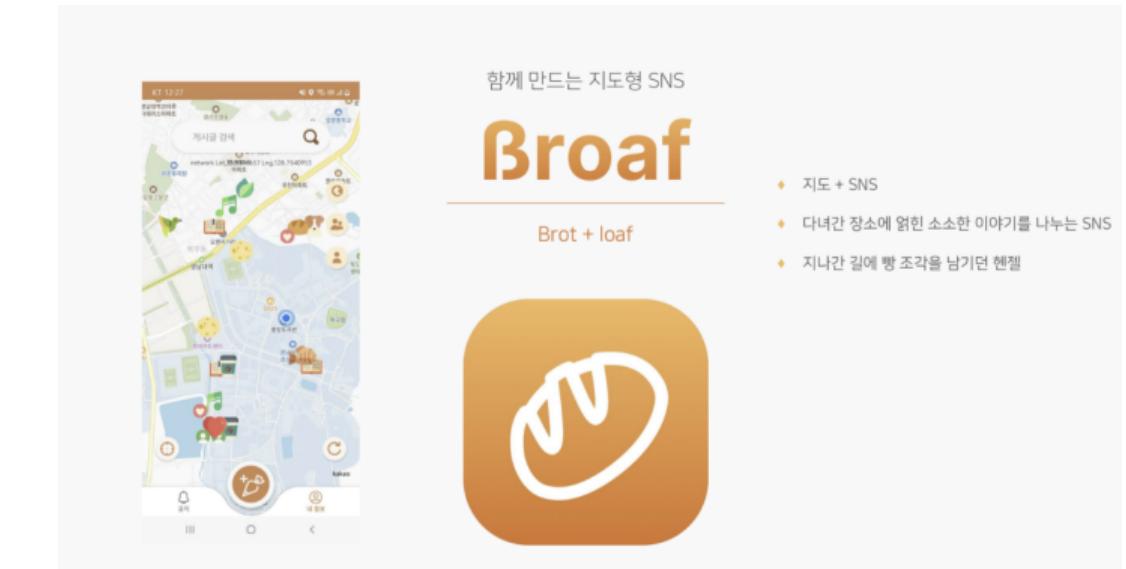
RouteMap

지도·리스트 상태 동기화, 초기 **로딩 지연 개선**(지연 로딩)
처음 화면에서 모든 정보들을 받아옴 → **화면 별로 분할**
다양한 API 연동 경험 → 외부 API 의존 문제 이해



NH Cafe

초기에 **수동 배포**(EC2 접속→재빌드→재시작)로
장애/설정 드리프트 발생
STT→GPT→TTS 대화 흐름은 OK
STT, TTS 모든 걸 GPT API 사용 시 **시간 △**
안드로이드 내 STT 사용. 추후 **엣지 컴퓨팅 필요성** 이해
배포/환경 표준화가 과제 → **자동화 필요 명확**



Broaf

KAKAO MAP API 검색/마커/경로선, UI **XML + RECYCLERVIEW 표준화**
팀원의 번아웃으로 일정 지연
우선순위 재정의, 팀원 케어 등으로 프로젝트 완수

THANK YOU

Song Jin-Woo

끌까지 검토해 주셔서 감사합니다.