□ 서식4_한이음 드림업 프로젝트 결과보고서

2025년 한이음 드림업
프로젝트 결과보고서

프로젝트명	도커 및 쿠버네티스 환경에서의 컨테이너 모니터링 시각화 대시보드 개발

요 약 본

프로젝트 정보	
프로젝트명	도커 및 쿠버네티스 환경에서의 컨테이너 모니터링 시각화 대시보드 개발
주제 영역	□ 생활 ■ 업무 □ 공공/교통 □ 금융/핀테크 □ 의료 □ 교육 □ 유통/쇼핑 □ 엔터테인먼트
기술 분야	■ SW·AI □ 방송·콘텐츠 □ 블록체인·융합 □ 디바이스 □ 차세대보안 □ 미래통신·전파
달성 성과	□ 논문게재 및 포스터 발표 □ 앱등록 ■ 프로그램등록 □ 특허 □ 기술이전 ■ 실용화 ■ 공모전(한이음 공모전) □ 기타( )
프로젝트 소개	본 프로젝트는 클라우드 네이티브 기술의 핵심인 도커및 쿠버네티스 환경의 운영 현황을 효과적으로 파악하기 위한 통합 모니터링을 제공할 수 있도록 컨테이너별 시스템 자원사용량을 수집 및 저장하는 서버 Application을 개발하는 것을 목표로 함. 최종 산출물은 데이터 수집 에이전트, 처리 서버, 데이터베이스, 그리고 시각화 대시보드로 구성된 완전한 3-Tier 애플리케이션임.
개발 배경 및 필요성	최근 IT 인프라가 가상화 컨테이너 환경으로 빠르게 전환됨에 따라, 수많은 컨테이너를 관리해야 하는 운영의 복잡성이 증가하고 있음. 특히 온프레미스와 클라우드가 혼재된 하이브리드 환경에서는 모니터링 시스템이 파편화 되어있어서 이용에 불편함이 따름. 따라서 파편되어있는 모니터링 환경을 통합하여 장애 대응시간을 단축하고, 운영비효율을 개선할 필요성이 있어 통합 대시보드을 제공하고자 함.
프로젝트 특·장점	1. 하이브리드 클라우드 통합 모니터링 제공 2. 업무시스템 별 확장성을 고려한 3 Tier 형식의 독립 Application 구조 3. 상용 통합 모니터링 도구 분야인 APM(Application Performance Management)의 영역을 다루고 있어, 프로젝트 고도화 시 상용화 발전 가능
주요 기능	1. 가상화 컨테이너 기술인 도커 및 쿠버네티스 구축 환경 2. 컨테이너 시스템별 컨테이너 자원사용량을 수집하는 Python Application 3. 서버 및 컨테이너 자원 사용량을 모니터링 및 관리 할 수 있는 기능 4. 구축된 모니터링 DB로부터 데이터를 조회하여 가시적인 모니터링을 수행할 수 있는 시각화 대시보드 기능
기대효과 및 활용 분야	1. 기업의 클라우드 전환 및 운영 효율성을 증대 2. 개발자는 복잡한 인프라 환경을 쉽게 파악 3. 컨테이너 기반 시스템을 운영하는 모든 기업 및 기관에서 실용적으로 활용 가능 4. 클라우드 인프라 구축 및 서비스 개발 관련 기업 취업 기대

(본문) 프로젝트 결과보고서

I. 프로젝트 개요

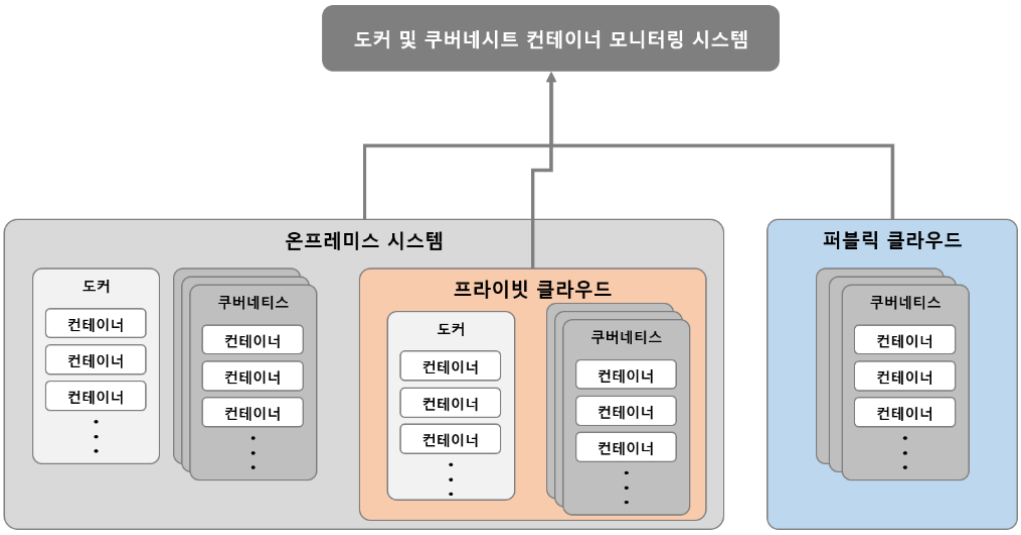
1. 프로젝트 소개

1) 기획의도

최근 기업 및 공공기관의 인프라가 가상머신VM 중심 구조에서 컨테이너 기반 클라우드 네이티브 환경으로 빠르게 전환되고 있다. 컨테이너는 배포 효율성과 확장성이 뛰어나지만, 컨테이너 수 자체가 많아지고 노드간 위치와 상태를 통합적으로 파악하기 어려운 운영상의 복잡성이 발생한다.

특히 기업들은 비용, 보안, 데이터 주권 등의 이유로 온프레미스, 프라이빗 클라우드, 퍼블릭 클라우드를 혼합하여 사용하는 하이브리드 환경을 구성하는 추세다. 그러나 이러한 환경에서는 컨테이너가 서로 다른 인프라에 분산되어 존재하기 때문에, 운영자는 각 환경별로 다른 모니터링 도구를 열어 봐야하고, 장애 발생 시 문제의 원인과 위치를 즉시 파악하기 어려운 구조적 한계에 직면한다.

이에 본 프로젝트는 이러한 분산형 운영 환경에서 컨테이너 단위 자원 사용량을 통합적으로 수집 및 관리할 수 있는 대시보드 시스템을 개발하고자 한다.
시스템은 도커 및 쿠버네티스 환경을 동시에 지원하며, 클러스터 내부의 노드와 컨테이너 메트릭을 통합 수집하여 단일 웹 화면에서 실시간으로 시각화함으로써 운영자가 효율적으로 상태를 파악하고 이상 징후를 조기에 감지할 수 있도록 설계되었다.



<도커 및 쿠버네티스 컨테이너 통합 모니터링 개념>

2) 개발목적

본 시스템은 하이브리드 클라우드 환경의 복잡한 인프라를 한눈에 관리할 수 있도록 데이터 수집, 데이터 처리, 데이터 시각화의 전 주기 통합 모니터링 구조를 구현하는 것을 목표로 한다.

구체적인 개발 목적은 다음과 같다.

1. 다중 환경의 자원 상태 통합 모니터링

- 도커 및 쿠버네티스 기반 컨테이너의 CPU, 메모리, 네트워크 사용량을 주기적으로 수집하고, 노드 단위로 통합하여 시각적으로 표현한다.
- 하이브리드 인프라 전반의 리소스 현황을 한 화면에서 파악할 수 있도록 클러스터 단위의 집계 구조를 구현하였다.

2. 실시간 시각화 기반의 효율적 운영 자원

- Chart.js를 이용해 컨테이너 자원 사용량 변화를 실시간으로 표시하고, 사용자 입력 없이 자동으로 갱신되는 비동기 방식으로 구성하였다.
- 이를 통해 운영자는 시스템 부하나 장애 가능성을 즉시 확인할 수 있으며, 문제 발생 시 빠른 의사결정이 가능하다.

3. 운영자 중심의 사용자 경험 개선

- 관리자 계정별 권한에 따른 접근 제어 기능을 제공하며, 자원 사용량 임계치 초과시 알림 및 이벤트 로그를 자동 생성한다.
- 직관적인 대시보드 UI를 통해 복잡한 클라우드 환경에서도 손쉽게 상태를 확인할 수 있도록 UX 중심으로 설계하였다.

3) 서비스 범위

본 프로젝트는 도커 및 쿠버네티스 기반 컨테이너 환경을 대상으로 하며, 다음 네가지 핵심 기능 영역으로 구성된다.

1. 컨테이너 및 노드 자원 수집 모듈

- 각 노드에서 도커 SDK와 쿠버네티스 API를 통해 컨테이너의 CPU, 메모리, 디스크, 네트워크 사용량을 주기적으로 수집한다.
- 수집된 데이터는 Flask/FastAPI 기반의 백엔드 서버로 전송되어 DB로 전송된다.

2. 실시간 시각화 대시보드

- Chart.js 및 Bootstrap을 활용해 CPU, 메모리 사용률, 노드 상태, 이벤트 발생 현황 등을 시각적으로 표현한다.
- 주요 그래프는 5초 주기로 자동 갱신되며, 운영자는 웹 브라우저에서 실시간 모니터링 가능하다.

3. 알림 및 이벤트 감지 기능

- 시스템 자원 사용률이 설정된 임계값을 초과할 경우 경고 알림을 발생시키며, 관리자 페이지에서 관련 이벤트를 로그 형태로 확인할 수 있다.
- 향후 Slack, 이메일 등 외부 알림 채널과의 연동이 가능하도록 확장성을 고려하여 설계하였다.

4. 관리자 페이지

- 사용자 등록, 수정, 삭제 등 계정 관리 기능을 제공하고, 권한에 따라 접근 가능한 페이지를 제한한다.

- 관리자는 전체 클러스터 상태를 통합적으로 조회할 수 있으며, 일반 사용자는 자신의 노드 및 컨테이너 정보만 확인할 수 있도록 분리하였다.

2. 개발 배경 및 필요성

1) 개발 배경

최근 컨테이너 기술의 표준으로 자리잡은 쿠버네티스는 서비스 확장과 배포를 자동화하는데 뛰어난 장점을 지닌다. 그러나 노드 및 컨테이너의 수가 수집, 수백 대로 증가함에 따라 자원 사용량 관리와 장애 대응이 점점 복잡해지는 문제가 대두되고 있다.

기업의 IT 인프라는 퍼블릭 클라우드, 프라이빗 클라우드, 온프레미스 서버를 혼합한 하이브리드 클라우드 형태로 발전하고 있으며, 이로 인해 모니터링 시스템의 단일화와 효율적 통합 관리의 필요성이 커지고 있다.

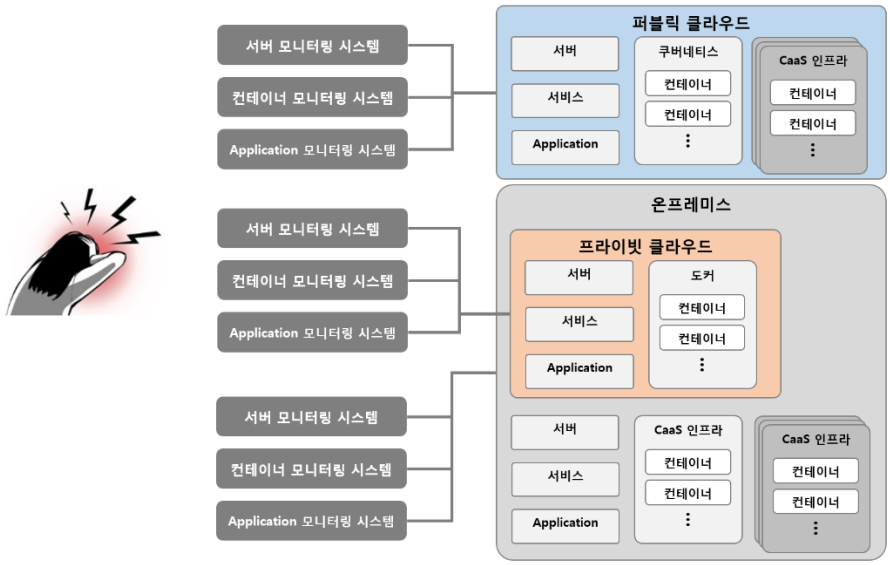
현재 업계에서는 상용화된 오픈소스 도구가 널리 사용되고 있지만, 설치 및 설정 과정이 복잡하고, 도커 단일 환경 또는 쿠버네티스 단일 클러스터에 국한되는 경우가 많으며, 사용자 맞춤형 알림 및 관리 기능이 제한적이다.

따라서 본 프로젝트는 다중 클러스터 및 다양한 운영 환경을 동시에 지원하면서, 운영 친화적 UI와 실시간 알림 기능을 제공하는 통합형 대시보드를 개발하여 이러한 한계를 보완하고자 한다.

2) 필요성

1. 하이브리드 인프라 운영자는 서로 다른 환경의 리소스 상태를 통합적으로 파악하기 위한 도구가 필요하다.
2. 컨테이너 기반 시스템의 자원 사용량은 시시각각 변하므로 실시간 모니터링을 통한 장애 조기 감지 및 대응 체계가 필수적이다.
3. 기존 모니터링 툴은 시각화 중심으로 구성되어 있어 관리자 권한 기반 접근제어 및 이벤트 로그 관리 기능이 부족하다.

따라서 본 시스템은 위와 같은 요구를 충족시키는 경량화된 통합 모니터링 솔루션을 구현함으로써, 클라우드 운영의 효율성을 향상시키고 운영비용 절감 및 장애 대응 속도 향상이라는 실질적 효과를 기대할 수 있다.



< 모니터링 시스템 다원화에 따른 운영자의 고충>

3. 프로젝트 특·장점

1) 기존 제품과의 차별성

현재 컨테이너 및 클러스터 모니터링 시장에서 널리 사용되는 대표적인 솔루션은 Promethesus와 Grafana 그리고 Kubernetes Dashboard이다.

항목	기존제품	본프로젝트
설치 및 구성 난이도	별도 서버 구성 및 설정 파일 필요하여 초기 설정이 복잡함	손쉬운 배포가능
데이터 수집 방식	간접 수집	직접 수집
시각화 구조	사용자별 맞춤 시각화에 제약 존재	동적 그래프 구성 페이지 단위 맞춤형 대시보드 구성
권한 및 접근 제어	단일 대시보드 중심 구조로 사용자 별 권한 제어 기능이 제한적	계정별 접근권한과 관리자 페이지 분리 구현
알림 시스템	복잡한 룰 파일 설정 필요	임계치 기반 실시간 감지 및 내부 이벤트 자동 로그 생성
배포 환경	각각의 컨테이너 이미지 필요	메모리 CPU 사용량 최소화

2) 기술적 독창성

1. 단일 프로세스 기반 통합 구조

기존 시스템은 데이터 수집기 , 시각화, 알림 등 여러 구성요소를 필요로 하지만 본 프로젝트는 Flask/FastAPI 기반 단일 애플리케이션 안에서 수집-저장-시각화-알림 기능을 모두 수행한다.

이를 통해 서비스 복잡도를 크게 낮추고 경량화된 운영 환경을 구현하였다.

2. API 중심 아키텍처 설계

각 기능은 RESTful API로 모듈화 되어있어 추후 별도의 프론트엔드나 외부서비스에서도 쉽게 연동이

가능하다. API 응답 포맷을 일원화하여 시스템 간 호환성을 강화하였다.

3. 실시간 반응형 대시보드 구현
Chart.js와 javascript fetch()을 이용하여 5초 단위 비동기 데이터 갱신을 구현하였다.
페이지 전체가 아닌 특정 그래프 영역만 부분 업데이트 되어, 렌더링 부하를 최소화하고 사용자 경험을 개선했다.

4. 관리자 중심 운영기능 내장
계정 CRUD, 접근제한, 로그조회가 가능하며 Flask 세션 기반 접근제어와 예외 페이지를 통해 운영보완성을 확보하였다.
이러한 통합형 관리자 기능은 오픈소스 솔루션에서 존재하지 않는 기능적 차별점이다.

5. 확장 가능한 알림 및 이벤트 구조
Threshold 기반 알림 감지 및 이벤트 로그 생성 기능을 분리 설계하였다.
향후 Slack, 이메일 등의 외부 연동이 용이하도록 확장성을 확보하였다.

6. 모듈 단위 의존성 최소화
모든 기능이 독립적인 파이썬 모듈로 구성되어 개별 기능 수정시 다른 컴포넌트에 영향을 주지 않는 낮은 결합도 구조를 유지한다. 이는 대규모 인프라로 확장될 때 유지보수 효율성을 극대화한다.

3) 독창성 및 활용 가능성

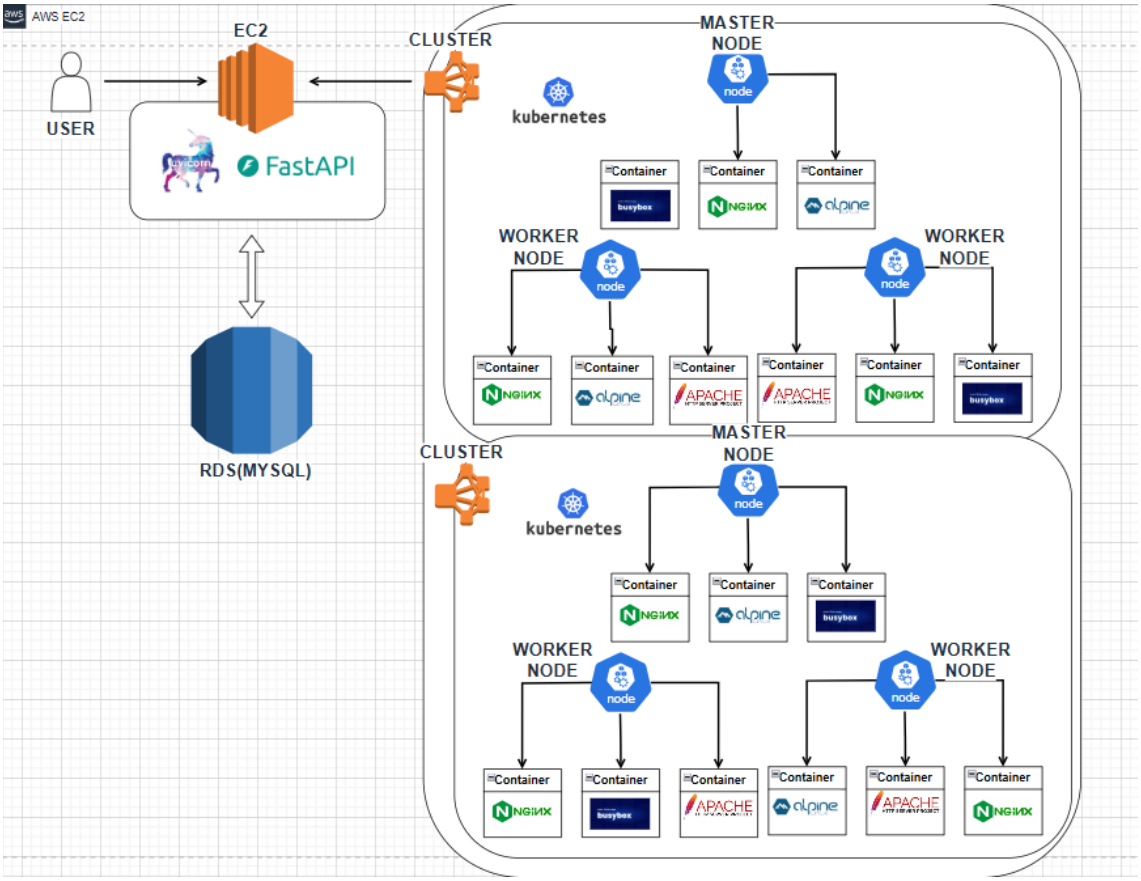
본 프로젝트는 단순한 대시보드 구현에 그치지 않고, 운영자의 실제 워크플로우를 반영한 경량화 통합 모니터링 시스템으로 개발되었다.
연구실, 스타트업, 학습 환경 등에서 별도의 모니터링 인프라 없이도 손쉽게 컨테이너 상태를 확인할 수 있는 실무적 가치를 가진다.

II. 프로젝트 내용

1. 프로젝트 구성도

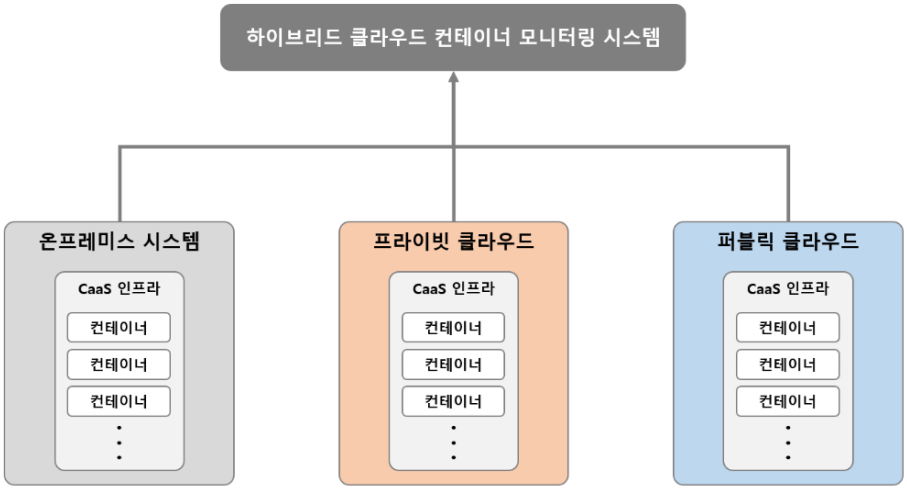
1) 시스템 아키텍쳐

- AWS EC2 (관리/모니터링 에이전트 프로그램)
- AWS EC2 (관리/모니터링 서버 프로그램)
- AWS RDS (MySQL) (모니터링 데이터 관리 DB)
- 모니터링 대상 시스템 (Kubernetes)



2) 하이브리드 클라우드 컨테이너 모니터링 시스템 구성도

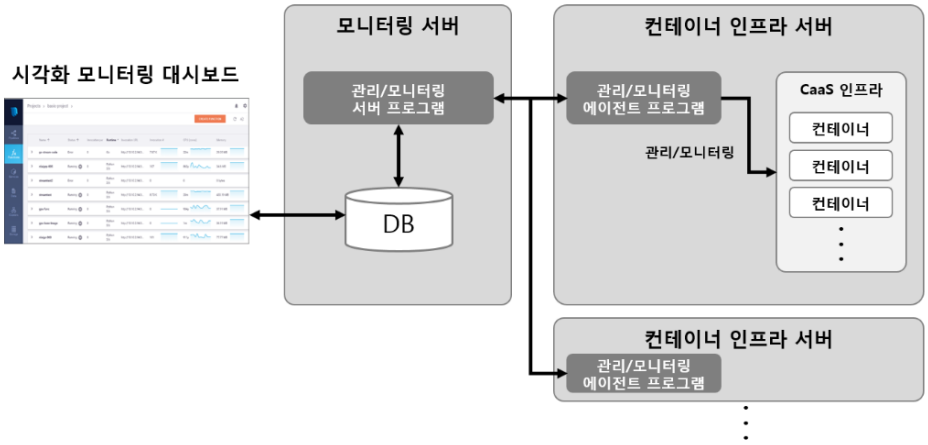
- 온프레미스 시스템 CaaS 인프라
- 프라이빗 클라우드 CaaS 인프라
- 퍼블릭 클라우드 CaaS 인프라



< 하이브리드 클라우드 컨테이너 통합 모니터링 개념 >

3) 모니터링 Application 구성도

- 관리/모니터링 에이전트 프로그램
- 관리/모니터링 서버 프로그램
- 모니터링 데이터 관리 DB



< 컨테이너 자원 모니터링 에이전트 및 서버 아키텍처 >

1) Fronted(대시보드 시각화)

- 개발 언어 → Vanilla JS(JavaScript)

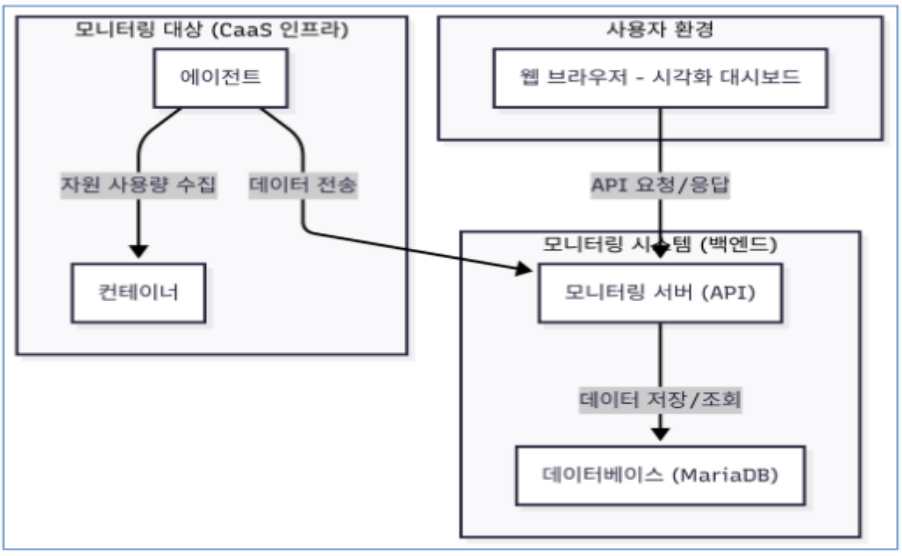
2) Backed

- 개발 언어 → Python
- 프레임 워크 → FastAPI
- 데이터베이스 → MYSQL(AWS RDS)

3) 데이터 수집 Agent

- 개발 언어 → Python
- 라이브러리 → kubernetes, prometheus

2) 서비스 구성도



< 모델 측면 서비스 구성도 >

- 시스템 사용량 시각화 대시보드

1. 각 노드의 Agent가 자원 사용량 수집 및 웹 서버로 데이터 전송
2. 서버에서 각 노드에게 전달받은 데이터를 DB에 저장
3. DB에 적재된 데이터를 바탕으로 사용량 시각화

2. 프로젝트 기능

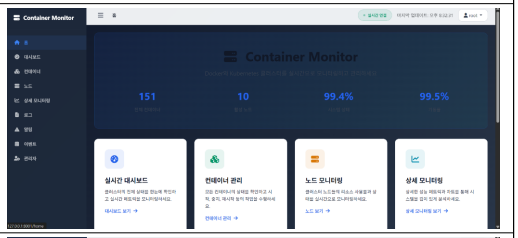
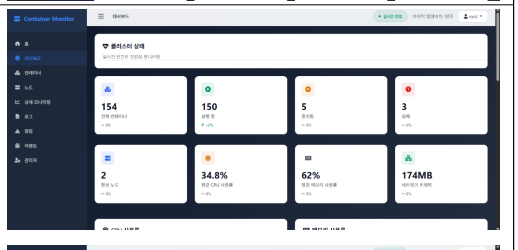
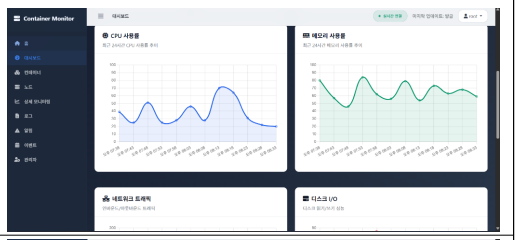
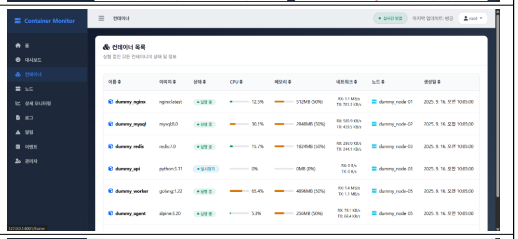
1) 전체 기능 목록

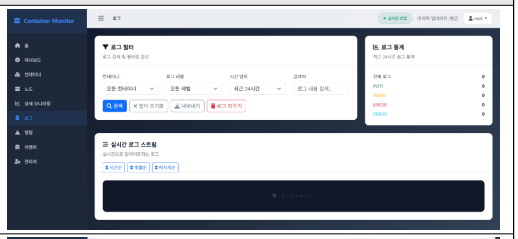
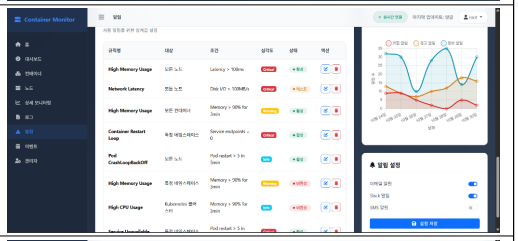
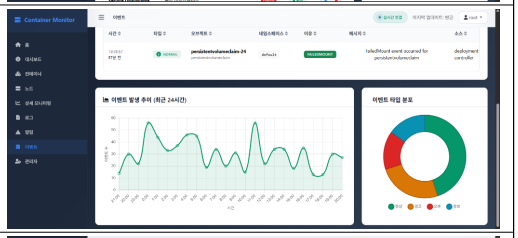
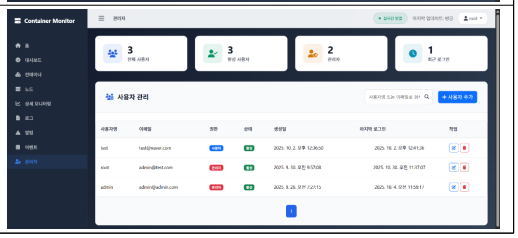
구분	기능	설명
S/W	사용자 인증 및 접근 제어	-사용자는 로그인 화면을 통해 이메일/비밀번호를 입력하여 인증하며, 성공 시 토큰이 발급되어 이후 모든 API 요청에 헤더로 첨부됩니다. 인증되지 않은 사용자가 관리자 페이지나 제한된 라우트에 접근할 경우, 접근 불가 팝업이 표시되며 5초 후 홈으로 자동 리다이렉트됩니다.
	클러스터 개요 대시보드	-대시보드 메인 페이지는 클러스터 전체의 상태를 한눈에 확인할 수 있도록 구성됩니다. 전체 컨테이너 수, 실행/중지/비정상 컨테이너 수, 활성 노드 수, 평균 CPU/메모리/디스크 사용률, 네트워크 트래픽 등의 핵심 지표를 카드 형태로 제공합니다. 실시간 API 데이터를 기반으로 자동 갱신됩니다.
	노드 목록 및 리소스 사용률	-'노드' 페이지에서는 쿠버네티스 클러스터에 속한 모든 노드의 정보를 테이블 형태로 표시합니다. 각 노드의 이름, 상태(Ready/NotReady/Warning), 역할(Master/Worker), CPU·메모리·디스크 사용률, 연결된 컨테이너 수, 업타임 등이 포함됩니다. 각 자원 사용률은 색상(초록/노랑/빨강)으로 구분되며, 직관적 시각화를 제공합니다.

구분	기능	설명
S/W	노드별 리소스 히트맵	-“리소스 사용률 히트맵” 섹션은 모든 노드의 실시간 상태를 카드 형태로 표시합니다. 각 노드는 CPU/메모리/디스크 사용률에 따라 색상으로 구분되며, 상태(Ready/NotReady)가 아이콘과 함께 표시됩니다. 주기적으로 데이터를 가져와 업데이트합니다.
	컨테이너 목록 및 상세 리소스 사용량	-컨테이너 페이지에서는 클러스터 내 실행 중인 모든 컨테이너 정보를 표시합니다. 컨테이너명, 이미지, 상태(running/stopped/paused), CPU/메모리/네트워크 I/O, 배포 노드, 생성일 등을 포함합니다. CPU 및 메모리 사용률은 진행바로 시각화됩니다.
	컨테이너 정렬 및 상태 표시	-모든 테이블 컬럼은 클릭으로 정렬이 가능하며, 정렬 방향은 아이콘(fa-sort-upfa-sort-down)으로 표시됩니다. 컨테이너 상태는 색상 뱃지(bg-success, bg-warning, bg-danger)로 시각적으로 구분되어 한눈에 파악할 수 있습니다.
	네트워크 트래픽 차트	-모니터링 페이지 내의 networkTrafficChart는 네트워크 트래픽 정보를 가져와 라인 차트를 표시합니다. X축은 시간, Y축은 MB/s 단위의 송·수신 트래픽이며, 실시간 변화를 시각적으로 모니터링할 수 있습니다
	디스크 I/O 차트	-diskIoChart는 노드별 디스크 읽기·쓰기 속도를 나타내며, Chart.js 라인 그래프 형태로 표시됩니다.
	응답시간 분석 차트	-responseTimeChart는 서비스별 평균 응답시간(ms)을 표시합니다. API 응답 데이터를 기반으로 각 서비스의 성능을 시각화하며, 서비스 병목 구간을 식별할 수 있습니다.
	요청 상태 분포 차트	-requestStatusChart는 HTTP 응답 상태(2xx, 4xx, 5xx 등)를 도넛 차트 형태로 표현합니다. 요청 성공률과 실패율을 시각적으로 비교할 수 있어, 장애 발생 시 즉시 식별이 가능합니다.
	상위 리소스 사용 컨테이너 목록 (TOP 10)	-“Top 리소스 사용 컨테이너” 섹션은 CPU 사용률이 높은 상위 10개의 컨테이너를 순위별로 표시합니다. 각 컨테이너의 CPU/메모리 사용률, 노드, 네트워크·디스크 I/O, 상태 등을 표시하며, 순위별 색상 배지(bg-danger, bg-warning, bg-info)로 구분됩니다.
	알림 통계 카드	-알림 페이지 상단의 메트릭 카드는 Critical, Warning, Info, Resolved 알림의 개수를 요약해 보여줍니다. 이전 대비 증감률도 함께 표시되어 최근 상태 변화를 빠르게 파악할 수 있습니다.
	알림 규칙 관리 및 편집	-알림 규칙 테이블에서는 자동 경고를 발생시키는 조건(CPU>85%, Memory>90%, Pod 재시작 등)을 조회·편집할 수 있습니다. “알림 규칙 편집” 모달에서는 규칙명, 대상, 조건, 심각도, 상태를 수정할 수 있습니다.
	로그 데이터 조회	-로그 페이지에서는 각 컨테이너의 로그를 시간순으로 조회할 수 있습니다. 로그 레벨(Info, Warning, Error) 필터와 키워드 검색 기능을 제공하여 원하는 로그만 추출할 수 있습니다.
	이벤트 목록 및 발생 통계	-이벤트 페이지에서는 쿠버네티스 클러스터의 이벤트(예: Pod Crash, Node NotReady 등)를 테이블로 표시합니다. 발생 시간, 네임스페이스, 타입, 메시지 등을 확인할 수 있으며, 최근 이벤트 발생량을 그래프로 시각화할 수 있습니다

	사용자 관리 (관리자 기능)	- 관리자 페이지에서는 관리자가 사용자 계정을 추가·수정·삭제할 수 있습니다. 계정의 권한, 활성 여부, 이메일 정보 등을 관리하며, 데이터는 백엔드 API와 연동됩니다.
	사용자 관리 (관리자 기능)	- 관리자 페이지에서는 관리자가 사용자 계정을 추가·수정·삭제할 수 있습니다. 계정의 권한, 활성 여부, 이메일 정보 등을 관리하며, 데이터는 백엔드 API와 연동됩니다.

2) S/W 주요 기능

기능	설명	프로젝트실물사진
로그인 페이지	사용자는 로그인 화면을 통해 이메일/비밀번호를 입력하여 인증하며, 성공 시 토큰이 발급되어 이후 모든 API 요청에 헤더로 첨부됩니다.	
홈	클러스터 전체의 상태를 한눈에 확인할 수 있도록 구성됩니다. 전체 컨테이너 수, 실행/중지/비정상 컨테이너 수, 활성 노드 수, 평균 CPU/메모리/디스크 사용률, 네트워크 트래픽 등의 핵심 지표를 카드 형태로 제공합니다.	
대시보드	클러스터 전체의 리소스 사용 추세를 시각적으로 보여주는 핵심 페이지 CPU, 메모리, 디스크, 네트워크 트래픽을 시간대별 라인 차트로 표시. 전체 컨테이너 수, 실행/중지/비정상 컨테이너 수 요약 카드 제공. 노드별 평균 자원 사용률, 네트워크 트래픽 합계 등 클러스터 집계 정보 표시.	 
컨테이너	컨테이너별 리소스 사용량 및 상태 페이지	
노드	쿠버네티스 클러스터 내 노드 목록 및 상태 페이지 CPU, 메모리, 디스크 사용률을 막대(progress-bar) 형태로 시각화. 노드별 연결된 컨테이너 수, 업타임 등 상세 정보 표시. 상태별 색상 구분	
상세 모니터링	실시간 클러스터 메트릭 분석 페이지 노드·컨테이너 단위의 실시간 메트릭을 종합적으로 표시합니다. - 네트워크 트래픽 (RX/TX) - 디스크 I/O (읽기/쓰기 속도) - 서비스 응답시간(ms) - HTTP 상태 코드 분포(2xx/4xx/5xx) - 노드별 리소스 히트맵(CPU·메모리·디스크) - 상위 리소스 사용 컨테이너 TOP10 테이블	 

기능	설명	프로젝트실물사진
로그	컨테이너 로그 조회 및 필터링 페이지 컨테이너에서 발생하는 로그를 실시간으로 조회하고, 로그 레벨(Info/Warning/Error) 또는 키워드 기반 검색으로 필터링할 수 있습니다.	
알림	시스템 경고 및 자동 알림 관리 페이지 Critical/Warning/Info/Resolved 상태의 알림 수를 카드로 요약하고, 활성 알림 목록을 테이블로 표시합니다.	
이벤트	쿠버네티스 시스템 이벤트 모니터링 페이지	
관리자	사용자 계정 및 권한 관리 페이지 관리자 전용 페이지로, 사용자 목록, 이메일, 권한, 활성 상태를 표시하고, 사용자 추가·수정·삭제(CRUD) 기능을 제공합니다.	

3) H/W 주요 기능

기능/부품	설명	프로젝트실물사진

프로젝트 실물사진을 반드시 첨부하여 실제 프로젝트 완성도 확인할 수 있도록 작성, 필요 시 줄 추가
H/W가 없는 경우 작성 안함(표는 유지)

3. 주요 적용 기술

1) 컨테이너 및 클러스터 관리 기술

본 시스템은 Docker와 Kubernetes를 기반으로 구성된 컨테이너 오케스트레이션 환경에서 동작한다. Docker는 애플리케이션을 개별 컨테이너 단위로 격리하여 실행하는 경량 가상화 기술로, 각 컨테이너의 CPU, 메모리, 디스크, 네트워크 리소스 등 모니터링을 가능하게 한다.

클러스터는 Master Node와 Worker Node로 구성되어 있으며, Master Node는 스케줄링, API 관리 및 클러스터 상태 제어를 담당하고, Worker Node는 실제 컨테이너 실행 및 리소스 사용 데이터를 제공한다. 또한, 각 노드에는 Kubelet이 리소스를 주기적으로 측정하여 데이터를 전달해준다.

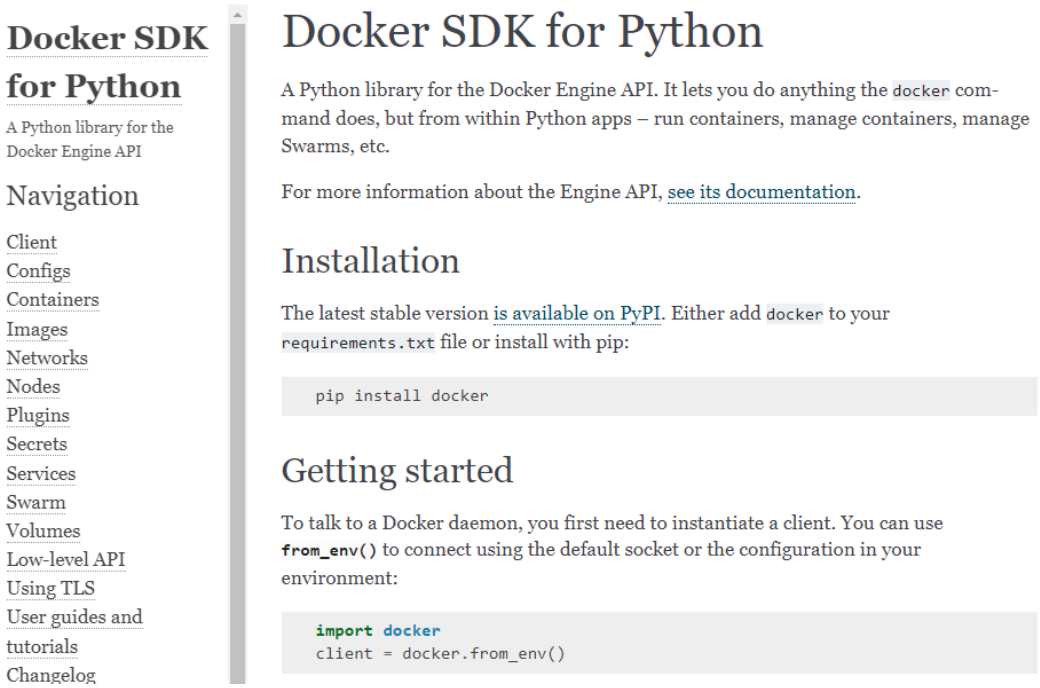
2) 데이터 수집 및 에이전트 구성 기술

각 노드에는 Python 기반 수집 에이전트(Agent)가 배포되어 있다. 이 에이전트는 psutil, docker, kubernetes 등 라이브러리를 이용해 시스템 및 컨테이너 자원을 실시간으로 수집한다. 수집된 메트릭은 Json 형식으로 FastApi 백엔드 서버로 전송되어 이를 DB에 적재하고, 적재된 DB 내용을 바탕으로 대시보드에 시각화한다.

3) 백엔드 API 및 데이터 관리

백엔드 서버는 FastApi 프레임워크를 기반으로 구현되었다. 다양한 Rest API를 통해 데이터를 프론트엔드에 제공한다. 서버는 각 노드에 있는 에이전트들에게 전달받은 데이터를 MYSQL DB에 추가적인 정보(시간 등)와 함께 적재하고, 불러오면서 대시보드에 실시간으로 데이터를 시각화 한다.

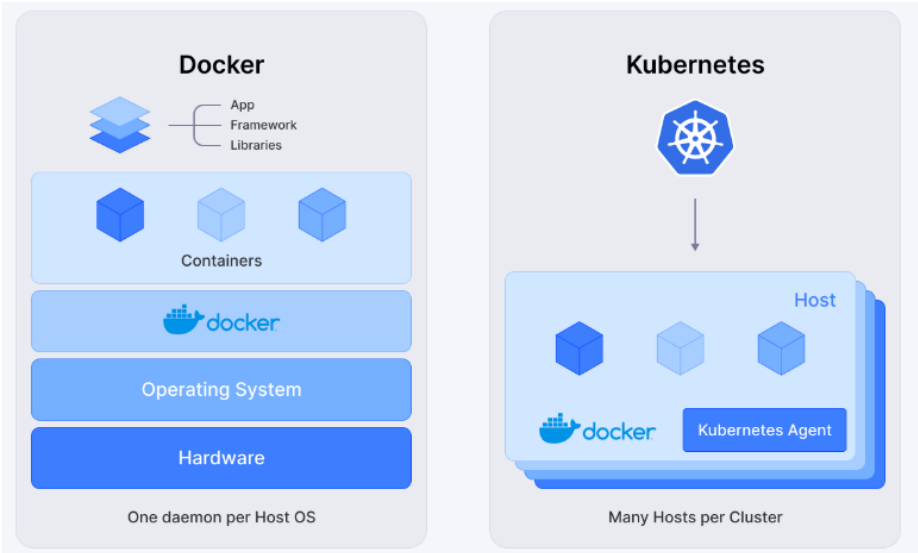
- Docker Container API 기술



< Docker Container Python SDK >

- Kubernetes 기술

  - Kubernetes 는 Container를 클러스터로 운영 및 오케스트레이션 기능을 제공하는 관리도구이며, 사실상 표준으로 사용되고 있는 기술임



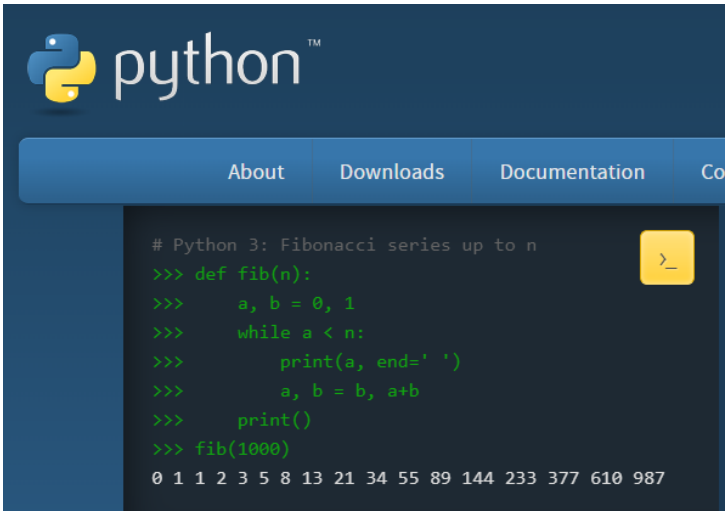
< 도커 및 쿠버네티스 시스템 환경 >

- Python Programming 기술

  - Python 은 스크립트 언어로, 초보자부터 전문가까지 용도에 따라 사용할 수 있는 프로그래밍 기술임.

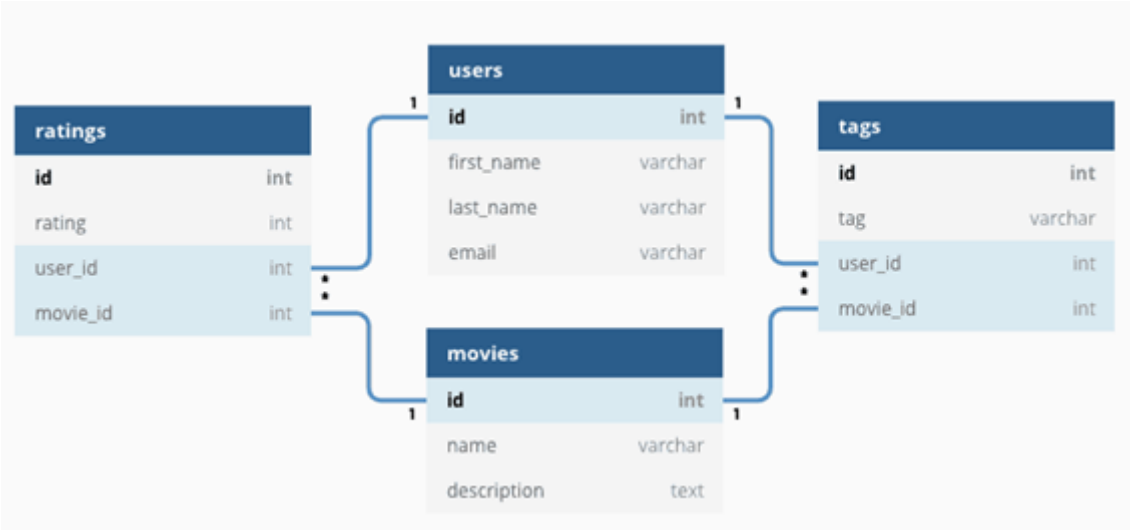
  - Docker 엔진에서는 Docker Container 자원 모니터링 처리를 위한 API를

제공하고 있음



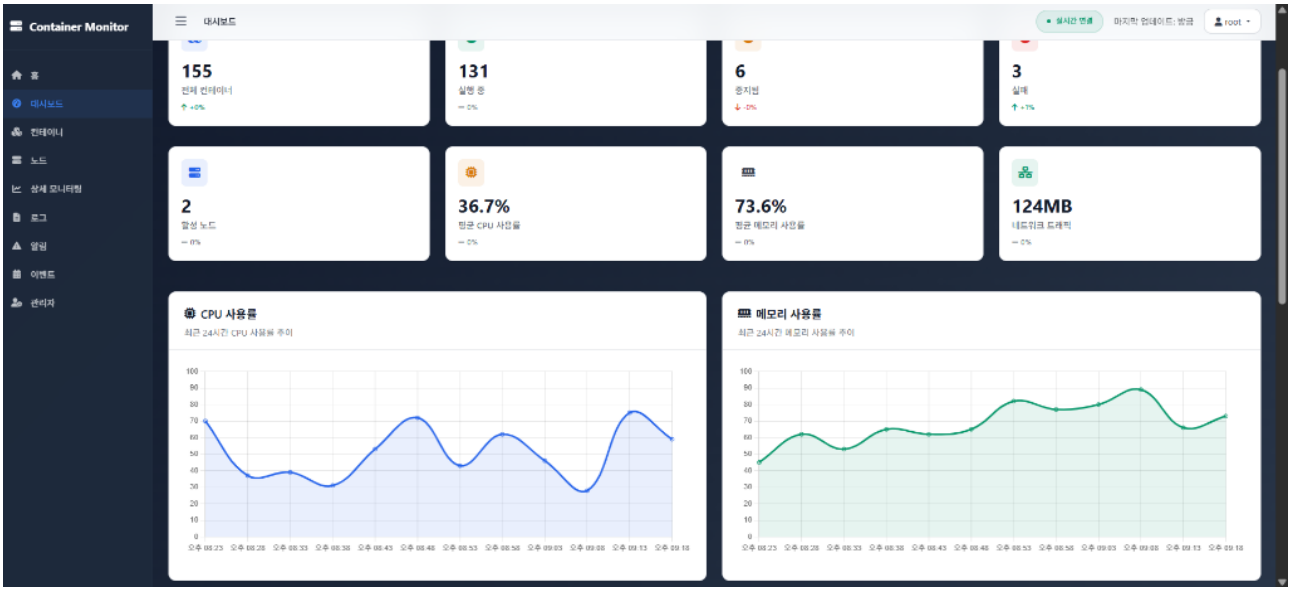
< Python >

- 데이터 저장을 위한 RDB CRUD 기술
(RDB는 관계형 모델을 사용하며, 키(key)와 값(value)들의 간단한 관계를 테이블화 시킨 매우 간단한 원칙의 전산정보 데이터베이스임)



< RDB 관계형 모델의 테이블 구조 >

4) 대시보드 시각화 기술



바닐라 JS를 기반으로 제작하였으며, Bootstrap, Font Awesome을 통해 기본 UI, 아이콘 세트를 사용하여 대시보드를 제작하였다.

4. 프로젝트 개발 환경

구분		상세내용
S/W 개발환경	OS	Windows, Ubuntu
	개발환경(IDE)	Visual Studio Code
	개발도구	Docker, Kubernetes, AWS
	개발언어	Python, HTML5, CSS3, JavaScripte, Flask
	기타사항	-
H/W 구성장비	디바이스	-
	센서	-
	통신	-
	언어	-
	기타사항	-
프로젝트 관리환경	형상관리	Github, Notion
	의사소통관리	Kakaotalk, Zoom
	기타사항	-

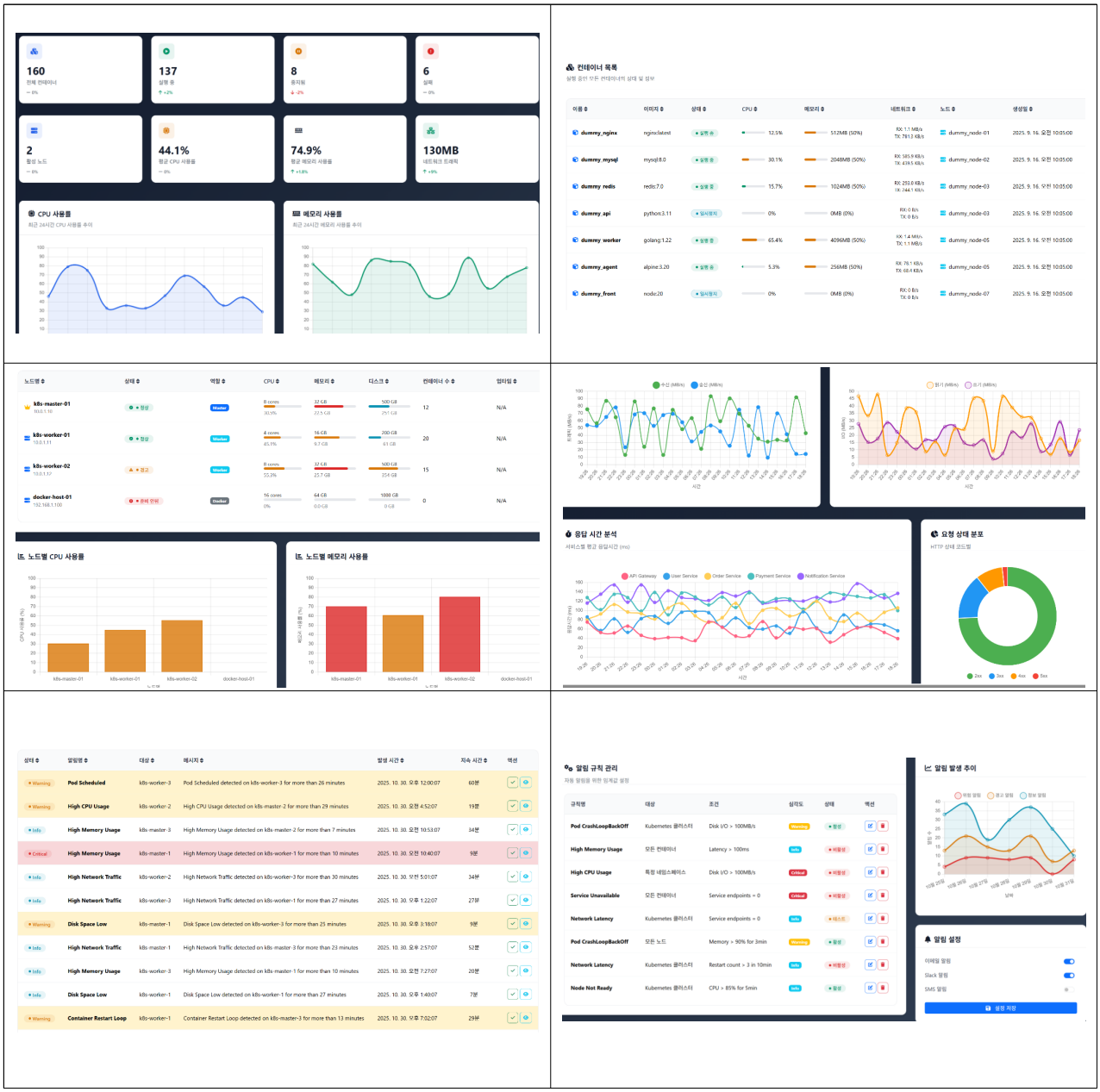
5. 장비(기자재/재료) 활용

번호	품명	작품에서의 주요기능
1	AWS 클라우드	S/W 설치 및 개발 환경으로 활용함
2	*라즈베리파이*	-
3		-
4		-

6. 프로젝트 작동 동영상

- https://youtu.be/j7B68e7mdZg

7. 결과물 상세 이미지


160
137
8
6
2
44.1%
74.9%
130MB
CPU 사용률
메모리 사용률
컨테이너 목록
노드별 CPU 사용률
노드별 메모리 사용률
응답 시간 분석
요청 상태 분포
알림 규칙 관리
알림 발생 추이
알림 설정

8. 달성 성과

□ 논문게재 및 포스터발표	게재(발표)자명	논문(포스터)명	게재(발표)처	게재(발표)일자
				2024. 00. 00.
□ 앱(APP) 등록	등록자명	앱(APP)명	등록처	등록일자
				2024. 00. 00.
□ 프로그램 등록	등록자명	프로그램명	등록처	등록일자
				2024. 00. 00.
□ 특허/실용신안 출원	출원자명	특허/실용신안명	출원번호	출원일자
				2024. 00. 00.
□ 기술이전	기술이전기업명	기술명	금액	이전일자
				2024. 00. 00.
■ 공모전	구분(교내/대외)	공모전명	수상여부(출품/수상)	상격
	대외	한이음 공모전	입선	
□ 실용화	*#실용화한 내용에 대한 구체적 설명 기재*			
□ 기타				

III. 프로젝트 수행 내용

1. 업무분장

번호	성명	역할	담당업무
1	멘토	멘 토	멘토링
2		지도교수	*# 지도교수가 없는 경우 공란으로 남김*
3	팀 장	팀 장	분석/설계, 클라우드 및 컨테이너 시스템 환경 구축
4	팀원 A	팀 원2	컨테이너 관리시스템 대시보드 웹 Front-end 개발
5	팀원 B	팀 원3	컨테이너 관리시스템 대시보드 웹 Back-end 및 DB연계 개발
6	팀원 C	팀 원4	컨테이너 시스템 자원 사용량 수집기 개발
7		팀 원5	

2. 프로젝트 수행일정

구분	추진내용	수행일정								
		3월	4월	5월	6월	7월	8월	9월	10월	11월
계획	*요구사항 수립 및 범위 산정*									
분석	*필요 기술 습득 및 분석*									
설계	*자원 사용량 수집 및 관리 API 설계*									
	모니터링 및 관리 에이전트 아키텍처 설계									
	데이터 처리/저장 서버 아키텍처 설계									
	시각화 대시보드 화면 설계									
개발	*도커 및 쿠버네티스 시스템 환경 구축*									
	모니터링 및 관리 에이전트 및 서버 개발									
	모니터링 데이터 처리/저장 서버 개발									
	시각화 웹 대시보드 연동 개발									
테스트	*단위/통합 테스트*									
종료	*산출물 정리 및 최종 보고*									
온·오프라인 미팅	*7회*									

3. 프로젝트 추진 과정에서의 문제점 및 해결방안

1) 프로젝트 관리 측면

문제점 :

1. 초기 단계에서 요구사항과 역할 분담의 세부 조율이 미흡하여 일부 작업이 지연되었다
2. 개발 및 테스트 환경이 EC2, 로컬, VM에 분산되면서 Docker 설정 및 컨테이너 위치 혼동이 발생했다.
3. 프로젝트 초반, 로컬 환경, AWS EC2, 로컬 VM 등 여러 환경에서 Docker/Kubernetes 설정을 병행하면서 각 노드 및 클러스터의 네트워크, 접근 권한, 방화벽 설정 등이 일관되지 않아 관리하는데 어려움이 있었다.
4. 프로젝트와 멘티들의 일정이 바쁜 시기가 겹칠 경우 일정 관리가 어려웠다.
5. 프로젝트 초기에 프론트엔드, 백엔드, 인프라 구성이 동시에 진행되어 개발 일정이 중첩되었고, Kubernetes 클러스터 환경이 완전히 구축되기 전에 API 연동 및 시각화 기능을 병행하면서 개발 효율이 떨어졌다.
6. 프론트엔드와 백엔드 간 인터페이스가 변경될 때마다 데이터 형식 불일치로 오류가 발생하였다.

또한 Git 브랜치 충돌 및 병합 시점 누락으로 코드 불일치가 발생했다.

해결방안 :
1. 일정 관리 툴(Notion / GitHub Projects)을 활용해 우선순위를 명확히 하고, 기능 단위로 이슈를 관리하여 개발 병목을 완화하였다.
2. 각 개발 환경별 Docker 설정 및 변수 파일을 문서화하여 혼선을 줄였으며 고정 포트와 공통 .env 구조를 동일하게 하여 배포 환경의 일관성을 확보하였다.
3. AWS EC2와 로컬 VM 간 네트워크 차이를 해결하기 위해 중앙 모니터링 서버를 구성하고, 고정 IP 할당 및 SSH 기반 접근을 적용해 안정적인 통합 테스트 환경을 마련했다.
4. 멘토의 정기적인 피드백과 관련 학습자룔르 기반으로 프로젝트를 유지했다.
5. 각 모듈(Backend / Frontend / Infra)을 독립적으로 로컬 Docker 환경에서 먼저 테스트 후 통합하는 방식으로 변경하였다.
6.Git Flow 전략(Feature / Develop / Main 브랜치)을 도입하고, 병합 전 PuII Request 리뷰를 의무화하였다.
7.OpenAPI(Swagger UI)를 FastAPI에 통합하여 API 명세를 자동 문서화하고,
프론트엔드 개발자가 항상 최신 스펙을 참고할 수 있도록 개선하였다.

2) 프로젝트 개발 측면

문제점 :
1. Python, Django, Docker API, Kubernates 등 사용 기술에 대한 이해가 부족하여 초기에는 개발 구조를 파악하고 기능을 구현하는 데 시간이 소요됨.
2. 컨테이너별 자원 사용량 계산 로직 구현시 단위 변환과 계산식 오류로 데이터 일관성이 낮았다.
3. Kubernetes API를 직접 호출할 경우 인증,토큰 문제로 인해 자원 사용량을 안정적으로 수집하기 어려웠다.
4. 클라우드 자원 사용량 제한으로 인한 제약사항을 고려해야했다.
5. 리소스 단위 불일치(MiB vs MB , mCPU vs %) 로 시각화 단계에서 혼란이 발생했다.
6. 도커와 쿠버네티스가 제공하는 네이티브 API는 응답 데이터 구조가 복잡하고 알아야할 세부사항이 많았고 직접 호출하고 관리하는 것으로 인해 코드의 복잡성이 크게 증가하였다.
7. 대시보드, 모니터링 페이지 등에서 CPU, 메모리 사용량 같은 데이터를 실시간에 가깝게 보여줘야했기 때문에 페이지를 주기적으로 새로고침하는 것은 서버에 부담을 주었다.
8. 인증, 관리자, 컨테이너, 노드, 로그 등 기능이 점차 추가되면서 프로젝트 코드 복잡성이 증가하였다.
9.Docker SDK와 psutil의 CPU 측정 방식이 달라, 컨테이너별 사용률 합계가 노드 전체 사용률과 일치하지 않았다.
10. 네트워크 트래픽 차트에서 선이 표시되지 않고 점만 출력되는 문제가 발생했다.
이는 API 응답에 borderColor가 누락되어 투명으로 렌더링된 것이 원인이었다.

11. DB에 저장된 알림이 해결되지 않은 상태에서 동일 조건이 다시 트리거되어 중복 알림이 다수 생성되는 현상이 발생하였다.

해결방안 :

1. Python 공식 문서 , Docker SDK 예제, Django 문서 등을 반복 학습하여 구조를 파악하고 기능을 구현할 수 있었다.

2. Docker stats() API 구조를 분석하여 CPU 사용률 계산 알고리즘을 개선하고, 데이터를 표준 포맷으로 구조화하여 서버를 연동하였다..

3. Rest API, Python SDK 등을 활용하여 환경 독립적인 구조로 설계하고 시각화 기능을 간소화해서 진행함. 다양한 환경에서의 설정 차이를 해결하기 위해 .env파일, 공통 디렉토리 구조 전략을 수립했다.

4. Kubernetes 환경에서는 kubernetes-client 라이브러리를 도입하여 인증 절차를 간소화하고, 데이터 수집의 정확성과 신뢰성을 확보했다.

5. .env 파일과 공통 디렉토리 구조를 통일하여 환경별 설정 불일치 문제를 해결했다.

6. 멘토의 가이드라인에 따라 REST API 설계 원칙을 적용해 모듈간 의존성을 줄이고 Flask 기반 경량 서버 구조를 완성했다.

7. 단위 불일치 문제는 수집 시점에 통일된 단위로 변환하는 전처리 로직을 추가하여 시각화 정확도를 향상시켰다.

8. FaskAPI를 통해 백엔드 서버를 구축하고 추상화 계층으로 활용하여 api/routes/ 디렉토리 안에 기능별로 파일을 분리하여 프론트엔드에 필요한 데이터만 제공하는 api 엔드포인트를 설계하였다.

9. models/디렉토리에서 Pydantic 모델을 사용하여 API 요청 및 응답 형식을 명확하게 정의하였다.

10. 하이브리드 접근 방식을 채택하여 초기에는 서버에서 랜더링 하여 빠르게 보여준 후, static/js/api/ 폴더의 js파일들이 주기적으로 백엔드 API를 호출하여 진행하였다.

11. 백엔드로부터 받은 JSON 데이터를 static/js/charts/의 스크립트들이 Chart.js 같은 라이브러리에 전달하여 페이지를 새로고침하지 않고 그래프와 데이터를 부드럽게 변할 수 있도록 하였다.

12. 명확한 디렉토리 구조를 설계하여 따르도록 했다.

13. ocker.stats(stream=False) API의 CPU 계산 공식을 적용하고,
CPU 코어 수 대비 퍼센트 계산 로직을 표준화하였다. psutil은 노드 단위 집계 전용으로 한정하여 데이터 왜곡을 방지하였다.

14. MonitoringCharts.initNetworkTrafficChart() 내에서 borderColor, backgroundColor를 기본값으로 지정하여 정상 렌더링되도록 수정하였다.

15. alert_uid를 기준으로 활성 알림(status='active') 중복 여부를 검사한 후,
기존 알림이 존재하면 새로 생성하지 않고 updated_at만 갱신하도록 로직을 변경하였다. 알림 규칙별 Cooldown(트리거 간격) 설정을 도입하여 불필요한 중복 알림을 방지하였다.

4. 프로젝트를 통해 배우거나 느낀 점

1. 단일 로컬 환경이 아닌 다중 Docker/클라우드 환경에서의 실전 경험을 통해, 표준화 된 인프라 설계와 환경 간 호환성 확보의 중요성을 이해할 수 있었음.

2. Docker , Kubernates , AWS, RDS 등 최신 클라우드 기술을 실습하여 클라우드 네이티브 개발 역량을 강화할 수 있었음.

3. 기술 문서 활용 능력과 오픈소스를 활용하고, 에러 핸들링 및 디버깅 역량이 크게 향상됨.

4. 단순 구현을 넘어 실제 서비스를 염두에 둔 아키텍처 설계 경험을 통해 실무 대응 능력 및 팀원 간 협업 역량을 향상시킬 수 있었음.

5. 복잡한 도커와 같은 외부 시스템을 다룰 때, 중간에 맞춤형 백엔드를 두는 것이 얼마나 효과적인지 다시 한번 깨달았습니다. 프론트엔드와 백엔드의 관심사를 명확히 분리하여 작업을 하는 것에 대한 중요성을 알게 되었습니다.
6. 동적인 데이터 시각화를 위해 서버 사이드 렌더링의 장점과 클라이언트 사이드 렌더의 장점을 결합하는 것이 유용하다는 것을 알게되었습니다.
7. 잘 설계된 구조는 시작이 반이다라는 말을 체감하게 되었습니다. FaskAPI가 제공하는 의존성 주입 시스템과 함께 이런 모듈화 구조를 채택한 덕분에 새로운 기능을 추가하거나 기존 기능을 수정할 때 영향을 받는 범위를 최소화할 수 있었습니다.

Ⅳ. 기대효과 및 활용분야

1. 프로젝트의 기대효과

최근 대두되고 있는 클라우드 컴퓨팅 기술인 가상화 컨테이너 기술로 구현한 CaaS(Containers-as-a-Service) 시스템 어플리케이션 서버구축 방안 산출물을 기업에 제공하여, 기존 IT환경의 운영환경 또는 S/W를 클라우드 컴퓨팅 기반의 도커 환경으로 손쉽게 이관할 수 있는 가이드를 제시할 수 있음.

- 특히, 단순히 하나의 컨테이너 시스템을 모니터링 할 수 있는 기능에 더하여, 온프레미스 및 퍼블릭 클라우드, 프라이빗 클라우드에서 운영하는 컨테이너 시스템을 통합 모니터링 할 수 있는 하이브리드 클라우드 시스템을 모니터링 제공하는 것에 큰 이점이 있음.
- 클라우드 기반의 컨테이너 인프라를 도입하고자 하는 기업 및 기관의 모니터링 기능의 요구사항에 대하여, CaaS(Containers-as-a-Service) 시스템의 모니터링 및 관리 기능을 제공하는 어플리케이션을 개발 오픈소스 산출물로 등록하여, 니즈가 있는 기업/기관에서 활용할 수 있도록 함.
- 본 프로젝트는 CaaS(Containers-as-a-Service) 시스템의 컨테이너 자원 사용량 모니터링 및 관리 기능을 시각화 웹 대시보드를 통해 제공하여, 시각적인 인사이트를 제공할 수 있음
- 기업의 Legacy환경과 클라우드 기반기술습득을 통해 기업의 IT환경을 이해할 수 있고, 실무적인 역량을 배양하여 본인이 희망하는 취업기회를 잘 선택할 수 있도록 도울 수 있음.

2. 프로젝트의 활용분야

- 컨테이너 환경을 도입하는 기업/기관에서 하이브리드 환경의 모니터링에 대한 요구사항을 대응할 수 있음

- 참여 멘티들에게 ICT 기반 기술인 운영체제, 프로그래밍, 데이터베이스 기술 역량을 부여할 수 있음.
- 최신 ICT트렌드인 클라우드 컴퓨팅 기술 및 가상화 컨테이너 기술인 CaaS(Container as a

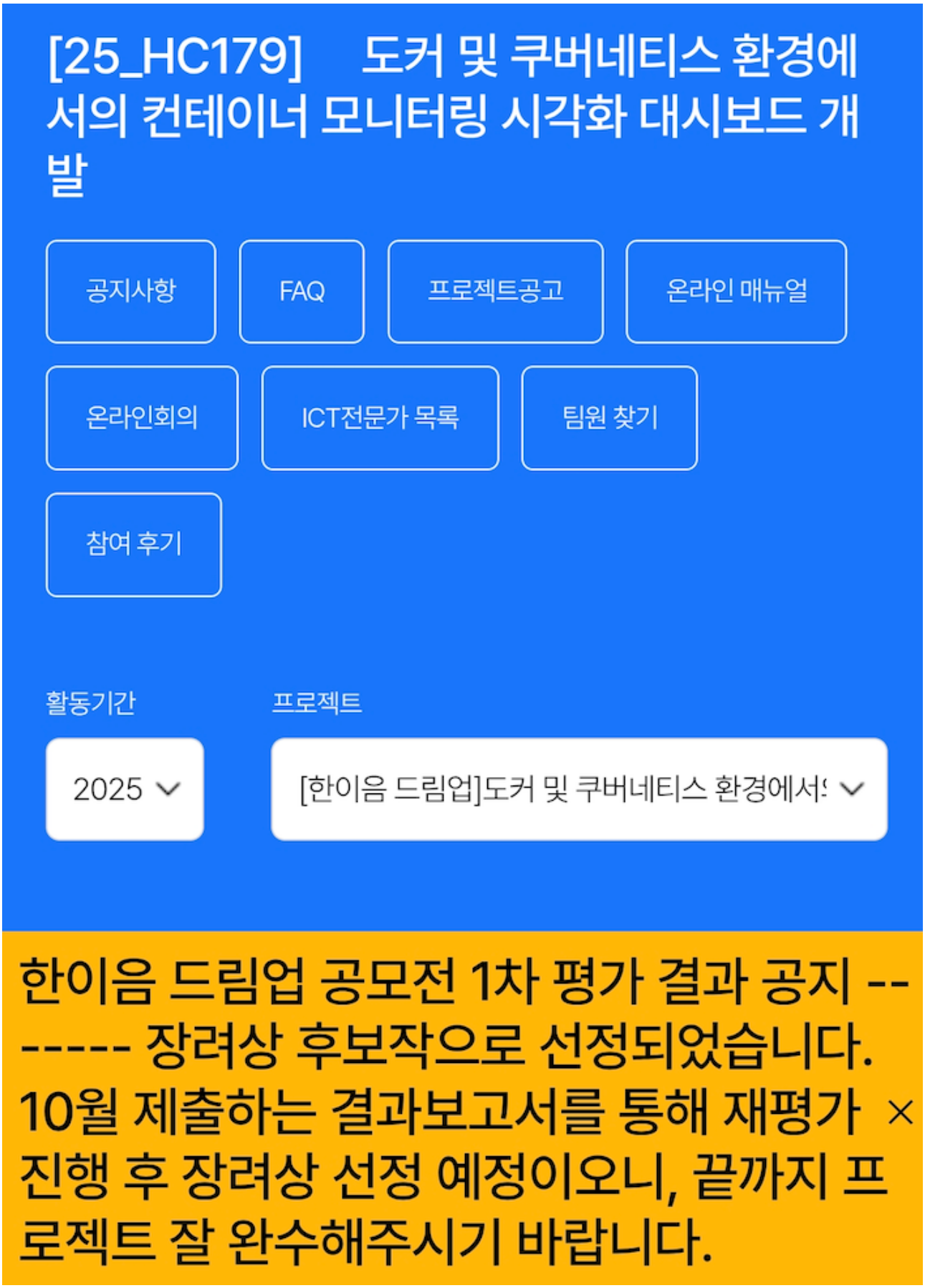
Service)기술 역량을 강화할 수 있음.
- 기업의 Legacy환경경험 및 클라우드 컴퓨팅 환경 기술습득을 통해 기업의 IT환경을 이해할 수 있고, 실무적인 역량을 배양하여 본인이 희망하는 취업기회를 잘 선택할 수 있도록 도울 수 있음.

Ⅴ. 참고자료

1. 참고 및 인용자료

-

별첨1 2025년 한이음 드림업 프로젝트 성과 증빙자료


[25_HC179] 도커 및 쿠버네티스 환경에서의 컨테이너 모니터링 시각화 대시보드 개발
공지사항
FAQ
프로젝트공고
온라인 매뉴얼
온라인회의
ICT전문가 목록
팀원 찾기
참여 후기
활동기간
2025
프로젝트
[한이음 드림업]도커 및 쿠버네티스 환경에서
한이음 드림업 공모전 1차 평가 결과 공지 ------ 장려상 후보작으로 선정되었습니다. 10월 제출하는 결과보고서를 통해 재평가 × 진행 후 장려상 선정 예정이오니, 끝까지 프로젝트 잘 완수해주시기 바랍니다.