JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 1, pp. 165-176, Jan. 2024

Check for updates

쿠버네티스를 활용한 컨테이너 기반 프로그래밍 과제 제출 환경 통합 플랫폼 구현

금 나 연[1] · 김 민 지[1] · 이 종 우[2*]
[1]숙명여자대학교 IT공학전공 학사과정
[2*]숙명여자대학교 인공지능공학부/ICT융합연구소 교수

Implementing an Environment-Integrated Container-Based Platform for Submitting Programming Assignments Using Kubernetes

Nayeon Keum[1] · Minji Kim[1] · Jongwoo Lee[2*]

[1]Bachelor's Course, Department of IT Engineering, Sookmyung Women's University, Seoul 04310, Korea
[2*]Professor, Division of Artificial Intelligence Engineering, Sookmyung Women's University, Seoul 04310, Korea

[요 약]

최근 교육부에서는 디지털 인재 양성을 위해 2025년부터 초등학교와 중학교에서 코딩교육을 의무화할 것이라고 밝혔다. 대규모의 수강생들이 프로그래밍 언어 과제를 수행할 때 다양한 문제가 발생할 수 있는데, 대표적으로는 상이한 개발 환경과 대규모 과제 관리에 관한 문제이다. 이를 해결하기 위해 현재 사용하는 방법들은 많은 비용과 관리를 위한 노력을 필요로 한다. 이와 관련된 다양한 연구들이 제시되고 있지만, 단순 스크립트 실행 결과만 확인하거나 특정 언어에서만 수행 가능하다는 점에서 한계가 있다. 본 논문에서는 이러한 기존 방법의 한계를 개선하여 학생별로 독립적인 컨테이너를 할당하고 채점자는 이러한 리소스들을 손쉽게 관리하고 열람할 수 있는 플랫폼을 Kubernetes를 활용하여 구현하였다. 사용성에 대한 설문조사와 각종 평가를 종합한 결과, 본 플랫폼을 이용하면 수강생과 채점자 모두 시간과 노력을 절감할 수 있을 것으로 예상되며, 기존 플랫폼이나 구현 방식과 비교해보았을 때 사용성이나 비용 절감 등에서 큰 차별성을 가진다는 것을 확인할 수 있었다. 그리고 오픈소스로써 자유롭게 사용자화하고 배포하여 상업적으로도 사용할 수 있어 비슷한 플랫폼을 구현할 때 참고 자료가 될 수 있을 것으로 기대한다.

[Abstract]

Recently, the Ministry of Education announced that coding education will be mandatory in elementary and middle schools starting in 2025. In the process of learning programming languages, problems due to different environments and problems related to large-scale task management may arise when students perform tasks. The methods used by most companies to solve this problem require immense costs and efforts for operations and management. Studies related to these problems are limited owing to script execution results or usage of only specific languages. Therefore, using Kubernetes, we implement a platform that assigns independent containers to each student and allows graders to easily manage and view these resources. This platform will save both students' and scorers' time and effort as it has shown significant differences in usability and cost savings compared to traditional platforms or implementations. Moreover, it is expected to serve as a reference when implementing a similar platform, as it can be freely customized, deployed, and commercially utilized as an open source.

색인어 : 오픈소스, 컨테이너, 쿠버네티스, 클라우드 컴퓨팅, 플랫폼
Keyword : Opensource, Container, Kubernetes, Cloud Computing, Platform

http://dx.doi.org/10.9728/dcs.2024.25.1.165

Received 24 October 2023; **Revised** 05 December 2023
Accepted 21 December 2023

***Corresponding Author; Jongwoo Lee**

Tel: +82-2-710-9952
E-mail: bigrain@sm.ac.kr

Ⅰ. 서 론

최근 교육부에서는 디지털 인재 양성을 위해 2025년부터 초등학교와 중학교에서 코딩교육을 의무화할 것이라고 밝혔다[1]. 이로 인해 코딩 교육 시장이 지금보다 더 큰 상승세로 커질 것으로 보이며, 동시에 프로그래밍 과제 제출 플랫폼의 필요성이 증가하고 있다.

프로그래밍 언어에 대해 배우는 과정에서 큰 규모의 수강생들이 과제를 수행할 때 다양한 문제가 발생할 수 있는데, 대표적으로는 상이한 개발 환경으로 인한 문제와 대규모 과제 관리에 관한 문제이다. 운영체제가 다르거나 사용하는 프로그래밍 언어의 버전, 통합 개발 환경 등이 달라 학생의 환경에서는 정상적으로 실행되어도 채점자의 환경에서는 다르게 보이는 문제가 생길 수 있다. 또, 채점자가 학생들의 코드를 하나하나 다운로드 받아 열어 보고, 실행시켜 결과를 확인하는 것은 많은 시간과 노력이 든다.

이를 해결하기 위해서는 두 가지 방법이 사용되는데, 수강생들에게 서버 또는 가상 환경을 배포하거나 하나의 거대한 리눅스 서버를 이용하여 수강생 수만큼 계정을 생성하는 방법이다. 그러나 이 방법들은 많은 비용과 운영, 관리를 위한 노력을 필요로 한다. 또한 보안, 설치 프로그램, 권한 설정 등 수강생 간의 충돌 문제나 용량에 관한 문제가 발생할 가능성이 높다.

이러한 문제를 해결하기 위해 과제 제출 플랫폼과 관련해서는 웹 기반 자바 가상교육시스템[2], 웹 브라우저 내 리눅스(Linux) 기반 C 프로그래밍 실습 시스템[3] 등의 방법이 제시되고 있지만, 파일의 실행 결과 확인만 가능하거나 특정 언어만 수행 가능하다는 점에서 한계가 있다.

또한 현재 존재하는 쿠버네티스(Kubernetes)를 활용한 플랫폼에 대한 연구는 특정 프레임워크를 위한 클러스터의 성능 분석[4]이나 기계 학습[5] 등으로 제한적이다.

본 논문에서는 이러한 기존 방법의 한계를 개선하기 위해 쿠버네티스를 사용하여 학생별로 독립적인 컨테이너를 할당하고 채점자는 이러한 리소스들을 손쉽게 관리하여 열람할 수 있는 플랫폼을 구현하였다.

이 플랫폼의 사용성을 입증하기 위해 예상되는 사용자층에서 총 50명을 대상으로 설문조사를 시행하였고 유사한 타 플랫폼 4개와의 정량적인 비교, 컨테이너 방식과 그 외 방식들 사이의 정량적인 비교를 통해 본 플랫폼이 가지는 경쟁력과 차별성을 확인하였다.

그리고 개발에 사용된 모든 프레임워크와 기술을 오픈소스로만 채택하고, 개발물을 깃허브에 공개하여 누구든 재현 가능하고 변형하여 모든 용도로 사용할 수 있도록 하였다. 따라서 앞으로 쿠버네티스 기반 프로그래밍 과제 제출 플랫폼을 구현하고자할 때, 참고 자료가 될 수 있을 것으로 기대한다.

Ⅱ. 관련 연구

[2]에서는 자바 컴파일러 실행이 웹 기반으로 이루어지는 자바 프로그래밍 교육을 위한 가상교육시스템을 구현하였다. HTML과 JSP 페이지로 웹 페이지를 제작하고 자바 컴파일러 및 사용자 실행 프로그램 관리 로직은 JavaBeans를 이용하여 처리하였다. 그러나, 웹에서 단순히 자바 파일을 실행하는 정도에 그치며, Java 외의 다른 프로그래밍 언어는 불가능하다는 한계가 있다.

[3]은 인터넷 상에서 웹 브라우저만으로 C 프로그래밍 실습 과정인 프로그램의 작성/수정/저장/컴파일/수행뿐만 아니라 Linux 운영체제 명령어를 사용한 프로그램 파일 관리가 가능한 C 프로그래밍 실습 시스템을 개발하였다. 웹 브라우저에서 프로그래밍 코드 작성 및 실행이 가능할 뿐만 아니라, Linux 운영체제 명령어를 통해 파일 관리가 가능하다는 이점이 있으나, C 언어만 수행 가능하였다. 또, 해당 논문을 이용하여 실제 시스템을 구현하기 위한 프로그램 로직에 대한 세부 설명은 존재하지 않았다.

[4]에서는 쿠버네티스 환경에서 MySQL 클러스터의 성능에 관한 연구를 통해 쿠버네티스의 pod(배포 가능한 가장 작은 컴퓨팅 단위)에서 데이터베이스 sharding(대규모 데이터베이스를 여러 머신에 저장하는 프로세스)이 MySQL 클러스터의 성능에 미치는 영향을 탐색하기 위한 실험을 수행하였다. 쿼리 프로세스를 분석하고 부정적인 영향을 미치는 요소를 찾은 다음에 MySQL 클러스터 매개변수를 조정하여, MySQL 클러스터 성능 향상을 꾀하고, 쿠버네티스에서 MySQL 클러스터를 효과적으로 배치하는 방법을 제시했다. 그러나, MySQL이라는 특수한 프레임워크에 대한 성능 분석만을 하여 범용 플랫폼 개발과는 차이가 있다.

[5]는, 클라우드에서 인프라 오류가 발생하여 노드 및 서비스 오류로 이어지는 경우가 많다는 것에 착안하여 서비스 다운타임을 피하기 위해 올바른 장애 예측이 가능하고 쿠버네티스 플랫폼에서 자동 서비스를 유지하기 위해 적절한 조치를 취하는 기계 학습 기반 설계 솔루션을 제안하였다. 그러나 장애 예측이라는 특수한 도메인을 가지고 있으며 서비스나 플랫폼 구현에 관련된 정보는 미흡하였다.

관련된 기존 연구들을 조사한 결과, 웹 기반 자바 가상교육시스템, 웹 브라우저 내 Linux 기반 C 프로그래밍 실습 시스템, 쿠버네티스에서 구현한 MySQL 클러스터의 성능 분석, 쿠버네티스 장애 예측 등의 연구가 존재하였다. 그러나, 본 연구는 쿠버네티스를 기반으로 수강생들에게 컨테이너를 할당하고, 다양한 언어(Python, C, C++)를 지원하는 플랫폼을 구현하였다. 그리고 구현에 사용된 모든 로직과 코드를 깃허브에 공개하였기 때문에 기존 관련 연구들과는 차별점이 있다.

Ⅲ. 배경 지식

본 장에서는 본 논문에서 구현한 플랫폼을 개발하는 데 사용된 오픈소스와 기술, 프레임워크의 배경 지식에 대해 설명한다.

3-1 도커와 쿠버네티스

1) 컨테이너와 가상머신

컨테이너(Container)와 가상 머신(VM)은 애플리케이션을 IT 인프라 리소스로부터 독립적으로 만드는 배포 기술이다. 컨테이너와 가상 머신은 기본 인프라를 가상화하거나 추상화하므로 사용자가 이를 신경 쓸 필요가 없으며 소프트웨어 인프라를 이미지 파일이라는 단일 파일로 패키징할 수 있다.

주요 차이점으로 컨테이너는 운영 체제(소프트웨어)의 커널을 가상화하여 애플리케이션이 모든 플랫폼에서 독립적으로 실행될 수 있도록 하고, 가상 머신은 물리적 시스템(하드웨어)을 가상화할 수 있어 하드웨어 리소스를 효율적으로 사용할 수 있다는 것이 있다.

이는 비용 절감과 성능 증가로 이어지는데, 컨테이너화된 앱의 인스턴스는 가상머신에 비해 사용하는 메모리가 훨씬 적고, 가동과 중단이 더 빠르며, 호스트 하드웨어 상의 밀집도가 훨씬 더 높아 IT 지출 감소로 이어진다[6]. 또한, 최소한의 CPU와 메모리만 사용하여 비용 절감과, 부하가 작아 높은 성능 제공할 수 있다는 장점이 있다[7]. 그리고 다음에 소개할 쿠버네티스와 함께 사용하면 애플리케이션 내 단일 구성 요소들이 기술 스택의 나머지 부분에 영향을 미치지 않고 업데이트되므로 운영 중단이 최소화되어 관리를 위한 노력과 비용이 절감될 수 있다[8].

그 밖에도 가상 머신은 운영 체제와 호스트 운영 체제 간에 통신하는 하이퍼바이저(Hypervisor)를 사용하고, 컨테이너는 컨테이너 엔진 또는 컨테이너 런타임을 사용한다. 컨테이너 엔진은 컨테이너와 운영 체제 간의 중개 에이전트 역할을 하며 도커(Docker)는 가장 많이 사용되는 오픈 소스 컨테이너 엔진이다. 또한 가상 머신 이미지 파일에는 자체 운영 체제가 포함되어 있으므로 크기가 더 크고, 컨테이너는 단일 애플리케이션을 실행하는 데 필요한 리소스만 패키징한다는 차이가 있다[9].

2) 도커와 쿠버네티스

컨테이너를 사용할 때 컨테이너를 쉽게 받거나 공유하고 구동할 수 있도록 해주는 컨테이너 런타임이 필요하다. 가장 대표적인 컨테이너 런타임인 containerd를 감싸 컨테이너 생명주기 외에도 다양한 기능을 수행할 수 있게 해주는 컨테이너 엔진은 도커이다[10]. 도커는 개발자가 컨테이너를 빌드, 배포 및 실행하도록 지원하는 상용 컨테이너화 플랫폼 및 런타임으로 단일 API를 통한 간단한 명령과 자동화를 갖춘 클라이언트-서버 아키텍처를 사용한다. 도커파일(Dockerfile)을 작성한 다음 도커 데몬 서버(dockerd)를 사용하여 이미지를 구축하는 적절한 명령을 실행하여, 변경할 수 없는 컨테이너 이미지로 애플리케이션을 패키징할 수 있다[11].

도커는 컨테이너화된 애플리케이션을 패키징하고 배포하는 효율적인 방법을 제공하지만, 도커만으로는 대규모로 컨테이너를 실행하고 관리하기 어렵다. 별다른 설정을 하지 않으면 단일 호스트에서만 동작하는 것이 주된 이유인데, 여러 서버/클러스터에서 컨테이너를 조정 및 예약하고, 가동 중지 시간 없이 애플리케이션을 업그레이드 또는 배포하고, 컨테이너의 상태를 모니터링하는 것 외에도 고려해야 할 많은 사항이 있어 이와 같은 문제를 해결하고자 컨테이너를 오케스트레이션하는 솔루션이 등장하였고, 그중 가장 대표적인 컨테이너 오케스트레이션 툴이 쿠버네티스이다[11].

쿠버네티스는 구글에서 출시한 컨테이너 오케스트레이션 툴로, 여러 노드에 컨테이너를 분산해서 배치하거나, 문제가 생긴 컨테이너를 교체하거나, 컨테이너가 사용할 비밀번호나 환경 설정을 관리하고 주입해주는 등 오케스트레이션의 역할을 한다[10]. 컨테이너 런타임으로써 도커를 사용했으나, CRI(Container Runtime Interface)에 호환되지 않아 v1.24부터는 도커 지원을 중단하였고, 그 외에 CRI-O, containerd 등의 컨테이너 런타임을 사용한다.

이를 통해 대량의 컨테이너와 사용자를 관리하고 부하를 효율적으로 분산하며 인증 및 보안, 다중 플랫폼 배포를 제공한다[11]. 쿠버네티스 외에도 도커 스웜(Docker Swarm), 메소스(Mesos), 하시코프 노마드(HashiCorp Nomad) 등이 있다.

3) 쿠버네티스 클러스터 컴포넌트 구조

쿠버네티스의 컴포넌트는 그림1과 같이 컨트롤 플레인(Control Plane)과 데이터 플레인(Data Plane)으로 나뉜다.

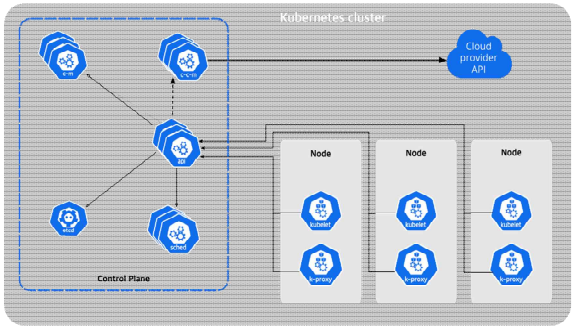

그림 1. 쿠버네티스 컴포넌트[12]
Fig. 1. Components of Kubernetes[12]

컨트롤 플레인은 전반적인 클러스터의 상태와 워커 노드에 할당되는 파드(pod)를 관리한다. 프로덕션 환경에서는 일반적으로 컨트롤 플레인이 여러 컴퓨터에 걸쳐 실행되고, 클러스터는 보통 여러 노드를 실행하므로 내결함성과 고가용성이

제공된다. 컨트롤 플레인에는 쿠버네티스 API를 노출하는 kube-apiserver, 클러스터의 데이터를 담는 키-값 저장소인 etcd, 실행할 노드를 스케줄링하는 kube-scheduler, 컨트롤러 프로세스를 실행하는 kube-controller-manager 등이 있다[12].

쿠버네티스는 컨테이너를 파드 내에 배치하고 노드에서 실행함으로 워크로드를 구동하는데, 노드는 가상 또는 물리적 머신이며 각 노드는 컨트롤 플레인에 의해 관리된다. 데이터 플레인, 즉 노드 컴포넌트에는 각 노드에서 실행되어 제공된 파드 스펙(PodSpec)에 따라 동작하는 지 확인하는 에이전트 kubelet, 네트워크 규칙을 유지 관리하여 여러 노드 사이의 네트워크 접근이 가능하도록 하는 네트워크 프록시인 kube-proxy, 그리고 컨테이너 실행을 담당하는 소프트웨어인 컨테이너 런타임으로 구성되어 있다[13].

4) 관련 AWS 서비스들

AWS에서 컨테이너를 운영하는 기본적인 방법은 Amazon EC2를 이용하여 직접 가상 서버를 띄우고, 도커나 쿠버네티스 클러스터를 설치하여 컨테이너를 배포하는 방식이다. 그러나 프로덕션 서비스를 운영하기 위해서 더 관리하기 쉬운 컨테이너 솔루션을 선택할 수 있다.

소규모로 컨테이너 관리 서비스를 이용하기 위해서는 Amazon Lightsail, AWS Elastic Beanstalk 등을 사용할 수 있으며 대규모 컨테이너를 이용한 완전관리형(full-managed) 서비스를 이용하려면 Amazon Elastic Kubernetes Service (Amazon EKS)나 Amazon Elastic Container Service(Amazon ECS)를 사용할 수 있다. 이 밖에도 서버리스 실행을 위한 AWS Fargate, AWS Lambda 등의 서비스도 있다[14].

3-2 리액트(React)

React는 사용자 인터페이스를 만들기 위한 자바스크립트 라이브러리이다[15]. 상호작용이 많은 사용자 인터페이스(UI; User Interface)를 만들 때 생기는 어려움을 줄이고자 애플리케이션의 각 상태에 대한 간단한 뷰를 설계하면 데이터가 변경됨에 따라 적절한 컴포넌트만 효율적으로 갱신하고 렌더링한다. 이러한 선언형 뷰는 코드를 예측 가능하고 디버그하기 쉽게 만들어준다. 컴포넌트 기반으로 사용자 인터페이스를 구성하여 다양한 형식의 데이터를 앱 안에서 손쉽게 전달할 수 있고, DOM과는 별개로 상태를 관리할 수 있다.

본 플랫폼은 리액트를 타입스크립트(TypeScript)로 개발하였는데, 타입스크립트는 자바스크립트 코드베이스에 type 정의를 추가하는 방법으로, JSX를 지원하며 useState과 같은 리액트 코드 베이스에서 사용되는 패턴을 모델링할 수 있다. 구성 요소의 프롭스(Props, 리액트에서 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달하는 방법)에 대한 type을 통해 정확성을 확인하고 코드의 안정성과 신뢰성을 높이고자 이를 채택하였다[16].

3-3 스프링(Spring)

스프링은 자바 플랫폼을 위한 오픈 소스 어플리케이션 프레임워크이다. 스프링은 경량 컨테이너로 자바 객체를 직접 관리하며, 각각의 계층이나 서비스들 간에 의존성이 존재하는 경우 프레임워크가 서로 연결시켜주는 의존성 주입(DI; Dependency Injection)을 지원한다. 객체의 생명주기를 관리하고 각 계층이나 서비스들 간의 의존성을 맞추는 기능들은 주로 환경설정을 담당하는 XML파일에서 설정되고 수행되었으나 현재는 자동 구성(Auto Configuration)을 사용한 YML파일로 구성하기도 한다. 스프링은 기존 라이브러리를 감싸는 정도로 사용이 가능하므로 수많은 라이브러리가 지원된다는 점에서 확장성이 높고 특정 라이브러리를 별도로 분리하기도 용이하다. 예시로 데이터베이스에 접속하고 자료를 저장 및 읽어오기 위한 여러가지 유명한 라이브러리(JDBC, MyBatis, Hibernate 등)에 대한 지원 기능을 제공한다. 스프링은 MVC(Model-View-Controller) 패턴을 사용하는데, DispatcherServlet이 컨트롤러 역할로써 각종 요청을 적절한 서비스에 분산시켜주며 이를 각 서비스들이 처리 하여 결과를 생성하고 그 결과는 다양한 형식의 뷰 서비스로 화면에 표시된다[17].

스프링부트(SpringBoot)는 독립형 스프링 어플리케이션을 실행할 수 있도록 도와주는 프레임워크이다. 스프링부트는 자체적으로 톰캣(Tomcat)이라는 서버를 가지고 있기 때문에 어플리케이션 생성 시에 별도의 서버 설정을 하지 않아도 된다. 이는 빌드와 배포 단계를 축소 시켜 편리함을 가져다준다. ‘starter’ 의존성을 제공하여 빌드 구성을 단순화하였다[18].

Ⅳ. S/W 아키텍처

본 장에서는 구현한 플랫폼의 소프트웨어 아키텍처에 대해 설명한다. 전체 시스템의 개요와 쿠버네티스 클러스터 아키텍처, 서비스 아키텍처, 클라우드 인프라 프로비저닝 파이프라인, CI/CD 파이프라인 등의 시스템 구성에 대해 자세히 제시한다.

4-1 시스템 개요

본 플랫폼의 시스템 아키텍처는 크게 4가지로 구분할 수 있으며 시스템 개요는 그림 2와 같다. 그림 2의 원본 이미지 링크는 부록에 추가로 첨부하였다. 4.2 절 쿠버네티스 클러스터 아키텍처에서는 쿠버네티스 클러스터 내부에서 서비스의 구조를 설명한다. 4.3 절 서비스 아키텍처에서는 본 플랫폼의 WEB, WAS 서버의 구현에 대해 서술한다. 4.4 절 클라우드 인프라 프로비저닝 파이프라인에서는 본 플랫폼을 클라우드에 배포하기 위해서 사용한 오픈소스 프레임워크 및 작동 과정을 기술하고, 마지막으로 4.5 절 CI/CD 파이프라인에서 서

비스 및 인프라 개발을 편리하고 안정적으로 하기 위해 구현한 다양한 파이프라인을 각각 설명한다.

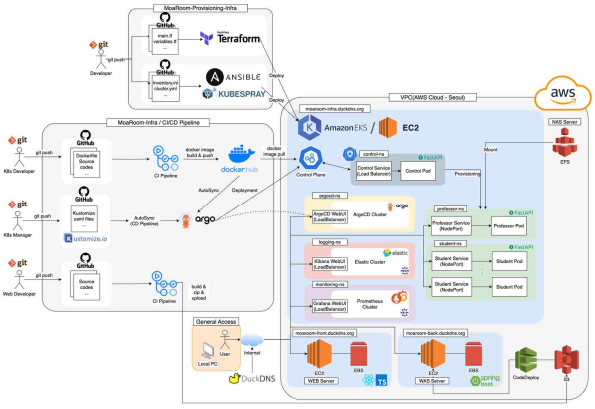

그림 2. 시스템 아키텍처
Fig. 2. System architecture

4-2 쿠버네티스 클러스터 아키텍처

본 플랫폼은 쿠버네티스 클러스터에서 동작한다. 서비스를 구성하는 오브젝트들은 용도와 역할에 따라 컨트롤(Control), 교수(Professor), 학생(Student) 세 개의 네임스페이스(Namespace)에 분리되어 있다.

1) 컨트롤 오브젝트

컨트롤 네임스페이스(Control Namespace)에는 오브젝트를 생성하고 삭제할 수 있는 kube-apiserver로의 접근권한을 가진 중앙 관리 서버(Control Server)가 있다. 중앙 관리 서버는 백엔드 서버로부터 교수, 학생 생성 요청이 왔을 때 kube-apiserver에 접근하여 이를 수행하며, 유저 파드에 ssh로 접속하기 위한 엔드포인트(Endpoint)를 반환하는 등의 역할을 수행한다. 백엔드 서버에서 접근할 수 있는 유일한 파드이며, 프론트엔드 서버에서는 이를 확인할 수 없다.

쿠버네티스에서 오브젝트를 생성/삭제하기 위해서는 kube-apiserver에 접근해야하는데, 이는 일반적인 오브젝트가 수행할 수 없고 특별한 권한이 필요하다. 쿠버네티스에서 권한 관리는 역할-기반으로 이루어지며 이를 RBAC(Role-Based Access Control)이라고 한다. Role이나 ClusterRole을 이용하여 kube-apiserver로부터 어떤 API를 이용할 수 있는지를 정의하고 쿠버네티스의 사용 권한을 정의한다. 이를 사용자/그룹 또는 Service Account와 연결하는 RoleBinding/ClusterRoleBinding을 정의하여 특정 오브젝트가 역할을 가지도록 설정할 수 있다. 일반 Role, RoleBinding과 ClusterRole, ClusterRoleBinding의 차이는 특정 네임스페이스에 한정된 정책인지, 클러스터 전체에 한정된 정책인지이다. 본 플랫폼에서 사용한 RBAC 아키텍처는 그림 3과 같다.

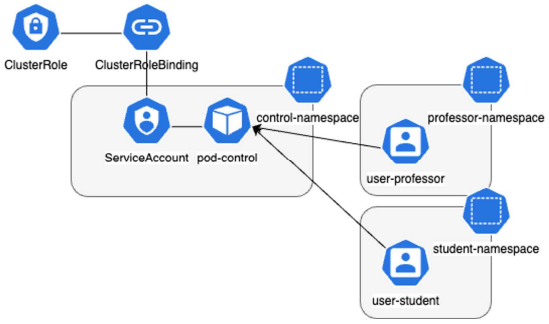

그림 3. RBAC 아키텍처
Fig. 3. RBAC architecture

컨트롤 네임스페이스의 중앙 관리 서버에 kube-apiserver로부터 모든 apiGroup과 파드, 서비스들(Service), nodes 리소스에 대해 모든 작업을 허용하는 ClusterRole을 설정한다. 그리고 이를 중앙 관리 서버와 연결하는 ClusterRoleBinding을 만들고 이를 디플로이먼트(Deployment) 타입의 중앙 관리 서버에 연결한다. 이를 통해 교수나 학생이 직접 kube-apiserver에 접속할 수 없고 오직 중앙 관리 서버를 통해서만 오브젝트를 관리할 수 있게 하여 안정성과 신뢰성을 높였다.

2) 교수와 학생 오브젝트

파드는 쿠버네티스에서 생성하고 관리할 수 있는 배포 가능한 가장 작은 컴퓨팅 단위로 하나 이상의 컨테이너의 그룹이다[19].

서비스는 파드 집합에서 실행 중인 애플리케이션을 네트워크 서비스로 노출하는 추상화 방법이다[20]. 컨테이너는 생성될 때마다 IP 주소가 변경되는데, 이에 정적인 네트워크 엔드포인트를 연결해주기 위한 구현체이다. 서비스를 파드와 연결시키게 되면, 서비스로 라우팅되는 트래픽을 파드로 포워딩할 수 있다. Kube-proxy에 netfilter, iptable을 업데이트하는 기능이 있어 이를 가능케 한다. 서비스는 논리적인 컨셉이며, kube-apiserver에서 서비스의 생성, 삭제를 담당하고 IP를 할당한다. 서비스가 생성될 때, coreDNS라는 쿠버네티스 내부 DNS 서버에 서비스의 dns와 ip가 저장되어 외부 인터넷을 거치지 않고 서비스 디스커버리(Service Discovery) 메커니즘을 사용할 수 있다.

교수와 학생 네임스페이스에는 실제 교수, 학생이 사용하는 컨테이너가 파드의 형태로 각각 존재하며, 서비스를 이용하여 컨테이너를 외부에 연결한다. 교수와 학생 서버 그리고 관리 서버는 모두 파드와 서비스로 구성되고 각 서비스는 외부에서 접근 가능하도록 구현되어 있다. 쿠버네티스 노드의 외부 IP에서 포트 기반으로 각 서비스에 접근하여 필요한 API를 호출하거나 파드의 컨테이너에 ssh로 접속하기 위한 WebSSH 서버가 띄워져 있다.

교수 서버는 과제를 생성할 때 마감 기한에 학생의 컨테이너로부터 과제를 반환할 크론잡(Cronjob)과 반환될 과제들의

디렉토리를 생성하는 작업을 수행하는 API가 컨테이너 별 엔드포인트를 통해 구현되어 있다.

학생 서버에는 학생이 수강 신청한 강의의 과제가 등록되었을 때 학생 컨테이너에 과제 디렉토리를 만들고, 과제 반환 시 이를 학생 컨테이너 환경에서 실행하여 값과 실행 시간을 반환하는 API가 구현되어 있다. 또한, 과제 관리를 쉽게 하기 위해 커맨드 라인 명령어(CLI; Command Line Interface)로 구현되어 있어 간단한 명령어만으로 과제의 디렉토리로 이동하거나 과제 정보를 확인하는 등의 작업을 수행할 수 있다.

4-3 서비스 아키텍처

본 플랫폼은 Duck DNS라는 DDNS(Dynamic Domain Name System)을 이용하여 쿠버네티스 노드 서버, 프론트엔드 서버, 백엔드 서버 각각에 대한 도메인을 사용하여 배포하였다. DuckDNS를 이용하면, 클라우드에서 할당 받은 IP 주소에 대한 도메인을 무료로 추가할 수 있고, 쉽게 변경하여 선택하였다. 프론트엔드 서버와 백엔드 서버는 각각 AWS CodeDeploy, S3, GitHub Actions를 이용하여 CI/CD 파이프라인을 구성하였고, 배포 자동화가 되어 있는데 이는 4.5 절에서 더 자세히 다루도록 한다.

1) WEB 서버

프론트엔드 서버는 리액트와 부트스트랩을 이용하여 구현하였다. 리액트는 자료형의 안정성을 위해 자바스크립트가 아닌 타입스크립트를 사용하였고, 디자인을 위해 부트스트랩의 여러 가지 구현체를 사용하였다. AWS EC2에 t2.mirco 인스턴스로 우분투 22.04를 운영체제로 사용하고, 8GiB의 AWS EBS와 Elastic IP가 연결되어 있다. UI/UX 디자인은 대표적인 툴인 피그마(Figma)를 사용하였고, 개발 환경으로는 VSCode를 사용하였다. 사용자 인터페이스에 대한 자세한 내용은 다음 장에서 다룰 예정이다.

2) WAS 서버

백엔드 서버는 스프링 부트로 구현되었다. AWS EC2에 t2.medium 인스턴스로 우분투 22.04를 운영체제로 사용하고, 16GiB의 AWS EBS와 Elastic IP가 연결되어 있다. 개발 환경으로는 인텔리제이(IntelliJ)를 사용하였다.

데이터베이스는 PostgreSQL v14.9를 사용하였으며 백엔드 서버와 동일한 서버에 구축하였다. 개체-관계 다이어그램(ERD; Entity-Relationship Diagram)은 그림 4와 같다. Users 테이블에 사용자 관련 정보를, Lecture와 Assignment 테이블에 강의와 과제 관련 정보를 저장하는 스키마를 구현하였다. Step 테이블에는 과제 제출 관련 정보를, Url 테이블에는 컨테이너에 접속하기 위한 엔드포인트 정보를 저장한다.

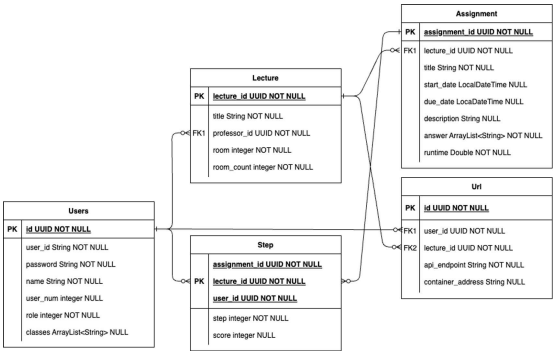

그림 4. ER 다이어그램
Fig. 4. ER diagram

4-4 클라우드 인프라 프로비저닝 파이프라인

본 플랫폼의 클라우드 인프라(Infrastructure)를 배포하기 위해서 두 개의 배포 파이프라인을 사용하는데, 테라폼(Terraform)을 이용한 파이프라인과 Kubespray을 이용한 파이프라인이다. 각 파이프라인은 배포 대상(AWS EKS, AWS EC2)에 대한 차이가 있으며, 그 외 기능하고자 하는 바, 즉 쿠버네티스 클러스터는 같은 구성으로 배포된다.

1) 테라폼

테라폼(Terrafrom)이란 클라우드나 데이터 센터에 있는 리소스를 관리하거나 프로비저닝 하기 위한 IaC(Infastructure-as-a-Code) 툴이다[21]. 하시코프(Hashicorp)에서 오픈소스로 개발 중이며 하시코프 설정 언어(HCL; Hashicorp Configuration Language)를 사용하여 클라우드 리소스를 선언한다. 버전 관리, 재사용 및 공유가 가능한 사람이 읽을 수 있는 하시코프 설정 언어로 된 구성 파일로 클라우드 및 온프레미스 리소스를 모두 정의할 수 있다. 그리고 일관된 워크플로우를 사용하여 수명주기 전반에 걸쳐 모든 인프라 자원을 프로비저닝하고 관리할 수 있다는 이점이 있다[22].

본 플랫폼을 클라우드(AWS EKS)에 배포하기 위해 테라폼을 사용하였다. 네트워크 관련(VPC, 서브넷, 가용 영역, CIDR) 리소스 정보부터 클러스터의 이름, 노드의 개수, IAM Policy 등 클러스터를 생성하기 위한 모든 정보를 정의하고 생성 완료 시 출력될 결과를 지정하였다.

2) Kubespray

Kubespray는 쿠버네티스 클러스터를 쉽게 설치하는 자동화 도구로 Ansible playbook, 인벤토리(inventory) 등의 프로비저닝 도구와 운영체제 및 클러스터의 설정 관리 작업을 코드 형으로 쉽게 설정할 수 있다[23].

테라폼과 다른 점은 서버를 미리 프로비저닝한 이후 해당 서버의 IP나 도메인을 이용하여 쿠버네티스 클러스터를 배포하는 방식으로 진행된다는 것이며 Ansible 인벤토리 파일과 다양한 속성의 사용자 정의(customize)를 이용하여 클러스

터를 배포한다.

본 플랫폼은 쿠버네티스 클러스터를 클라우드에 배포하기 위해 Kubespray와 Ansible의 조합을 이용하여 미리 프로비저닝한 AWS EC2에 배포하였다. 배포할 노드의 종류, Ansible의 Host, IP 등을 정의하고 다양한 진자(Jinja) 템플릿을 사용하여 스태틱하게 클러스터에서 동작할 리소스들을 정의하였다.

4-5 CI/CD 파이프라인

CI/CD (Continuous Integration/Continuous Delivery)는 서비스 개발 단계를 자동화하여 서비스를 더욱 짧은 주기로 사용자에게 제공하는 방법이다. CI/CD의 기본 개념은 지속적인 통합, 지속적인 서비스 제공, 지속적인 배포이고, CI/CD는 새로운 코드 통합으로 인해 개발 및 운영팀에 발생하는 문제를 해결하기 위한 솔루션을 통칭하는 단어이다.

특히, CI/CD는 애플리케이션의 통합 및 테스트 단계에서부터 제공 및 배포에 이르는 애플리케이션의 라이프사이클 전체에 걸쳐 지속적인 자동화와 지속적인 모니터링을 제공한다. 이러한 구축 사례를 일반적으로 CI/CD 파이프라인이라 부르며, 개발 및 운영팀의 애자일 방식 협력을 통해 데브옵스(DevOps) 또는 사이트 신뢰성 엔지니어링(SRE; Site Reliability Engineering) 방식으로 지원된다[24].

본 플랫폼에서는 깃허브 액션와 AWS CodeDeploy, S3를 이용하여 CI/CD 파이프라인을 구축하였다. 파이프라인은 각각 컨테이너 이미지 관련, 쿠버네티스 관련, 서비스 관련 파이프라인으로 나눌 수 있다.

1) 컨테이너 이미지 관련 CI/CD 파이프라인

도커를 이용하여 컨테이너 이미지를 빌드할 때 필요한 파일들과 도커파일을 인프라 관련 깃허브 레포지토리에 디렉토리 별로 구분하였다. 각 디렉토리에서 빌드를 수행하면 각 종류별 도커 이미지를 만들 수 있고 푸시를 통해 도커허브(DockerHub)에 등록할 수 있다.

깃허브 액션을 이용하여 CI/CD 파이프라인을 구현하여 메인 브랜치에서 각 디렉토리에 변경사항이 생길 때마다 자동으로 테스트하고, 빌드하고, 푸시하도록 워크플로우를 구현하였다.

그리고 배포된 이미지에 변경 사항이 생기면, ArgoCD를 이용하여 해당 이미지를 사용하고 있는 컨테이너가 자동으로 변경될 수 있도록 구현하였다. 이에 관한 내용은 다음 절에서 더 자세히 다룬다.

2) 쿠버네티스 관련 CI/CD 파이프라인

본 플랫폼에는 실행을 보장해야 하는 중앙 관리 서버가 존재한다. Kustomize는 쿠버네티스 오브젝트들의 구성(configuration)을 사용자 정의할 수 있는 툴이다[25].

GitOps 구현 도구 중 쿠버네티스 어플리케이션의 자동 배포를 위한 오픈소스 도구인 ArgoCD와 Kustomize를 결합하여[26] 쿠버네티스에 관리 서버가 신뢰성 있게 배포된 상태를 확인할 수 있도록 CI/CD 파이프라인을 구현하였다. 개발자가 코드 변경 사항을 푸시하면 깃허브 액션을 통해 CI 파이프라인이 적용되고, 도커허브에 빌드된 이미지가 푸시된다. 구성 관련 코드의 변경 사항을 깃허브에 푸시하면 마찬가지로 깃허브 액션을 통해 CD 파이프라인이 적용되어 ArgoCD 서버에 동기화되고, ArgoCD 서버는 동기화된 변경 사항을 지정된 쿠버네티스 클러스터에 반영한다. 통합적인 구조는 그림 5와 같다. 이전 절에서 언급했듯이 코드 변경 사항으로 인해 현재 쿠버네티스 클러스터에서 사용하고 있는 이미지가 다시 빌드된 경우에도 ArgoCD는 이를 쿠버네티스 클러스터에 동기화시키는 역할을 한다.

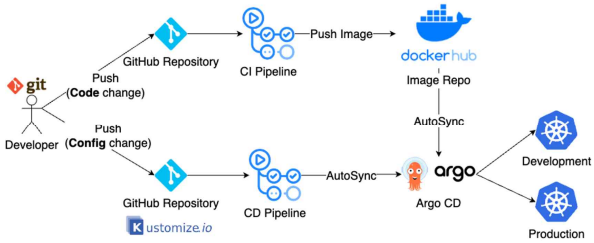

그림 5. 쿠버네티스 CI/CD 파이프라인
Fig. 5. Kubernetes CI/CD pipeline

3) 서비스 개발 CI/CD 파이프라인

프론트엔드, 백엔드를 개발할 때 깃허브 레포지토리에서 생기는 개발의 변경 사항이 지속적으로 테스트되고 배포에 반영될 수 있도록 CI/CD 파이프라인을 구축하였다. 그림 6과 같이 개발자가 코드 변경사항을 푸시하면 깃허브 액션에 구현된 워크플로우대로 파일을 빌드하여 AWS S3에 업로드하고, AWS CodeDeploy에 deployment를 생성하고 CodeDeploy는 S3에 업로드된 빌드된 파일을 가져온다. 이 파일들은 WEB, WAS 서버가 배포되어있는 AWS EC2에 각각 반영되어 실행된다.

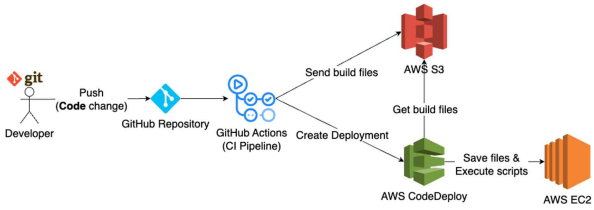

그림 6. 서비스 CI/CD 파이프라인
Fig. 6. Service CI/CD pipeline

V. 동작 과정

본 장에서는 구현한 플랫폼의 전반적인 동작 과정을 설명

한다. 사용자는 회원가입 시 학생/교수 선택을 진행하며, 본장에서는 로그인 후의 상황을 가정하여 설명을 진행한다.

5-1 강의 생성 및 수강

교수는 그림 7과 같이 마이페이지의 '강의 개설' 버튼을 통해 강의를 개설할 수 있다. 강의 개설 시 강의명, 분반, 인원수를 입력받는다. 학생은 강의 개설이 불가하므로 버튼이 화면에 표시되지 않으며, 그림 8과 같이 강의 목록 페이지의 개설된 강의 목록을 확인한 후 원하는 강의가 있다면 '강의 신청하기' 버튼을 눌러 강의를 수강할 수 있다.

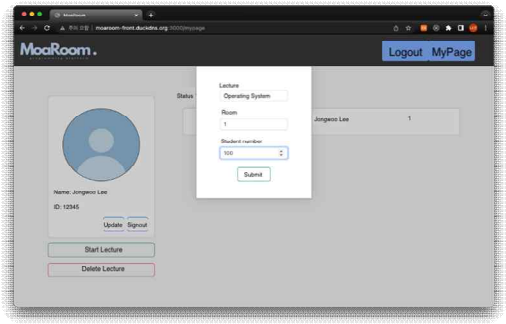

그림 7. 강의 생성 페이지
Fig. 7. Lecture creation page

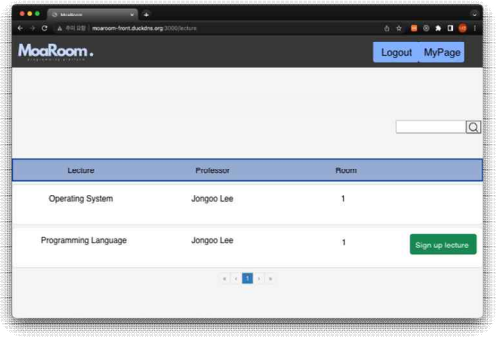

그림 8. 강의 신청 페이지
Fig. 8. Lecture assign page

5-2 과제 생성

교수는 그림 9와 같이 '과제 추가' 버튼을 통해 과제를 생성할 수 있다. 과제 생성 시 과제명, 시작일, 마감일, 설명, 답안, 실행 시간을 입력할 수 있고 시작일 미입력 시, 현 시간이 적용된다. 과제가 생성되는 시점에 학생 컨테이너에는 과제를 수행하기 위한 폴더가, 교수 컨테이너에는 과제를 저장하기 위한 폴더가 생성된다. 과제 시작일이 과제 생성 시간 이후라면 학생에게는 과제가 표시되지 않고 교수의 과제 목록에는 진행 대기 중으로 표시된다.

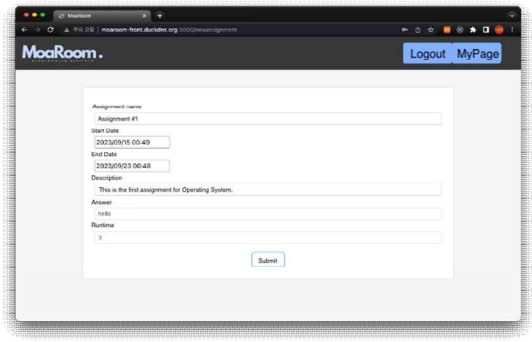

그림 9. 과제 생성 페이지
Fig. 9. Assignment creation page

5-3 과제 수행 및 마감

과제가 시작되면 학생들은 과제 페이지 내에서 웹으로 접속할 수 있는 본인 컨테이너에 접속하여 과제를 수행할 수 있으며 화면은 그림 10과 같다. 컨테이너 내부에는 'moaroom [command]'으로 사용할 수 있는 커맨드 라인 명령어가 구현되어 있어 학생은 쉽게 과제 디렉토리를 찾고 과제를 수행할 수 있다. 수행 가능한 언어는 Python, C, C++가 있다. 과제가 마감되면 자동으로 해당 과제를 수행하는 학생들의 컨테이너에서 과제 파일들이 교수 컨테이너로 넘어간다. 이때, 학생 컨테이너에서 과제의 수행 결과와 실행 시간이 함께 넘어가며, 과제 마감 시에만 학생들의 과제가 교수에게 넘어가므로 학생들은 과제 마감 전까지는 자유롭게 수정이 가능하다. 반대로 마감 후에는 수정한다고 하더라도 수정 내용이 반영되지 않는다.

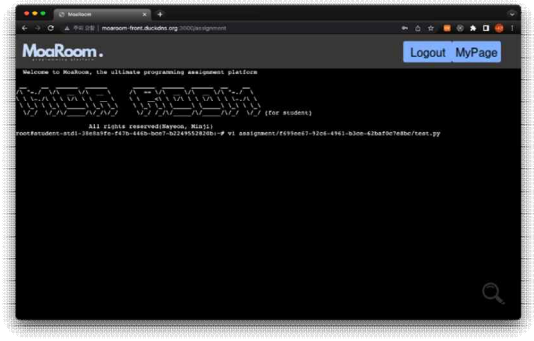

그림 10. 과제 수행 페이지
Fig. 10. Assignment page

5-4 과제 확인 및 채점

과제 마감 기한이 지나면 교수는 제출자 목록 페이지를 통해 학생들이 제출한 과제를 확인할 수 있고 해당 과제는 수동 채점 또는 일괄 채점으로 채점할 수 있다. 교수는 학생들의 과제 파일 내용과 실행 결과, 실행 시간을 한 화면에서 확인할 수 있다. 수동 채점은 학생들이 제출한 과제 내용을 보고

학생 별로 채점하여 숫자로 점수를 입력하는 방식이며, 일괄 채점은 제출자 목록에 있는 학생들의 과제의 답과 실행 시간을 정답과 비교하여 점수를 계산한 뒤 일괄 반영된다.

채점이 끝나면 학생은 그림 11과 같이 학생의 과제 제출 페이지에서 본인의 점수를 확인할 수 있다. 반영된 점수는 교수의 제출자 목록 페이지에서도 확인할 수 있다.

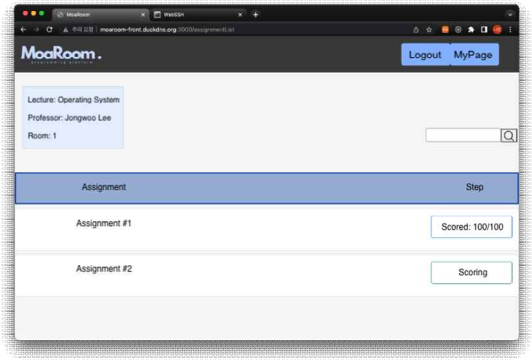

그림 11. 과제 채점 페이지
Fig. 11. Assignment scoring page

Ⅵ. 평 가

6-1 정량 평가

1) 사용 가능 기술들과의 비교

표 1. 사용 가능 기술들과의 비교 표
Table 1. Comparison table with available technologies

	Assign multiple VMs	Add user accounts	Assign multiple containers
OS diversity	High (but limited)	Low (dependent to host OS)	High
Custom images	X	X	O
Resource usage	Max	-	Min
Separation from the Host	O	X	O
Monitoring	Needs external s/w	Needs external s/w	Can import as a template

표 1은 가상머신을 할당하는 것, 리눅스 사용자 계정을 추가하는 것, 본 플랫폼에서 사용한 컨테이너 할당 방식을 비교하는 표이다. 사용하는 이미지는 가상머신의 경우 운영체제 종류에 제한적이지만, 컨테이너를 사용하면 기본적인 Ubuntu, CentOS 뿐만 아니라 경량 타입의 alpine 또는 slim, 특정 프레임워크에 의존적인 FastAPI, Node.js 기반의 이미지도 사용이 가능하며, 시작 프로세스 또한 Dockerfile에서 커스텀 가능하다. 리소스 사용량은 가상머신을 사용할 경우가 최대, 컨테이너 사용이 최소이며, 사용자 계정 추가로 구현할 경우 호스트와 플랫폼 사용자를 분리하는 것이 불가능하다. 그리고 가상머신을 사용하거나 사용자 계정을 추가하는 경우 모니터링 시 외부 s/w가 추가적으로 필요하지만, 컨테이너를 사용하는 경우 모니터링을 위한 여러 오픈소스 프레임워크를 템플릿 기반으로 추가할 수 있다.

2) 유사 플랫폼들과의 비교

표 2. 유사 플랫폼들과의 비교 표
Table 2. Comparison table between similar platforms

	Platform of this paper	classroom.github.io	Programmers	Elice	Snowboard (SMWU)
Self compile possible	O	△ (Only when using Codespace)	O	O	X
Submission	Auto	Manual	Maunal	Manual	Manual
Manage multiple files	O	O	△ (Only when using Vscode)	O	O
File upload needed	X	△ (No need to when using Codespace)	X	X	O
Open source	O	X	X	X	X
License fee	X	O	O	O	O
Resource restrictions	X (Customizable)	O	O	O	O
Customize developing env.	O	X	X	X	X
OS playground support	O	X	X	X	X
IDE support	X	O	O	X	X
Self-host	O	X	X	X	X
External platform dependencies	X	O (GitHub)	X	X	X

표 2는 웹 기반 프로그래밍 과제 제출 기능을 제공하는 다양한 플랫폼들과의 차이점을 나타내는 표이다. 대부분의 플랫폼에서 자체 컴파일이 가능하지만 깃허브 클래스룸의 경우 코드스페이스를 사용해야 개발 환경에서 직접 컴파일이 가능하다. 과제 제출 방식에 있어서 본 플랫폼은 마감 기한이 되면 코드를 작성한 환경에서 자동으로 제출이 가능하지만 그 외 플랫폼들은 직접 업로드를 하거나 수동으로 제출을 해야 한다. 깃허브 클래스룸의 경우에 컴파일 방식과 마찬가지로 코드스페이스를 사용한다면, 마감 기한에 쓰기 권한이 없어지

는 방식으로 본 플랫폼의 제출 방식과 동일하다. 그리고 본 플랫폼은 오픈소스로 구현되어 있어 라이선스 비용이 존재하지 않으며, 수정 및 확장이 가능하다는 이점이 있고, 리소스 제약이 지불 금액에 따라 다른 플랫폼들과 다르게 사용자가 직접 커스텀할 수 있다. 또한 컨테이너를 할당하기 때문에 개발 환경을 운영체제 단에서 커스텀할 수 있으며, 시스템 콜이나 프로세스, 스레드 등 운영체제와 관련된 과제를 수행할 수 있다.

6-2 정성 평가

정성적인 평가를 위하여 프로그래밍 수업에 제출자(학생)로 참여해본 44명과 채점자(조교, 교수)로 참여해본 6명에 대한 설문조사를 실시하였다.

프로그래밍 수업에서 과제를 제출할 때 불편했던 상황에 대한 중복 투표 및 의견을 종합한 결과 운영체제나 운영체제의 버전이 다른 경우가 59.1%로 가장 높았으며, 프로그래밍 언어 버전이 다를 경우와 업로드할 파일이 여러 개인 경우가 각 50%, 31.8%로 뒤를 이었다. 기타 의견으로 이메일로 제출하는 것이 불편하거나 교재와 본인의 IDE가 서로 다른 경우가 있었다.

과제를 채점할 때 가장 불편했던 상황으로는 학생마다 과제를 제출하는 환경이 다를 경우에 전원이 공감하였고, 학생들의 과제를 일일히 열어서 채점해야하는 경우가 83.3%, 과제를 일일히 다운로드해야하는 경우가 66.7%, 채점된 결과를 모아서 볼 수 없는 경우가 50%로 가장 비율이 높았다.

프로그래밍 과제 제출 환경으로 본 논문에서 구현한 플랫폼을 채택하는 것에 대해 전원 3점 이상이라고 답변하였으며, 5점이 50%, 4점이 32%, 3점이 18%를 기록하였다.

채택에 대한 이유로 학생들의 환경을 일괄적으로 통일할 수 있고, 과제를 일일히 다운로드하지 않고 웹 UI로 확인할 수 있다는 점이 가장 많았다. 그리고 어떤 IDE나 언어책을 사용하든 본인의 환경을 고려하여 채점을 해주는 것이므로 학생 입장에서 채점이 잘못될 우려가 없다는 점, 업로드 과정을 거칠 필요가 없어 비효율이나 실수를 줄일 수 있다는 점, 그리고 쿠버네티스 기반으로 운영되어 플랫폼 운영 비용을 절감할 수 있다는 점 등도 있었다.

플랫폼의 가장 큰 이점을 키워드로 투표를 받았을 때, 사용자 편의성이 64%, 운영 노력 절감이 36%였으며, 이점이 없다는 0.0%였다. 그러나 설문에 응답한 학생 중 사용자 편의성이 60%, 채점자 중 운영 노력 절감이 66.7%로, 각 대상에 맞는 결과가 나왔음을 짐작할 수 있다.

단점으로는 GUI가 없어 복잡한 개발을 하기 어렵다는 점, 개발 환경을 만드는 것도 수업의 연장선이기 때문에 환경 통합 플랫폼이 굳이 필요가 없다는 점, 출력 기반이기 때문에 복합적인 과제의 경우 채점이 어려울 것 같다는 점, 터미널 환경에 대한 진입 장벽 등이 있었다.

6-3 평가 종합

플랫폼에 대한 정량적인 평가와 설문조사 내용을 기반으로 한 정성적 평가를 종합적으로 분석해보았을 때, 학생들이 과제를 제출할 때 발생하는 다양한 문제나 실수의 가능성을 줄일 수 있어 사용의 편의성을 입증할 수 있고, 교수 입장에서도 과제 내고, 수렴하고 평가하는 강의 운영의 관점에서 용이함을 알 수 있었다.

대중적으로 사용되는 다른 플랫폼들과 다양한 항목에서 비교해본 결과, 본 플랫폼은 자동 제출이 가능하고 별도의 사용 비용 없이 모든 기능을 제공한다. 직접 호스팅해야하기 때문에 라이선스 비용에 따른 리소스 제약이 없고, 개발 환경 커스텀과 운영체제 기반의 과제 수행도 가능하다는 점 등을 고려해보면, 기존 플랫폼 대비 큰 차별성을 가지는 것을 알 수 있었다.

그리고 사용자가 아닌, 개발자 또는 플랫폼 운영자의 입장에서도 가상머신이나 사용자 계정 할당으로는 구현하기 어렵거나 불가능한 이미지 커스텀, 모니터링, 호스트와의 분리를 할 수 있는 등의 이점이 있다. 또한 쿠버네티스를 사용하는 것 자체 만으로도 운영 및 리소스 비용 절감을 누릴 수 있으며 이식성이 높고, 기존에 존재하는 다양한 쿠버네티스 서드파티 프레임워크를 추가하여 기능을 확장하기 편리하다.

Ⅶ. 결 론

본 논문에서는 가상 머신보다 가볍지만, 독립성을 보장하는 컨테이너를 쉽고 빠르게 배포/확장하고 관리를 자동화해주는 오픈소스인 쿠버네티스와 다양한 오픈소스, 기술들을 활용하여 제출자 별로 상호 독립적인 프로그래밍 환경을 조성한 프로그래밍 과제 제출 플랫폼을 구현하였다.

플랫폼의 사용층을 대상으로 한 설문조사에 따르면, 본 플랫폼을 사용할 경우 학생들이 과제를 제출할 때 발생할 수 있는 여러 문제들의 발생 가능성을 줄이고, 교수가 강의를 운영하는 데 있어서 겪을 다양한 애로사항을 예방할 수 있다는 가능성을 확인할 수 있었다. 유사한 타 플랫폼에 비해서도 자동 제출이나 라이선스 비용, 과제 범위 확장 등의 측면에서 큰 차별성을 가지는 것을 알 수 있었으며, 플랫폼을 운영하는 개발자의 입장에서도 기존 방식들보다 더 다양하고 확장성 높은 커스텀이 가능하다는 이점이 있었다. 쿠버네티스를 도입한 것 자체만으로도 운영 및 리소스 비용 절감을 누릴 수 있었으며, 다양한 쿠버네티스 서드파티 프레임워크를 사용할 수 있다는 장점도 있었다.

그리고 개발에 사용된 모든 소스와 프레임워크, 기술을 오픈소스로만 채택하였으며, 개발 결과물을 깃허브에 공개하여 누구든 재현 가능하고 변형하여 사용할 수 있게 하였다. 따라

서 앞으로 쿠버네티스 기반 플랫폼을 구현하고자할 때, 훌륭한 참조 역할을 할 수 있을 것으로 기대한다.

감사의 글

본 연구는 2023년도 과학기술정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 수행되었습니다.

부록

[1] Project Repository, https://github.com/MoaRoom

[2] System architecture, https://github.com/MoaRoom/Moaroom-Provisioning-Infra/assets/68985625/69a5fac7-455a-4934-b219-127eaab97e88

참고문헌

[1] S. M. Park, "'Coding Education' to Be a Mandatory Subject in Elementary and Middle School from 2025," Dong-A Ilbo, August 2022.

[2] S. H. Lee et al., "Design and Implementation of Web Based Java Virtual Education Center," Korean Institute of Information Scientists and Engineers (KIISE) Academic Presentation Papers 28.1B, pp. 643-645, 2001.

[3] O. S. Lee, K. K. Kwon, and D. H. Shin, "Linux-Based C Programming Practice System Performed on a Web Browser," in *Proceedings of Korean Institute of Information Scientists and Engineers (KIISE) Fall Joint Conference*, pp. 25-33, 1999.

[4] G. Zhu et al., "MySQL Cluster Performance Analysis According to Pod and Sharding Changes in Kubernetes Environment," *The Journal of KING Computing*, Vol. 16, No. 6, pp. 48-55, 2020.

[5] D. D. Vu, M. N. Tran, and Y. H. Kim, "Kubernetes Service Maintenance Architecture Using Machine Learning," *The Journal of Korean Institute of Communications and Information Sciences*, pp. 142-143, February 2021.

[6] Samsung SDS, Dockers and Containers to Respond Quickly to the Changing Business Environment [Internet]. Available: https://www.samsungsds.com/kr/insights/docker_container.html

[7] OpenMaru, Introduction to Container Technology [Internet]. Available: https://www.openmaru.io/Container-Technology-Introduction

[8] Redhat, Kubernetes vs OpenStack [Internet]. Available: https://www.redhat.com/ko/topics/openstack/kubernetes-vs-openstack

[9] Amazon Web Services, The Difference between Containers and Virtual Machines [Internet]. Available: https://aws.amazon.com/ko/compare/the-difference-between-containers-and-virtual-machines/

[10] Samsung SDS, Learn about Kubernetes Part 1 [Internet]. Available: https://www.samsungsds.com/kr/insights/220222_kubernetes1.html

[11] Atlassian, Kubernetes vs Docker [Internet]. Available: https://www.atlassian.com/ko/microservices/microservices-architecture/kubernetes-vs-docker

[12] Kubernetes, Kubernetes Components [Internet]. Available: https://kubernetes.io/ko/docs/concepts/overview/components/

[13] Kubernetes, Cluster Architecture [Internet]. Available: https://kubernetes.io/ko/docs/concepts/architecture/

[14] Amazon Web Services, How to Choose AWS Container Services [Internet]. https://aws.amazon.com/ko/blogs/korea/how-to-choose-aws-container-services/

[15] React, Starting React [Internet]. Available: https://ko.legacy.reactjs.org/

[16] React, Using TypeScript [Internet]. Available: https://react.dev/learn/typescript

[17] Spring, Spring Framework [Internet]. Available: https://spring.io/projects/spring-framework

[18] Spring, Spring Boot [Internet]. Available: https://spring.io/projects/spring-boot

[19] Kubernetes, Pod [Internet]. Available: https://kubernetes.io/ko/docs/concepts/workloads/pods/

[20] Kubernetes, Service [Internet]. Available: https://kubernetes.io/ko/docs/concepts/services-networking/service/

[21] Hashicorp, Terraform [Internet]. Available: https://www.terraform.io/

[22] Terraform, What is Terraform? [Internet]. Available: https://developer.hashicorp.com/terraform/intro

[23] Kubernetes, How to Install Kubernetes Using Kubespray [Internet]. Available: https://kubernetes.io/ko/docs/setup/production-environment/tools/kubespray/

[24] Red Hat, What Is CI/CD (Continuous Integration/ Continuous Delivery) [Internet]. Available: https://www.redhat.com/ko/topics/devops/what-is-ci-cd

[25] Kustomize, Kubernetes Native Configuration Management [Internet]. Available: https://kustomize.io/

[26] Argo CD, What Is Argo CD? [Internet]. Available: https://argo-cd.readthedocs.io/en/stable/

금나연(Nayeon Keum)

2019년～현　재: 숙명여자대학교 IT공학전공/통계학과 학사과정

※관심분야 : 데브옵스/플랫폼 엔지니어링(DevOps/Platform Engineering), 클라우드 엔지니어링(Cloud Engineering), 오픈소스(Opensource), 서버리스(Serverless), 엠엘옵스(MLOps) 등

김민지(Minji Kim)

2019년～현　재: 숙명여자대학교 IT공학전공/통계학과 학사과정

※관심분야 : 웹서버 개발(Web Server Development), 플랫폼 개발(Platform Development), 데이터베이스(Database), 빅데이터(Bigdata), 네트워크(Network) 등

이종우(Jongwoo Lee)

1990년 : 서울대학교 컴퓨터공학과 (학사)

1992년 : 서울대학교 컴퓨터공학과 대학원 (공학석사)

1996년 : 서울대학교 컴퓨터공학과 대학원 (공학박사)

1996년～1998년: 현대전자 정보시스템사업본부 과장

1999년～1999년: 현대정보기술 책임연구원

1999년～2002년: 한림대학교 정보통신공학부 조교수

2002년～2003년: 광운대학교 컴퓨터공학부 조교수

2003년～2004년: 아이닉스 소프트(주) 개발이사

2008년: 뉴욕주립대 스토니브룩 Research Scholar

2012년～2013년: 숙명여자대학교 지식정보처장

2012년～2018년: NAVER 주식회사 사외이사

2014년～2019년: 한국정보과학회 컴퓨팅의 실제 논문지 편집위원장

2020년: 한국정보과학회 총무부회장

2020년～2022년: 숙명여자대학교 공과대학장

2004년～현　재: 숙명여자대학교 인공지능공학부(구 IT공학전공) 교수

2022년～현 재: 숙명여자대학교 소프트웨어중심대학사업단장

※관심분야 : Mobile System Software, Storage Systems, Computational Finance, Cluster Computing, Parallel and Distributed Operating Systems, Embedded System Software