



揚州大學

本科生毕业设计

毕业设计题目 基于 Docker 的在线编程学习平台设计与实现

学生姓名/学号 _____

所在学院 _____

专业及班级 _____

指导教师 _____

校外导师 _____

完成日期 2023 年 5 月 18 日

摘 要

随着互联网和云计算技术的发展, Docker 容器技术已成为应用部署和管理的重要工具。与此同时, 在线编程平台在提高编程教学和学习效率方面发挥着越来越重要的作用。在这样的背景下, 利用 Docker 容器技术构建在线编程平台具有广泛的前景和价值。

为了充分利用 Docker 容器技术提高教学和学习效率, 本毕设研发了一套基于 Docker 容器技术的在线编程平台。具体而言, 采用微服务架构、Vue3 前端框架和 Spring Cloud 后端框架构建了平台的基础架构; 使用 Harbor 私有镜像仓库管理和 Kubernetes 容器编排平台实现了高效的容器部署和管理; 借助 KubeSphere 容器管理平台进行平台本身的管理。本毕设能够为教师和学生提供一个集成化的编程环境, 简化本地开发环境的配置过程, 并为课程材料和学生进展提供便捷的管理系统, 从而提高编程教学和学习的质量和效率。

关键词: 在线编程、实验平台、Docker、Kubernetes、微服务

Abstract

As the Internet and cloud computing technologies continue to advance, Docker container technology has emerged as a vital tool for application deployment and management. Concurrently, online programming platforms are playing an increasingly crucial role in enhancing the efficiency of programming education and learning. In light of this context, developing an online programming platform using Docker container technology holds significant potential and value.

To fully harness the capabilities of Docker container technology in improving teaching and learning efficiency, this thesis presents the development of an online programming platform based on Docker container technology. Specifically, the platform's foundational infrastructure is built using a microservices architecture, Vue3 frontend framework, and Spring Cloud backend framework. Efficient container deployment and management are achieved through the Harbor private image repository management and Kubernetes container orchestration platform. Additionally, the KubeSphere container management platform is utilized to manage the platform itself. This thesis offers an integrated programming environment for teachers and students, streamlining the configuration process of local development environments, and providing a convenient management system for course materials and student progress, ultimately boosting the quality and efficiency of programming education and learning.

Keywords: Online Programming, Experimental Platform, Docker, Kubernetes, Microservices

目 录

摘 要	I
Abstract	II
目 录	III
第 1 章 绪论	1
1.1 背景与意义	1
1.2 目标与内容	1
1.3 编程学习平台发展现状	2
第 2 章 相关技术介绍	4
2.1 Docker 容器技术	4
2.2 Vue 前端框架	5
2.3 Spring Cloud 后端框架	5
2.4 MySQL 和 Redis 数据库	6
2.5 Nacos 服务注册和发现中心	6
2.6 Harbor 私有镜像仓库	7
2.7 Kubernetes 容器编排平台	7
2.8 Fabric8io/kubernetes-client 库	8
2.9 KubeSphere 容器管理平台	8
第 3 章 系统需求分析	9
3.1 系统总体分析	9
3.1.1 系统功能描述	9
3.1.2 系统功能结构	9
3.2 系统功能分析	11
3.2.1 镜像模块	11
3.2.2 实验模块	11
第 4 章 系统概要设计	12
4.1 平台整体架构设计	12
4.2 数据库逻辑结构设计	13
4.3 核心模块设计	14
4.3.1 镜像模块设计	14
4.3.2 实验模块设计	14
第 5 章 系统详细设计及实现	16
5.1 数据库表设计	16
5.2 开发环境	22
5.2.1 硬件环境	22
5.2.2 软件环境	22
5.3 核心功能模块详细设计	23
5.3.1 镜像模块详细设计	23
5.3.2 实验模块详细设计	24
5.4 实现细节	25
5.4.1 harbor 上传设计细节	25
5.4.2 部署实验设计细节	28
第 6 章 平台部署与测试	29

6.1 环境部署	34
6.2 测试目标与测试方法	35
6.2.1 测试目标	35
6.2.2 测试方法	35
6.3 测试结果	35
6.3.1 功能测试结果	35
6.3.2 性能测试结果	41
6.4 测试结果分析	42
第 7 章 总结与展望	43
致 谢	44
参考文献	45
附 录	48

第 1 章 绪论

1.1 背景与意义

在当今数字化时代，教育方式正在发生巨大的变革，远程教育成为新的趋势。然而，传统的编程学习需要安装软件和环境，存在诸多问题，例如需要学生自己配置编程环境，存在不同操作系统的差异，而且如果需要在不同的机器上进行编程，需要将代码和环境一起打包，非常麻烦。这些问题影响了学生的学习效率和学习体验，而在线编程平台则可以解决这些问题。

因此，开发一款基于 Docker 容器技术的在线编程平台，为教师和学生提供一个集中式的学习和教学环境，将是非常有意义的任务。该平台将为教师提供一种简便的方式来管理和评估学生的学习成果，同时也将为学生提供一个灵活的学习环境，使他们能够随时随地学习和实践编程知识。与传统编程学习相比，在线编程平台可以大大降低学习成本和难度，提高学习效率和体验。

因此，本研究旨在开发一个基于 Docker 容器技术的在线编程平台，解决传统编程学习中存在的诸多问题，提高学生的学习体验和效果，为远程教育提供更好的支持和帮助。

1.2 目标与内容

本研究的目标是设计并实现一款面向教师和学生的在线编程学习平台，以解决现有编程学习平台的局限性，并提供更高效、便捷、互动性强且富有趣味性的学习环境，从而为学生带来更优质的学习体验。同时，该平台还将为教师提供更加灵活且个性化的教学工具，有助于提高教学效果和学生的参与度。

研究内容将聚焦于基于 Docker 容器技术的在线编程平台的设计与实现。平台将采用微服务架构，前端使用 Vue3 框架，后端使用 Spring Cloud 框架，借助 Harbor 私有镜像仓库管理以及 Kubernetes 容器编排平台。此外，还将运用 KubeSphere 容器管理平台对整个平台进行管理。

该平台将整合学生端与教师端的功能，使学生可以通过交互式编程环境进行练习，并实时查看自己的实验成绩和进度。教师则可以在平台上创建并发布编程实验，指导学生进行实验操作，并能实时查看学生的实验情况和成绩。

此外，该平台将提供一个镜像市场，方便教师上传和共享自己的实验镜像，以实现资源的复用。同时，平台还将支持用户认证、课程管理、实验管理等功能，使教师和学生能更方便地管理和使用平台资源。

综上所述，本研究旨在开发一款新型在线编程平台，为编程学习者提供基于容器技术的学习与教学方式。这将为编程入门者带来更友好的学习环境，同时为教师提供更灵活的教学工具，从而有助于提升编程教学和学习的质量和效率。

1.3 编程学习平台发展现状

在全球范围内，编程学习平台的种类繁多，各具特色。它们主要可归结为三类：传统课程教学平台、交互式编程环境平台以及虚拟编程环境平台，各自拥有其独特的优点和存在的局限性。

首先，让我们来看看传统课程教学平台。这类平台的代表包括 Coursera、Udemy、Edx、Code Conquest、Codecombat、Pluralsight、网易云课堂和慕课网等。它们主要以视频教程和文字资料的形式，提供各种编程课程以辅助学生进行学习。这类平台的优点在于其丰富的学习资源和高质量的教学内容。例如，一些平台拥有来自全球顶尖大学的课程，学习资源的权威性和专业性得以保证。然而，学习过程中，学生需要自行在本地电脑上配置相应的编程环境，这对于编程新手来说可能造成一定的困扰。尤其是一些复杂的环境配置可能会成为学习的阻碍，影响学生的学习积极性。

其次，交互式编程环境平台，如 LeetCode、Codewars、Codeforces、Hackerrank、Codecombat、Codecademy、Khan Academy 和牛客等，它们通过提供交互式编程环境，使学生在实际操作中学习编程，增强了学习的实践性。这些平台的特色在于它们的"学中做"的教学模式，学生在完成各种级别和类型的编程题目的过程中，实际动手操作，提升了编程技能。然而，这类平台虽然提供了统一的编程环境，但其功能相对较弱，如缺乏智能代码提示等特性，这可能限制了学生的学习深度。在编程学习中，智能代码提示可以帮助学生更快地理解和掌握编程语言的语法和结构，而这一功能的缺失可能会影响学生的学习效率。

最后，我们来看看虚拟编程环境平台，例如实验楼和腾讯云实验室。这类平台通过提供在线 IDE（如 VS Code、Jupyter）或在线 Terminal 等工具，使学生可以在网页上直接进行编程实践。这类平台的优点在于其便捷性，学生无需在本地电脑上配置编程环境，直接在网页上编程，大大简化了学习过程。然而，这些平台通常只提供固定的编程

课程，使用限制较多，且其网站设计并不适应传统教学模式，这可能会影响学生的学习体验和学习效果。例如，一些平台可能没有提供足够的自主学习空间，学生无法根据自己的兴趣和需求选择学习内容，这可能会降低学习的积极性和效果。

总体而言，这几类编程学习平台都有其优点和局限性。在本项目中，希望将传统的教学模式和虚拟编程环境平台相融合，针对扬州大学的编程学习现状，设计一个在线编程平台，为第三类虚拟编程环境平台的一些场景提供一些补充。

第 2 章 相关技术介绍

2.1 Docker 容器技术

Docker 是一种开源的应用容器引擎,可以让开发者将应用程序和依赖环境打包到一个可移植的镜像中,然后在任何支持 Docker 的操作系统上运行。Docker 的优势在于它可以提供快速、可靠和一致的应用部署和交付,无论是在开发、测试还是生产环境中。

Docker 的核心概念是容器 (container),它是一个在主机上运行的沙箱化的进程,与其他进程和系统资源隔离。容器的隔离是通过 Linux 内核的命名空间 (namespace) 和控制组 (cgroup) 来实现的。命名空间可以让容器拥有自己独立的视图,如 PID、网络、用户等。控制组可以限制和控制容器使用的物理资源,如 CPU、内存、磁盘 IO 等。Docker 结构如图 2-1 所示

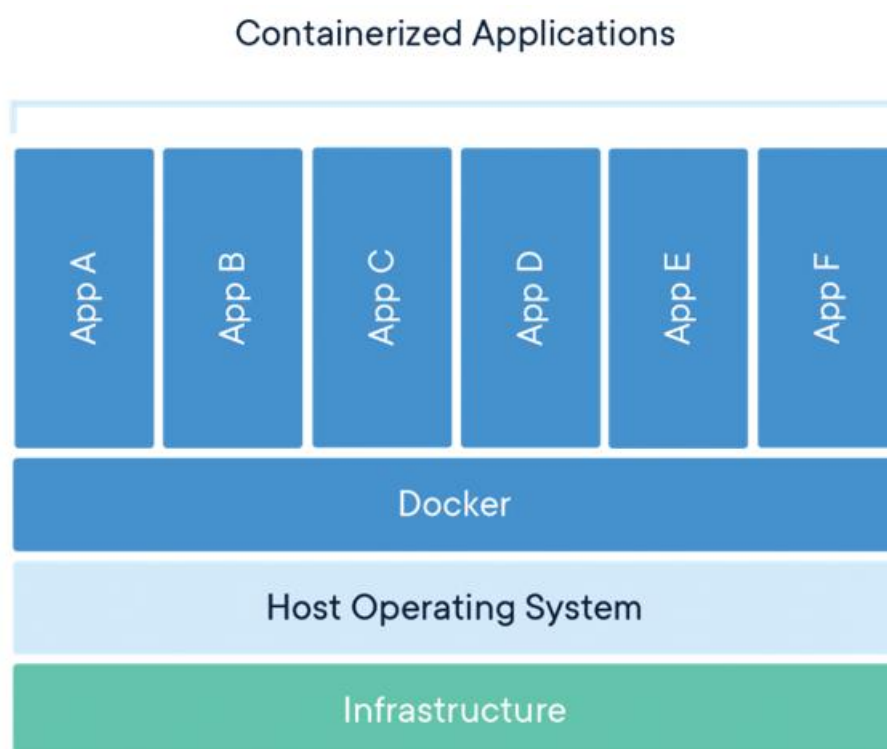


图 2-1 docker 结构图

容器是基于镜像 (image) 来创建的,镜像是一个只读的文件系统,包含了操作系统、应用程序和依赖库等。镜像是分层的,每一层都是一个只读文件系统,可以被多个镜像共享。镜像可以通过 Dockerfile 来编写和构建,也可以从仓库 (registry) 中拉取或推送。仓库是存放镜像的地方,可以是公共的或私有的。

当容器启动时,Docker 会在镜像的最上层加载一个可读写的文件系统,用于保存运

行时的变化。这个文件系统会利用 **copy-on-write** 技术来减少空间占用和性能损耗。当容器停止时，这个文件系统会被删除，而镜像不会受到影响。

2.2 Vue 前端框架

Vue.js 是一种流行的开源 JavaScript 框架，用于构建交互式用户界面。它具有简单易用、灵活可扩展、高效性能等特点，因此在前端开发中得到广泛应用。Vue.js 的核心库只关注视图层，易于与其他库或现有项目集成。Vue.js 还提供了许多有用的工具和插件，例如 Vue Router 和 Vuex，用于处理路由和状态管理。

Vue.js 是由 Evan You 在 2014 年创建的。如图 2-2 所示 Vue.js 基于 MVVM (Model-View-ViewModel) 架构模式，它将用户界面分解成三个部分：模型 (Model)，视图 (View) 和视图模型 (ViewModel)。模型表示数据和业务逻辑，视图表示用户界面，而视图模型充当模型和视图之间的中介人。Vue.js 采用了响应式数据绑定，这意味着当模型发生变化时，视图会自动更新。Vue.js 还支持指令和组件化，使得开发人员可以轻松地创建可重用的代码。

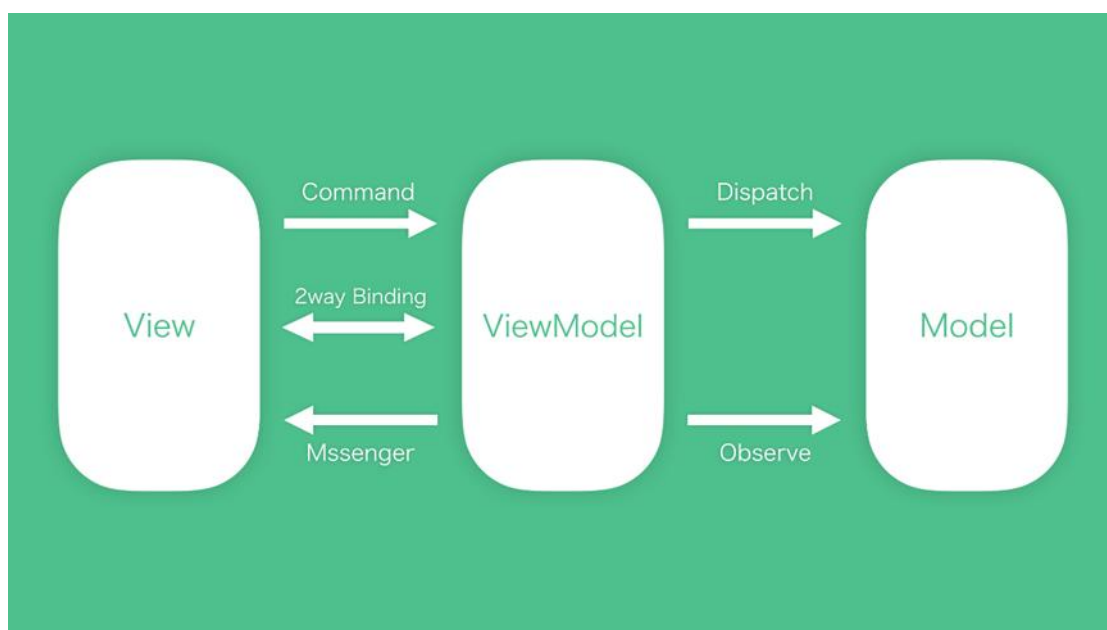


图 2-2 Vue 的 MVVM 模型

2.3 Spring Cloud 后端框架

Spring Cloud 是一个基于 Spring Boot 构建的开源框架，用于构建分布式系统的快速开发工具包。它提供了一组用于快速开发、配置和部署云原生应用程序的工具和服务。Spring Cloud 基于微服务架构，它将应用程序拆分成一组小的、独立的服务，这些服务

可以独立开发、部署和扩展。Spring Cloud 提供了许多有用的工具和插件，例如服务发现、负载均衡、断路器、配置管理等。

Spring Cloud 的核心是 Spring Cloud Netflix，它是一个集成了 Netflix 开源工具包的 Spring Cloud 子项目。Netflix 开源工具包包括 Eureka（服务发现）、Ribbon（客户端负载均衡）、Hystrix（断路器）、Feign（REST 客户端）等。Spring Cloud 还支持其他云原生技术，例如 Spring Cloud Stream 和 Spring Cloud Data Flow，用于构建可扩展的数据流和批处理应用程序。

Spring Cloud 提供了许多有用的功能，例如服务注册与发现、负载均衡、断路器、配置管理、分布式跟踪等。它还提供了集成了 Zipkin 和 Sleuth 的分布式跟踪解决方案，用于跟踪应用程序的请求流程。Spring Cloud 还支持多种部署选项，例如单机部署、Docker 容器部署、Kubernetes 集群部署等。

总之，Spring Cloud 是一个非常强大的分布式系统框架，它为开发人员提供了许多有用的工具和服务，用于构建高可用、可扩展、可维护的云原生应用程序。

2.4 MySQL 和 Redis 数据库

MySQL 是一种关系型数据库管理系统（RDBMS），使用 SQL 作为查询语言。它是一种传统的数据库系统，适用于存储结构化数据，例如表格数据、用户信息等。MySQL 支持 ACID 事务，具有数据一致性和可靠性，是企业级应用程序的首选数据库系统。MySQL 也支持储存过程、触发器等高级特性，可以支持复杂的业务逻辑。MySQL 的性能表现非常出色，具有高并发处理能力和快速响应时间，适合于处理大量的读写请求。

Redis 是一种键值存储数据库，使用键值对作为数据模型，可以存储任意类型的数据。它是一种内存数据库，可以快速地读取和写入数据，适用于缓存和实时数据存储。Redis 也支持持久化功能，可以将数据保存到硬盘中，保证数据的持久性。Redis 还提供了丰富的数据结构，例如字符串、列表、集合、有序集合、哈希表等，可以满足各种应用程序的需求。Redis 具有高速的读取和写入性能，适合于处理大量的数据请求，例如 Web 应用程序中的缓存、计数器、队列等场景。

2.5 Nacos 服务注册和发现中心

Nacos 是阿里巴巴开源的一种动态服务发现、配置管理和服务治理平台。它支持多种服务注册和发现方式，例如 DNS、HTTP 和 RPC 协议。Nacos 具有高可用、可扩展、

易用性等优点，适用于各种云原生应用程序的开发和部署。

Nacos 的架构非常简单，如图 2-3 所示，由一个注册中心和多个服务实例组成。注册中心负责存储服务信息和配置信息，服务实例负责提供服务并将自己的信息注册到注册中心中。Nacos 还提供了 Web 界面和 API 接口，方便开发人员管理和监控服务。

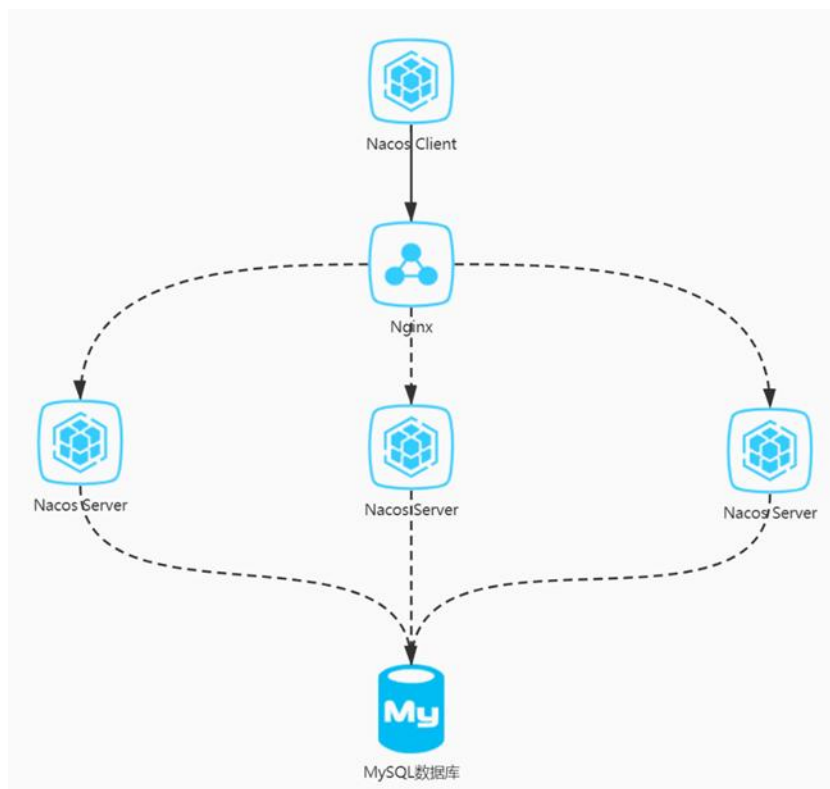


图 2-3 nacos 架构图

2.6 Harbor 私有镜像仓库

Harbor 是一个开源的私有镜像仓库，用于存储、管理和分发 Docker 镜像。它提供了一种安全、可靠、易于管理的方式，用于管理企业级的 Docker 镜像和相关的元数据。Harbor 支持多租户、多仓库、多镜像管理，可以满足不同团队和项目的需求。Harbor 还提供了 LDAP/AD 集成、高可用、复制、漏洞扫描等高级特性，可以保证镜像的安全性和可用性。

2.7 Kubernetes 容器编排平台

Kubernetes 是一种开源的容器编排平台，用于自动化部署、扩展和管理容器化应用程序。它提供了一种可移植、可扩展、自我修复、自动化的平台，用于管理容器化应用程序和服务。Kubernetes 是 Google 在 2014 年发布的，现已成为云原生计算基金会 (CNCF) 的毕业项目。

如图 2-4 所示, Kubernetes 的核心组件主要包含 Master 节点、Node 节点、Pod、Service 和 Volume。作为 Kubernetes 的控制节点, Master 节点负责监控和管理整个集群的状态和资源, 它主要包括 API 服务器、etcd (一种分布式键值存储系统)、控制器管理器以及调度器。相较之下, Node 节点则是 Kubernetes 的工作节点, 其主要职责是运行容器和服务, 涵盖了 Kubelet (用于管理 Pod)、容器运行时 (如 Docker、CRI-O 等)、以及 kube-proxy (网络代理) 等组件。Pod 在 Kubernetes 中被定义为最小的部署单元, 它包含一个或多个容器, 并共享网络和存储资源, Pod 还支持水平扩展, 以及自动化调度和滚动更新。Service 则是 Kubernetes 的服务发现和负载均衡机制, 主要负责将请求路由到 Pod 中的容器。而 Volume 是 Kubernetes 的存储抽象层, 用以管理容器的持久化存储需求。

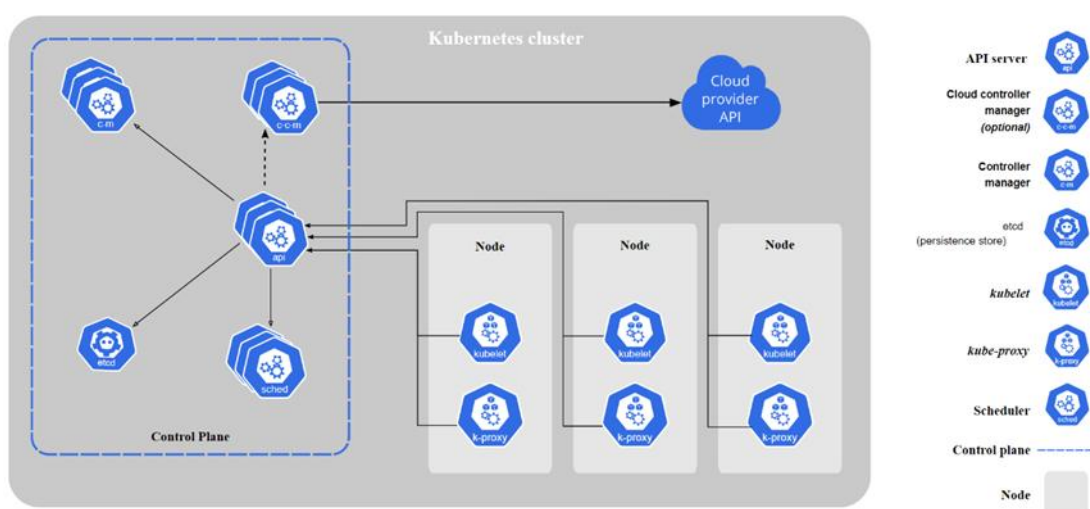


图 2-4 Kubernetes 架构图

2.8 Fabric8io/kubernetes-client 库

Fabric8io/kubernetes-client 是一个开源的 Java 客户端, 用于访问和管理 Kubernetes API。它提供了一个简单、可靠、易于使用的方式, 用于编写 Kubernetes 应用程序和自动化工具。Fabric8io/kubernetes-client 具有丰富的 API 接口、强大的类型安全、自动重试和故障转移等特性, 可以方便地与 Kubernetes API 交互。

2.9 KubeSphere 容器管理平台

KubeSphere 是一个开源的、分布式的、多租户的企业级平台, 它是基于 Kubernetes 的容器平台, 提供简单易用的操作界面以及向导, 帮助企业进行 Kubernetes 资源管理、DevOps 工程、应用生命周期管理、微服务治理、多集群管理以及多租户管理。

第3章 系统需求分析

3.1 系统总体分析

3.1.1 系统功能描述

本系统旨在设计一个基于 Docker 容器技术的在线编程学习平台，以便提供更高效、便捷和交互性强的编程学习体验，解决传统编程学习中的问题，并为远程教育提供支持和帮助。系统需要实现上传镜像，配置实验环境，学生参加实验，老师查看实验，批阅成绩等基本功能。

3.1.2 系统功能结构

系统功能结构如图 3-1 所示，平台有四类用户，分别为注册用户、学生、老师、管理员。

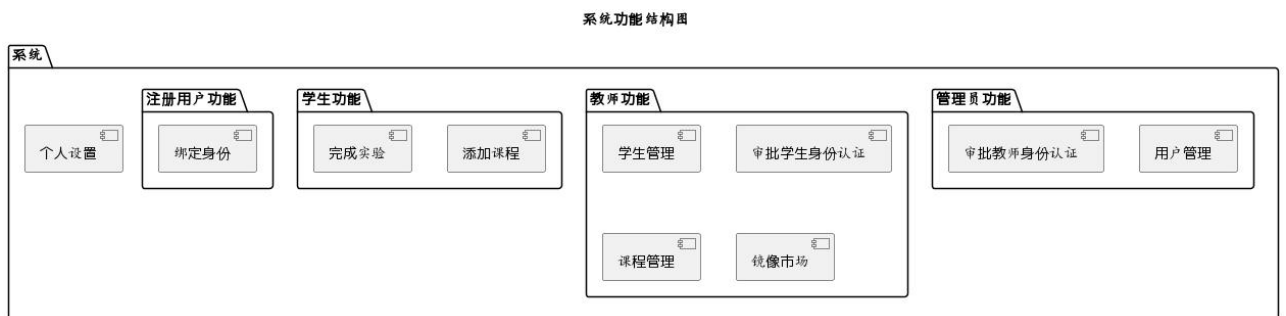


图 3.1 系统功能结构

对于管理员，如图 3-2 所示，系统提供了个人信息设置功能，允许管理员修改其昵称、密码等个人信息。同时，管理员可以进行用户管理，包括新增、修改、删除和查询用户信息，方便进行日常管理工作。此外，系统还赋予了管理员审批教师身份认证的权力，当注册用户申请成为教师时，管理员可以进行审核和认证。

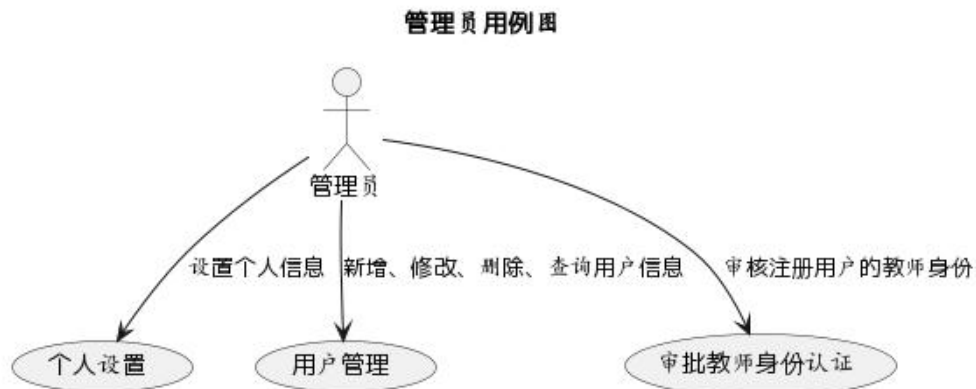


图 3-2 管理员用例图

对于教师，如图 3-3 所示，系统同样提供了个人信息设置功能，允许教师修改其昵称、密码等个人信息。教师可以在其课程中添加或删除学生，同时也可以管理学生的课程学习情况。当用户申请成为所在学院的学生时，教师有权审核批准。此外，教师还可以进行课程管理，包括新增、修改、删除课程以及管理课程下的实验。镜像市场功能则允许教师上传镜像到镜像市场或使用镜像市场中的镜像部署实验环境。

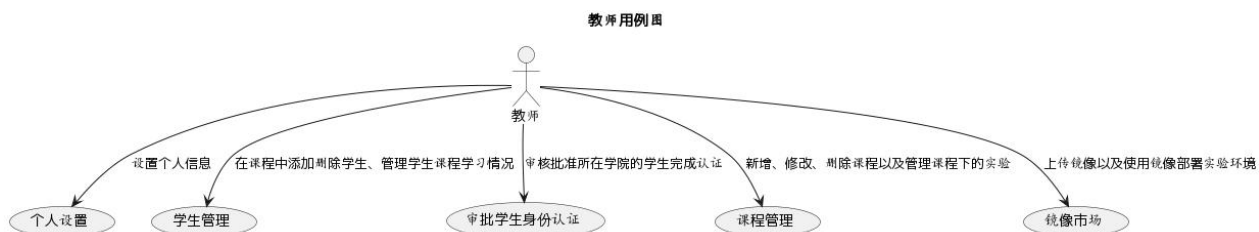


图 3-3 教师用例图

对于学生，如图 3-4 所示，系统提供了个人信息设置功能，允许学生修改其昵称、密码等个人信息。学生可以选择添加老师的课程，并在已添加的课程中完成实验。

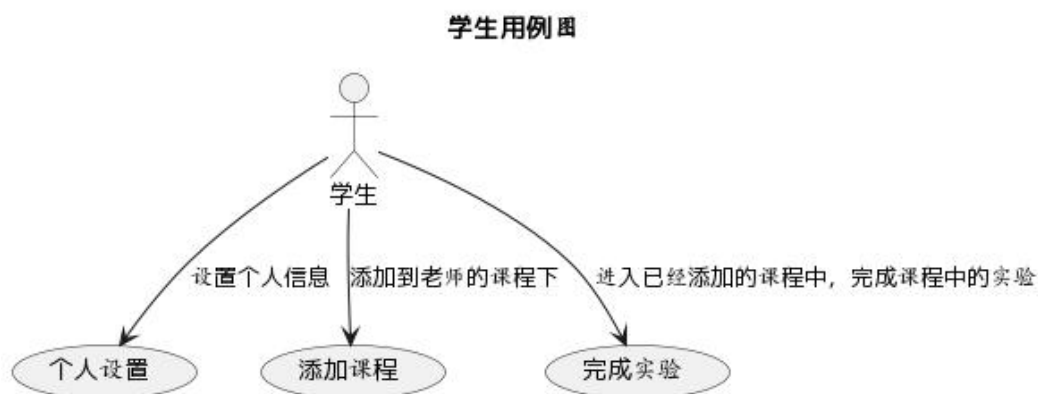


图 3-4 学生用例图

对于注册用户，如图 3-5 所示，系统提供了绑定身份功能，用户在注册后需要进行身份认证，可以选择认证成为学生或教师。

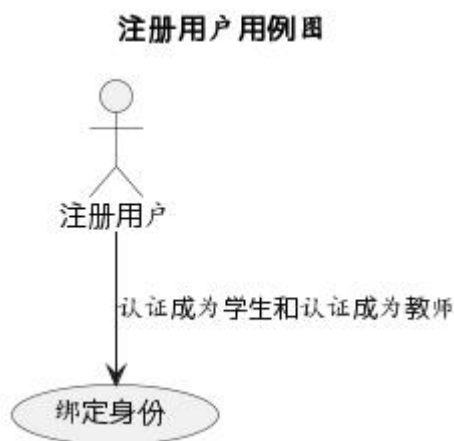


图 3-5 注册用户用例图

3.2 系统功能分析

3.2.1 镜像模块

镜像模块是平台的核心部分之一，它需要提供以下功能：

镜像上传：支持教师上传 Dockerfile、镜像包和容器包三种类型的镜像。

镜像查看：允许用户查看镜像市场中的镜像列表、镜像详细信息、下载次数等。

镜像修改：教师可以对自己上传的镜像进行修改，包括镜像名称、描述、标签等信息。

镜像删除：教师可以删除自己上传的镜像，以便维护镜像市场的整洁。

3.2.2 实验模块

实验模块是另一个重要的功能模块，需要满足以下功能需求：

实验创建：教师可以创建新的实验，填写实验名称、描述、目标、预计完成时间等信息，并上传实验所需的镜像。

实验管理：教师可以查看、编辑和删除已创建的实验，以及查看学生参与实验的情况。

实验测试：教师可以预览实验环境，确保实验能够正常运行。

学生实验：学生可以查看实验列表、进入实验环境、开启实验和销毁实验镜像。

实验评分：教师可以查看学生的实验情况、给学生实验点评和打分，以便了解学生的学习进度和成果。

第 4 章 系统概要设计

4.1 平台整体架构设计

整个系统基于 k8s 容器编排技术构建，整个项目结构如图 4-1 所示。本项目运行在 codeonline-projects 和 codeonline-all 两个 namespace 中。

在 codeonline-projects 运行了整个开发平台除了 NFS 的部分，包括前端，后端各个模块以及 mysql, redis, harbor, nacos 等。而 NFS 直接部署在 Centos 服务器上而在 codeonline-all 这个 namespace 中，运行所有的实验容器。

系统启动时，各个模块会向 Nacos 注册并获取自身配置。前端请求首先经过 gateway 鉴权过滤，然后路由到相应的后端模块进行处理，最后返回前端。后端的模块主要包含三个部分，一个部分是若依的模块。第二个部分是业务部分，处理各种业务逻辑。第三个部分是关于 k8s 和 harbor 等处理模块，当中整合了 k8s, harbor 的一些 api，用来处理 k8s, harbor 等操作请求。例如当前端上传镜像时会把镜像 push 到 harbor 中，需要查看是可以通过 api 查看镜像信息，以及 pull 镜像。又如当学生开启实验时，后端会根据老师提供的配置文件在 k8s 的 codeonline-all 这个 namespace 下部署一个实验容器，给学生实验。而实验过程中，学生的实验文件将会被挂载到 NFS 服务器中。

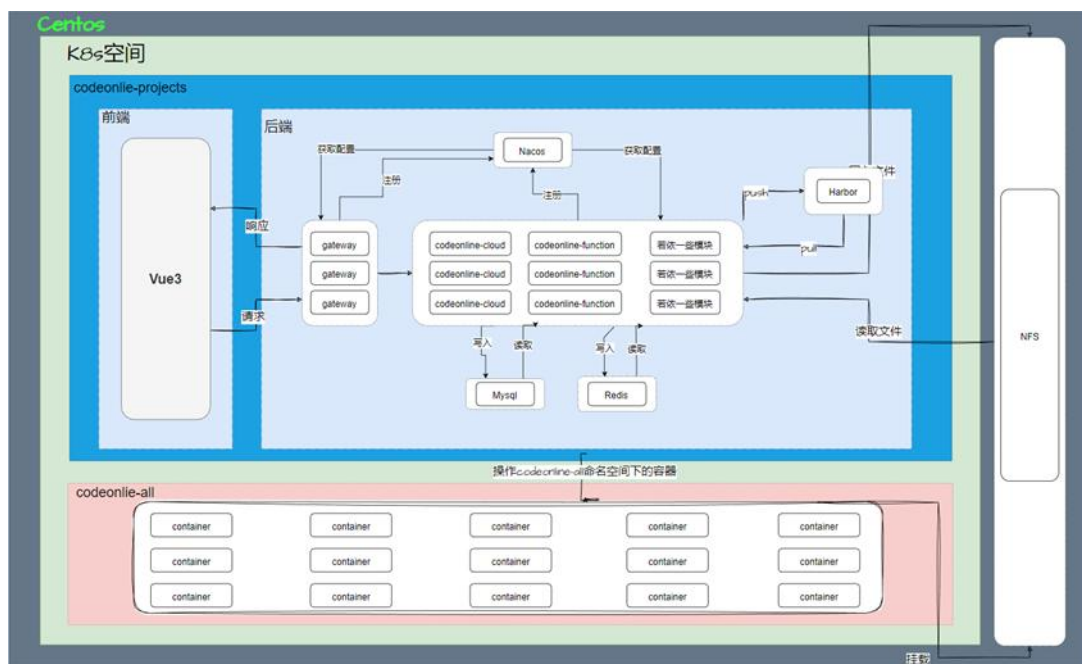


图 4-1 项目架构图

4.2 数据库逻辑结构设计

这里使用 ER 图来描述关键部分的表结构。数据库中表名和描述如表 4-1 所示。

表 4-1 数据库表名和描述

表名	描述
business_course	课程表，记录课程相关信息
business_lab	实验信息表，用于记录实验信息
business_user_and_course_relation	用户课程关联表，用于关联学生和课程
business_user_lab_score	用户实验关联表，用于关联用户和实验，记录实验成绩和评价
course_lab_file	实验文件表，用于记录实验上传的实验资料
harbor_upload	harbor 镜像信息表，用于记录上传的镜像信息
k8s_configure	k8s 配置表，记录 k8s 配置信息
k8s_configure_relation	k8s 配置关系表，关联实验和配置信息
k8s_user_and_deploy_relation	k8s 用户部署关系表，记录用户部署的容器信息

课程表和实验信息表之间是一对多的关系，如图 4-2 所示，一个课程可以有多个实验。课程表和用户之间是一对多的关系，通过用户课程关联表关联。实验信息表 and 用户实验关联表之间是一对多的关系。每个实验下都会有多个学生成绩。实验信息表和实验文件表之间是一对一的关系。每个实验有一个实验资料。实验表通过 k8s 配置关系表和 k8s 配置表之间是一对一的关系。每个实验有一个对应的 k8s 配置文件。

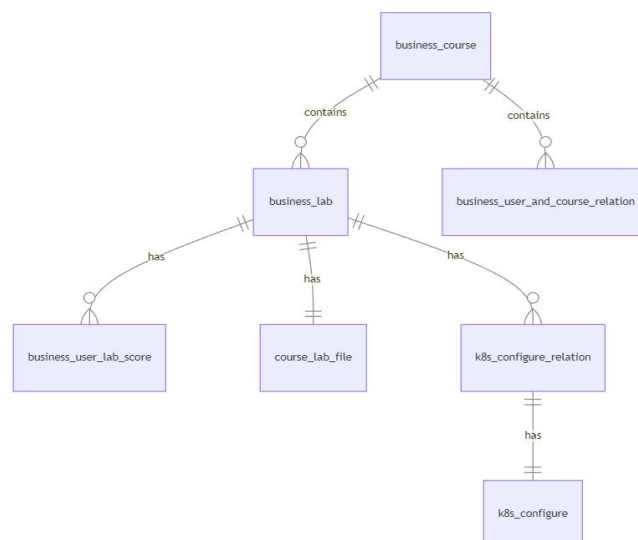


图 4-2 ER 图

4.3 核心模块设计

4.3.1 镜像模块设计

如图 4-3 所示，镜像模块主要有两个部分：镜像上传和镜像查看。在镜像市场页面，用户能浏览自己上传的镜像以及所有公开的镜像。点击上传按钮，用户可以上传自己的镜像，上传时需填写镜像名、标签、类型等信息。系统会根据上传类型（dockerfile、镜像包、容器包）进行相应的处理，处理结果会被写入 redis，前端通过轮询查询结果。

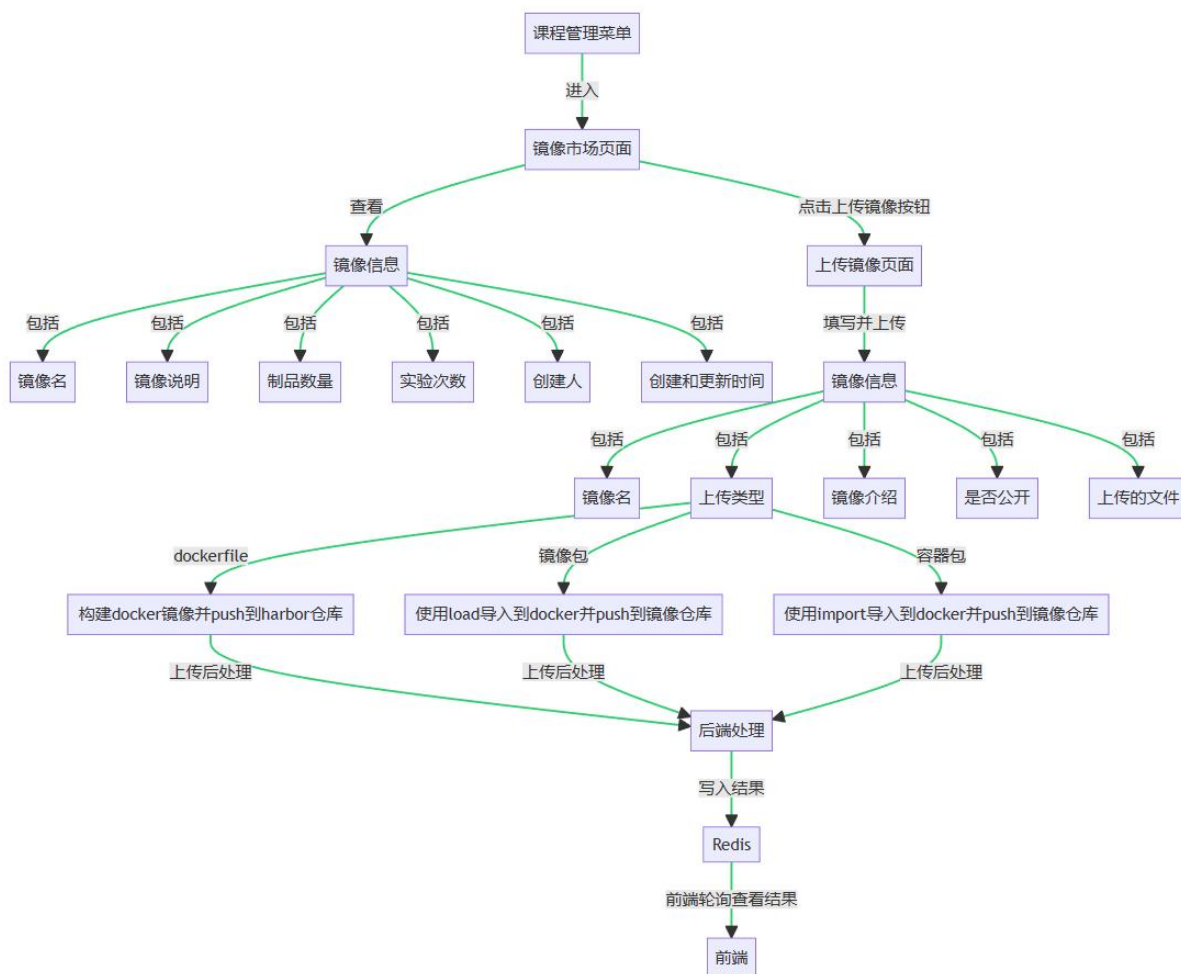


图 4-3 镜像模块设计图

4.3.2 实验模块设计

如图 4-4 所示，实验模块由新建实验、测试实验和管理实验三个功能组成。教师在创建课程后，可以添加新的实验，填写包括实验名称、时间、内容和 k8s 配置等信息。完成添加后，新的实验将在实验管理界面显示，教师可对实验进行修改、删除、测试和管理等操作。学生端则可在课程页面查看并进入实验，进行实验操作。

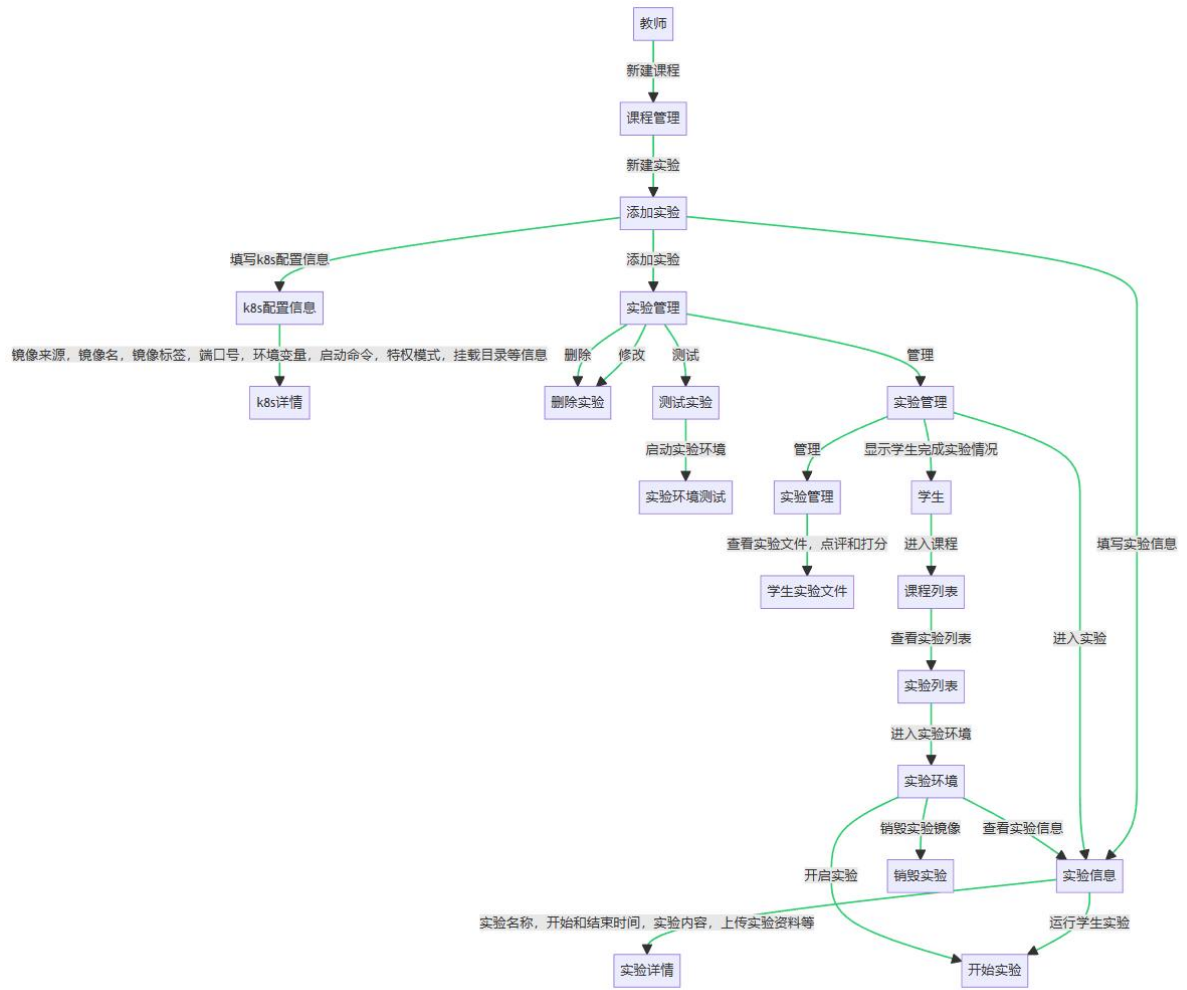


图 4-4 实验模块设计图

第 5 章 系统详细设计及实现

5.1 数据库表设计

下面是一些关键部分的表结构：

business_affair：如表 5-1 所示，**business_affair** 是一张统一的事务表，以统一的格式记录各种事务，如在学生/老师注册后需要分别向该学院老师/管理员进行认证，学生/老师填写信息会提交到 **business_identity** 表中，再提取相关信息提交到 **business_affair**，以供查看。

表 5-1 统一事务表

列名	数据类型	是否为主键	注释
affair_id	bigint(20)	√	事务 id
handle_people_id	bigint(20)		处理人
affair_people_id	bigint(20)		事务发起人 id
affair_people_name	varchar(32)		事务发起人昵称
affair_type	varchar(32)		事务类型
content	varchar(255)		事务内容
affair_status	varchar(32)		事务状态
create_by	varchar(64)		创建者
create_time	datetime		创建时间
update_by	varchar(64)		更新者
update_time	datetime		更新时间
remark	varchar(255)		备注
access_id	bigint(20)		关联具体事务的 id
reply	varchar(255)		审核回复

business_course：如表 5-2 所示，**business_course** 是课程表，记录课程相关信息。

表 5-2 课程表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	课程 id
teacher_id	bigint(20)		教师 id
teacher_name	varchar(32)		教师名
course_name	varchar(64)		课程名
course_status	varchar(32)		课程状态(未开始,进行中,已结束)
introduction	mediumtext		课程简介
course_start_time	datetime		开始时间
course_end_time	datetime		结束时间
course_week	int(11)		课程周数
create_by	varchar(64)		创建者
create_time	datetime		创建时间
update_by	varchar(64)		更新者
update_time	datetime		更新时间

business_identity: 如表 5-3 所示, business_identity 是身份认证表, 学生/老师注册后需要分别向该学院老师/管理员进行认证, 学生/老师填写信息会提交到 business_identity。

表 5-3 身份认证表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	
user_id	bigint(20)		注册用户
identity_id	bigint(20)		注册的身份(100 是老师, 101 是学生)
university_id	bigint(20)		大学 id
college_id	bigint(20)		学院 id
class_id	bigint(20)		班级 id

university_name	varchar(64)		大学名
college_name	varchar(64)		学院名
class_name	varchar(64)		班级名
approver_id	bigint(20)		审批人 id
create_by	varchar(64)		创建者
create_time	datetime		创建时间
update_by	varchar(64)		更新者
update_time	datetime		更新时间
remark	varchar(255)		备注
has_finished	tinyint(4)		是否完成
status	varchar(32)		目前状态

business_lab: 如表 5-4 所示, 实验信息表, 用于记录实验信息。

表 5-4 实验信息表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	id
lab_id	bigint(32)		实验 id
course_id	bigint(20)		课程 id
lab_name	varchar(255)		实验名称
lab_content	mediumtext		实验内容
lab_start_time	datetime		实验开始时间
lab_end_time	datetime		实验结束时间
create_by	varchar(64)		创建者
create_time	datetime		创建时间
update_by	varchar(64)		更新者
update_time	datetime		更新时间

business_user_and_course_relation: 如表 5-5 所示, 用户课程关联表, 用于关联学生

和课程。

表 5-5 用户课程关联表

列名	数据类型	是否为主键	注释
relation_id	bigint(20)	√	关联 id
user_id	bigint(20)		学生 id
course_id	bigint(20)		课程 id
create_by	varchar(64)		创建者
create_time	datetime		创建时间
update_by	varchar(64)		更新者
update_time	datetime		更新时间
remark	varchar(500)		备注

business_user_lab_score: 如表 5-6 所示, 用户实验关联表, 用于关联用户和实验, 记录实验成绩和评价。

表 5-6 用户实验关联表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	
course_id	bigint(20)		
student_id	bigint(20)		
teacher_id	bigint(20)		
lab_id	bigint(20)		
score	int(11)		
comment	text		

course_lab_file: 如表 5-7 所示, 实验文件表, 用于记录实验上传的实验资料。

表 5-7 实验文件表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	主键
course_id	bigint(20)		课程 id
lab_id	bigint(20)		实验 id

teacher_id	int(11)		教师 id
file_url	varchar(255)		文件 URL
file_name	varchar(255)		文件名

harbor_upload: 如表 5-8 所示, harbor 镜像信息表, 用于记录上传的镜像信息。

表 5-8 镜像信息表

列名	数据类型	是否为主键	注释
upload_id	int	√	上传镜像 id
user_id	int		所属用户
image_name	varchar		镜像名
image_tag	varchar		镜像版本
image_url	varchar		镜像路径
is_public	tinyint		是否公开

k8s_configure: 如表 5-9 所示, k8s 配置表, 记录 k8s 配置信息。

表 5-9 k8s 配置表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	
configure	mediumtext		配置

k8s_configure_relation: 如表 5-10 所示, k8s 配置关系表, 关联实验和配置信息。

表 5-10 k8s 配置关系表

列名	数据类型	是否为主键	注释
id	bigint(20)	√	主键
user_id	bigint(20)		创建人 id
lab_id	varchar(32)		实验 id
configure_id	bigint(20)		配置文件 id
has_public	tinyint(4)		是否公开

k8s_user_and_deploy_relation: 如表 5-11 所示, k8s 用户部署关系表, 记录用户部署的容器信息。

表 5-11 k8s 用户部署关系表

列名	数据类型	是否为主键	注释
id	bigint	√	
user_id	bigint		用户 id
teacher_id	bigint		教师 id
lab_id	varchar		实验 id
deploy_namespace	varchar		deploy 命名空间
deployment_name	varchar		deploy 名称
service_name	varchar		service 名称
destroy_time	datetime		
has_destroy	tinyint		
create_by	varchar		创建者
create_time	datetime		创建时间
update_by	varchar		更新者
update_time	datetime		更新时间

sys_user: 如表 5-12 所示, 用户表。

表 5-12 用户表

列名	数据类型	是否为主键	注释
user_id	bigint	√	用户 ID
dept_id	bigint		部门 ID
user_name	varchar		用户账号
nick_name	varchar		用户昵称
user_type	varchar		用户类型 (00 系统用户)
email	varchar		用户邮箱
phonenumber	varchar		手机号码

sex	char		用户性别（0 男 1 女 2 未知）
avatar	varchar		头像地址
password	varchar		密码
status	char		帐号状态（0 正常 1 停用）
del_flag	char		删除标志（0 代表存在 2 代表删除）
login_ip	varchar		最后登录 IP
login_date	datetime		最后登录时间
create_by	varchar		创建者
create_time	datetime		创建时间
update_by	varchar		更新者
update_time	datetime		更新时间
remark	varchar		备注

5.2 开发环境

5.2.1 硬件环境

表 5-13 硬件环境表

硬件类型	描述
内存	64G
CPU	13600K
硬盘	2T

5.2.2 软件环境

表 5-14 软件环境表

类型	技术	版本	用途
前端	Vue3	-	构建前端应用
前端	JavaScript	ES6+	编写前端应用逻辑

后端	Java	JDK8	编写后端应用逻辑
后端	Spring Cloud	-	框架支持微服务架构
数据库	MySQL	5.6	存储应用数据
数据库	Redis	-	存储应用缓存数据
镜像	Docker	-	打包、发布应用镜像
镜像	Harbor	-	私有镜像仓库管理
容器	Kubernetes	1.22.12	容器编排和管理
容器	KubeSphere	3.3.1	容器管理平台
通讯	nacos	2.1.2	服务注册和发现中心
库	Fabric8io/kubernetes-client	4.13.2	用于编写 Kubernetes 操作的 Java 库
系统	CentOS	7.0+	系统基础环境
存储	NFS	-	共享存储
服务	nginx	1.18+	反向代理和负载均衡服务
工具	Node.js	16.19.0	辅助前端构建和工具运行
IDE	Idea	2022.3.1	编写 java 后端的 IDE
IDE	WebStorm	2022.3.1	编写 vue 前端的 IDE

5.3 核心功能模块详细设计

5.3.1 镜像模块详细设计

镜像模块主要包括镜像上传和镜像查看两个部分。

在课程管理菜单下有一个镜像市场页面。在此页面中可以看到自己上传的镜像和所有公开的镜像，每张镜像卡片可以查看镜像信息，如镜像名，镜像说明，制品数量，实

验次数，创建人，创建和更新时间等信息。教师可以通过这些信息来创建自己的实验环境。

在镜像市场中，点击上传镜像按钮，可以打开上传镜像页面，在此次可以填写镜像信息并上传镜像，需要填写的信息有：镜像名、镜像标签、上传类型、镜像介绍、是否公开、上传的文件等。其中上传类型有三种：**dockerfile**、镜像包、容器包。对于三种类型后端会做不同的处理，对于 **dockerfile**，会先构建 **docker** 镜像再 **push** 到 **harbor** 仓库中，而对于镜像包和容器包则是分别使用 **load** 和 **import** 的方式导入到 **docker** 中再 **push** 到镜像仓库。当用户上传后，后端会根据类型不同来处理上传流程，完成后会将结果写入到 **redis** 中，前端则是通过轮询的方式不断的查看 **redis** 中是否写入了结果，查询到后把结果信息返回给用户，这里使用轮询的方式是因为后端处理过程耗时较长，尤其是当使用 **dockerfile** 上传时。

5.3.2 实验模块详细设计

对于教师端来说有新建实验，测试实验，管理实验等几个部分。

当老师新建一个课程后，就可以在课程管理中去新建实验了。添加实验需要填写两部分信息，实验信息和 **k8s** 配置信息。实验信息包括实验名称，开始和结束时间，实验内容，上传实验资料等。**k8s** 配置信息包括：镜像来源（**harbor** 或者 **dockerhub**），镜像名，镜像标签，端口号，环境变量，启动命令，特权模式，挂载目录等信息。完成这些信息确定后就可以添加此次实验。

当课程添加完成后，实验管理界面将会多一个实验信息，其中有四个按钮：修改、删除、测试、管理。点击测试按钮将会进入到实验环境界面，在此界面中老师可以启动一个实验环境，以便测试配置是否正常。

点击管理按钮后将进入对应实验的管理页，在此管理页会显示课程下所有学生完成本次实验的情况如学生 **ID**、学生姓名、开始实验时间、结束实验时间、实验时长、实验状态成绩等，并且有两个操作按钮：进入实验、管理。

点击进入实验将会进入到学生的实验环境中，在此处可以运行学生的实验以检查学生的实验完成情况。而点击管理，则可以进入管理页，此处可以看见学生的实验文件并且对本次学生实验完成情况进行点评和打分。

对于学生端来说，学生进入改课程下后可以看见实验列表，点击进入后到达实验环境页面，在此页面中可以查看实验信息，开启实验，以及在实验完成后销毁实验镜像。

5.4 实现细节

5.4.1 harbor 上传设计细节

镜像上传允许使用三种方式上传镜像，分别是 `dockerfile`、镜像包、容器包。`dockerfile` 是用于构建镜像的文件，镜像包是 `dockerfile` 构建而成或者使用 `docker save` 命令导出，而容器包是 `docker export` 命令导出。导出的镜像包只包含静态文件和配置信息，不包含容器中运行的状态和数据；而导出的容器文件系统包含了容器中所有的文件和状态信息，但是不包含容器的配置信息和元数据。

在前端，用户需要前些镜像名，镜像标签等信息，并且选择对应的模式上传镜像。后端会存储上传文件，并根据不同的模式进行相应的处理。对于 `dockerfile`，会先构建成镜像。对于镜像包会使用 `docker load` 导入，而容器包会使用 `docker import` 导入。接着，将会给镜像打上标签，格式为 `harbor 仓库_用户名_镜像名:标签名`，接着就可以 `push` 到镜像仓库了。`push` 完成后，会去读取 `harbor` 中是否存在该镜像，以确定是否成功，当确认成功后，先将镜像信息写道数据库中，接着将成功信息写道 `redis` 中。而在前端会通过轮询的方式，请求后端是否成功，知道获取到 `redis` 中的信息。因为后端上传镜像过程耗时场，所以设计成异步的形式。

下面是上传部分核心代码展示：

```
1.  @SneakyThrows
2.  @Async
3.  @Override
4.  public void dockerfileToImageAndPush(HarborUpload harborUpload, String harborKey) {
5.      String url = harborUpload.getImageUrl();
6.      String[] split = url.split("http://.*?statics/");
7.      String path = split[1];
8.      path = localFilePath + path;
9.      //docker build
10.     String dockerfileName = harborUpload.getImageName() + ":" + harborUpload.getImageTag();
11.     String buildCommand = "docker build -f " + path + " -t " + dockerfileName + ".";
12.     try {
13.         String buildResult = shellMan.exec(buildCommand);
14.         System.out.println(buildResult);
15.     } catch (IOException e) {
16.         throw new HarborShellException("docker build error: " + e.getMessage());
17.     }
18.     //docker push
19.     pushImage(path, dockerfileName,dockerfileName);
20.     if (hasPushed(dockerfileName)) {
```

```

21.     harborUploadMapper.insertHarborUpload(harborUpload);
22.     redisService.setCacheObject(harborKey, "success", 10l, TimeUnit.MINUTES);
23.
24.     }else {
25.         redisService.setCacheObject(harborKey, "fail", 10l, TimeUnit.MINUTES);
26.     }
27.
28.
29.     }
30.
31.     @SneakyThrows
32.     @Async
33.     @Override
34.     public void loadImageAndPush(HarborUpload harborUpload, String harborKey) {        String url = harborUplo
ad.getImageUrl();
35.         String[] split = url.split("http://.*?statics/");
36.         String path = split[1];
37.         path = localFilePath + path;
38.
39.         log.info(path);
40.
41.         String sourceImageName;
42.         String targetImageName= harborUpload.getImageName() + ":" + harborUpload.getImageTag();
43.
44.         log.info(targetImageName);
45.         String loadCMD = "docker load -i " + path;
46.         try {
47.             sourceImageName = shellMan.exec(loadCMD);//Loaded image: base-centos:1.0.0
48.             sourceImageName = sourceImageName.replace("Loaded image: ", "").replace("\n", "");
49.         } catch (IOException e) {
50.             throw new HarborShellException(String.format("fail: %s", loadCMD, e.getMessage()));
51.         }
52.
53.         pushImage(path, sourceImageName,targetImageName);
54.         if (hasPushed(targetImageName)) {
55.
56.             harborUploadMapper.insertHarborUpload(harborUpload);
57.             redisService.setCacheObject(harborKey, "success", 10l, TimeUnit.MINUTES);
58.         }else{
59.             redisService.setCacheObject(harborKey, "fail", 10l, TimeUnit.MINUTES);
60.         }
61.
62.     }
63.
64.     @Override
65.     public void importImageAndPush(HarborUpload harborUpload, String harborKey) {

```

```

66.
67.     String url = harborUpload.getImageUrl();
68.     log.warn(url);
69.     String[] split = url.split("http://.*?statics/");
70.     String path = split[1];
71.     path = localFilePath + path;
72.     log.info(path);
73.     String sourceImageName = harborUpload.getImageName() + ":" + harborUpload.getImageTag();
74.     String importCMD = "docker import " + path + " " + sourceImageName;
75.     log.info(importCMD);
76.     try {
77.         shellMan.exec(importCMD);//Loaded image: base-centos:1.0.0
78.     } catch (IOException e) {
79.         throw new HarborShellException(String.format("加载镜像包失败，命令为:%s，报错信息: %s", importCMD, e.getMessage()));
80.     }
81.     pushImage(path, sourceImageName,sourceImageName);
82.     if (hasPushed(sourceImageName)) {
83.         log.info("推送成功");
84.         harborUploadMapper.insertHarborUpload(harborUpload);
85.         redisService.setCacheObject(harborKey, "上传成功", 10l, TimeUnit.MINUTES);
86.     } else {
87.         redisService.setCacheObject(harborKey, "上传失败", 10l, TimeUnit.MINUTES);
88.     }
89. }
90.
91.
92. public void pushImage(String path, String sourceImageName,String targetImageName) {
93.     String tagCMD = String.format("docker tag %s %s/%s/%s", sourceImageName, harborUrl, harborSpace, targetImageName);
94.     log.info(tagCMD);
95.     try {
96.         shellMan.exec(tagCMD);
97.     } catch (IOException e) {
98.     }
99.     String loginCMD = String.format("docker login %s -u %s -p %s", harborUrl, harborName, harborPassword);
100.    try {
101.        String tagResult = shellMan.exec(loginCMD);//Login Succeeded
102.        if (!tagResult.contains("Login Succeeded")) {
103.            throw new HarborShellException(String.format("fail", tagResult));
104.        }
105.    } catch (IOException e) {
106.        throw new HarborShellException(String.format("fail %s", e.getMessage()));
107.    }
108.    String pushCMD = String.format("docker push %s/%s/%s", harborUrl, harborSpace, targetImageName);
109.    log.info(pushCMD);

```



```

110.     try {
111.         shellMan.exec(pushCMD);
112.     } catch (IOException e) {
113.         throw new HarborShellException(String.format("fail %s", pushCMD, e.getMessage()));
114.     }
115.     String deleteCMD1 = String.format("docker rmi %s", sourceImageName);
116.     String deleteCMD2 = String.format("docker rmi %s/%s/%s", harborUrl, harborSpace, targetImageName);
117.
118.     try {
119.         shellMan.exec(deleteCMD1);
120.         shellMan.exec(deleteCMD2);
121.     } catch (IOException e) {
122.         throw new HarborShellException(String.format("fail %s", deleteCMD1, deleteCMD2, e.getMessage()));
123.     }
124.     String deleteTagCMD = String.format("docker rmi %s/%s/%s", harborUrl, harborSpace, sourceImageName);
125.     try {
126.         shellMan.exec(deleteTagCMD);
127.     } catch (IOException e) {
128.         throw new HarborShellException(String.format("fail %s", deleteTagCMD, e.getMessage()));
129.     }
130.     File file = new File(path);
131.     if (file.exists()) {
132.         file.delete();
133.     }
134. }
135.
136. private boolean hasPushed(String sourceImageName) {
137.     String name = sourceImageName.split(":")[0];
138.     String tag = sourceImageName.split(":")[1];
139.     Repository repository = repositoryApi.getRepository(name);
140.     log.info("repository: {}", repository);
141.     if (repository == null) {
142.         return false;
143.     }
144.     List<String> repositoryTags = repositoryApi.getRepositoryTags(name);
145.
146.     return repositoryTags.stream().anyMatch(tag::equals);
147. }

```

5.4.2 部署实验设计细节

实验容器部署是本项目的核心功能。在项目中我使用 Fabric8io/kubernetes-client 库来操作 k8s，封装了一个工具库以方便使用。

首先是填写配置信息，在项目中我已经配置好大部分的配置信息，只需要老师配置

镜像名、端口、启动命令、环境变量、挂载目录等关键信息即可。配置信息会以 json 的形式存储在数据库中。接着，当用户需要启动环境时，将会读取配置信息，部署一个容器。接着将容器信息返回到前端。譬如我们选择 `vscode` 环境，将会返回 `vscode` 的地址，前端会将页面嵌入到实验页中，也可以在新窗口打开，若是选择 `jupyter` 环境将会提示在新窗口打开，因为如果内嵌会存在跨域问题，这里我没有解决。最后实验结束后，可以删除容器，此时因为将文件挂载在 NFS 服务器上了，所以实验代码不会丢失。

下面展示部署部分的核心代码：

```

1.  public class K8sDeployment {
2.
3.      private K8sConfigure k8sConfigure;
4.
5.      private String labId;
6.
7.      private String studentId;
8.
9.      private String teacherId;
10.
11.     private String nfsPath;
12.
13.     private String nfsServer;
14.
15.     KubernetesClient client = new KubernetesClientBuilder().build();
16.
17.     Container container = new Container();
18.
19.     List<ContainerPort> containerPorts = new ArrayList<>();
20.
21.     List<EnvVar> envVars = new ArrayList<>();
22.
23.     List<VolumeMount> volumeMounts = new ArrayList<>();
24.
25.     List<Volume> volumes = new ArrayList<>();
26.
27.     Map<String, String> labels = new HashMap<>();// tag
28.
29.     ObjectMeta PodMetadata = new ObjectMeta();// Pod metadata
30.
31.     ObjectMeta deploymentMetadata = new ObjectMeta();// deployment metadata
32.
33.     PodTemplateSpec podTemplateSpec = new PodTemplateSpec();
34.

```

```

35. Deployment deployment = new Deployment();// deployment
36.
37. PodSpec podSpec = new PodSpec();// Pod spec
38.
39. DeploymentSpec deploymentSpec = new DeploymentSpec();// deployment spec
40.
41. private void populateLabels() {
42.     labels.put("app", "codeonline-app");
43.     labels.put("teacherId", teacherId);
44.     labels.put("studentId", studentId);
45.     labels.put("labID", labId);
46. }
47.
48. private void populatePodMetadata() {
49.     PodMetadata.setLabels(labels);
50.     PodMetadata.setName("codeonline-pod-" + labId + "-" + studentId);
51. }
52.
53. private void populateDeploymentMetadata() {
54.     deploymentMetadata.setLabels(labels);
55.     deploymentMetadata.setName("codeonline-deployment-" + labId + "-" + studentId);
56.     deploymentMetadata.setNamespace(CloudConstants.K8S_NAMESPACE);
57. }
58.
59. private void populateContainerPorts() {
60.     for (int i = 0; i < containerPorts.size(); i++) {
61.         ContainerPort containerPort = new ContainerPort();
62.         containerPort.setContainerPort(containerPorts.get(i).getContainerPort());
63.         containerPort.setName("tcp-" + containerPorts.get(i).getContainerPort());
64.         containerPort.setProtocol("TCP");
65.         containerPorts.add(containerPort);
66.     }
67. }
68.
69. private void populateEnvVars() {
70.     for (int i = 0; i < k8sConfigure.getEnvs().size(); i++) {
71.         Map<String, String> env = k8sConfigure.getEnvs().get(i);
72.         if (!env.isEmpty()) {
73.             EnvVar envVar = new EnvVar();
74.             envVar.setName(env.get("key"));
75.             envVar.setValue(env.get("value"));
76.             envVars.add(envVar);
77.         }
78.     }
79. }
80.

```

```

81.     private void populateVolumeMounts() {
82.         if(!StringUtils.isEmpty(k8sConfigure.getVolume())&&!StringUtils.isEmpty(nfsPath) && !StringUtils.isEmpt
y(nfsServer)){
83.             VolumeMount volumeMount = new VolumeMount();
84.             volumeMount.setName("code-volume-" + labId + "-" + studentId);
85.             volumeMount.setMountPath(k8sConfigure.getVolume());
86.             volumeMounts.add(volumeMount);
87.         }
88.     }
89.
90.     private void populateVolumes() {
91.         if (!StringUtils.isEmpty(k8sConfigure.getVolume())&&!StringUtils.isEmpty(nfsPath) && !StringUtils.isEmpt
y(nfsServer)) {
92.             Volume volume = new Volume();
93.             volume.setName("code-volume-" + labId + "-" + studentId);
94.             volume.setNfs(new NFSVolumeSource(nfsPath, false, nfsServer));
95.             volumes.add(volume);
96.         }
97.     }
98.
99.     private void populateContainer() {
100.        container.setImage(k8sConfigure.getImageName());
101.        container.setName("codeonline-container-" + labId + "-" + studentId);
102.        container.setPorts(containerPorts);
103.        container.setEnv(envVars);
104.        if (!StringUtils.isEmpty(k8sConfigure.getStartCmd())) {
105.            container.setCommand(Collections.singletonList(k8sConfigure.getStartCmd()));
106.            if(!StringUtils.isEmpty(k8sConfigure.getStartArgs())){
107.                String[] split = k8sConfigure.getStartArgs().split(", | ");
108.                container.setArgs(Arrays.asList(split));
109.            }
110.        }
111.        container.setSecurityContext(new SecurityContextBuilder().withPrivileged(k8sConfigure.isPrivilege()).build()
);
112.        container.setVolumeMounts(volumeMounts);
113.    }
114.
115.
116.
117.
118.     private void populatePodSpec() {
119.        podSpec.setContainers(new ArrayList<Container>() {{
120.            add(container);
121.        }});
122.        podSpec.setRestartPolicy("Always");
123.        podSpec.setDnsPolicy("ClusterFirst");

```

```
124.     if ("harbor".equals(k8sConfigure.getSourceFrom())){
125.         List<LocalObjectReference> localObjectReferences = new ArrayList<>();
126.         localObjectReferences.add(new LocalObjectReference("harbor-registry-secret"));
127.         podSpec.setImagePullSecrets(localObjectReferences);
128.     }
129.     podSpec.setSchedulerName("default-scheduler");
130.     podSpec.setVolumes(volumes);
131. }
132.
133. private void populatePodTemplateSpec() {
134.     podTemplateSpec.setMetadata(PodMetadata);
135.     podTemplateSpec.setSpec(podSpec);
136. }
137.
138. private void populateDeploymentSpec() {
139.     deploymentSpec.setReplicas(1);
140.     LabelSelector labelSelector = new LabelSelector();
141.     labelSelector.setMatchLabels(labels);
142.     deploymentSpec.setSelector(labelSelector);
143.     deploymentSpec.setTemplate(podTemplateSpec);
144. }
145.
146. private void populateDeployment() {
147.     deployment.setMetadata(deploymentMetadata);
148.     deployment.setSpec(deploymentSpec);
149. }
150.
151.
152.
153. public Deployment populate(){
154.     populateLabels();
155.     populatePodMetadata();
156.     populateDeploymentMetadata();
157.     populateContainerPorts();
158.     populateEnvVars();
159.     populateVolumeMounts();
160.     populateContainer();
161.     populateVolumes();
162.     populatePodSpec();
163.     populatePodTemplateSpec();
164.     populateDeploymentSpec();
165.     populateDeployment();
166.     return deployment;
167. }
168.
169. public K8sDeployment() {
```

```
170.     }  
171.  
172.     public K8sDeployment(K8sConfigure k8sConfigure, String labId, String studentId, String teacherId,String nfsP  
    ath, String nfsServer) {  
173.         this.k8sConfigure = k8sConfigure;  
174.         this.labId = labId;  
175.         this.studentId = studentId;  
176.         this.teacherId = teacherId;  
177.         this.nfsPath = nfsPath;  
178.         this.nfsServer = nfsServer;  
179.     }  
180. }
```

第 6 章 平台部署与测试

6.1 环境部署

本项目涉及到的软件环境较多，因此搭建环境稍显复杂。

1. 使用若依脚手架搭建开发环境

我选择使用 RuoYi-Cloud 作为开发的脚手架，前端部分没有默认的 vue2 版本，而是选择配套的 RuoYi-Cloud-Vue3 项目。由于需要大量修改项目名，此处使用 fastbuild-factory 项目来批量改名。

在项目克隆之后，前端部分需要安装 nodejs，我使用的是 16.19.0 版本。安装 yarn，使用 yarn 安装相关依赖以及启动前端。而项目的后端，需要先将 sql 目录下的 sql 文件导入到数据库中，ry-config 配置信息，ry-cloud 是项目的数据库。接着安装 nacos，配置 nacos 中的数据库部分以便可以正常连接到数据库读取 yaml 信息，之后启动 nacos。除此之外还需要将 redis 启动起来。接着就可以启动项目的各个服务了，其中 gateway, system, auth 三个部分必须启动，其他部分可以选择性启动。

2. 使用 kubesphere 搭建 k8s 环境

首先根据 kubesphere 通过 All in one 模式或者集群模式安装 kubesphere 环境，我使用的 kubesphere 版本为 3.3.1，k8s 版本为 1.22.12。在 kubesphere 上需要启用应用商店和 devops 两个可插拔组件，如果通过集群方式安装，可以在安装的 yaml 文件中修改启用两个组件，而 all in one 模式下只能在安装后启用。

在安装完成后可以新建一个企业空间，在企业空间中新建两个项目空间 codeonline-projects 和 codeonline-all，分别是平台部署的空间以及学生实验的空间，除此之外还需要建立一个 devos 空间来自动化部署平台。

3. 在 kubesphere 上部署外围的环境

对于 nacos, reids, harbor, mysql 等外围的一些服务，可以部署在新的服务器上也可以直接部署在项目空间中。这些服务可以很方便的在 kubesphere 的应用商店中安装。安装过程需要注意几点，nacos 需要配置挂载目录，挂载 application.properties 并且修改其中数据库部分。harbor 需要将端口类型改为 nodePort, tls 改为 false, url 使用 http，因为我们是 http，而 docker 默认不允许，我们还需要在 docker 的 daemon.json 中添加我们的 harbor 链接并且重启服务。还有 mysql 服务必须在 nacos 之前启动。

4. 使用 devops 自动化发布

因为项目有很多的模块，如果手动部署非常困难，因此需要 devpos 来简化部署流程。在 kubesphere 中建立一个 devops 项目，分别建立前端部署和后端部署两条流水线。前端部署流程为：git 拉去代码，构建项目，将构建后的文件复制到 nginx 中，docker 打包，push 到 harbor 仓库，k8s 部署。后端也是类似：拉去代码，maven 构建，docker 打包，推送到 harbor，k8s 部署。

5. NFS 服务器安装

先安装 nfs 服务器，接着新建一个共享目录，然后将共享目录以读写权限暴露给指定网段中的所有主机，最后重启服务即可。

6.2 测试目标与测试方法

6.2.1 测试目标

测试目标是评估我们的在线编程平台在并发用户下的性能，以及其功能的完整性。

6.2.2 测试方法

对整个平台进行系统测试，验证平台在各种操作环境下是否能正常运行。测试内容包括功能测试、性能测试等。

6.3 测试结果

6.3.1 功能测试结果

经过对多项功能，包括用户注册、登录、身份绑定、教师创建修改删除课程、教师创建修改删除实验、测试实验、查看实验成绩，以及学生加入课程，完成实验，查看实验结果等进行了检测，结果显示这些功能均能正常运行，未出现异常。以下展示一些关键部分的功能测试。

1. 镜像模块测试

镜像模块主要包括镜像上传和镜像查看两个部分。

如图 6-1 所示，课程管理菜单下的镜像市场页面展示了个人和公开的镜像。

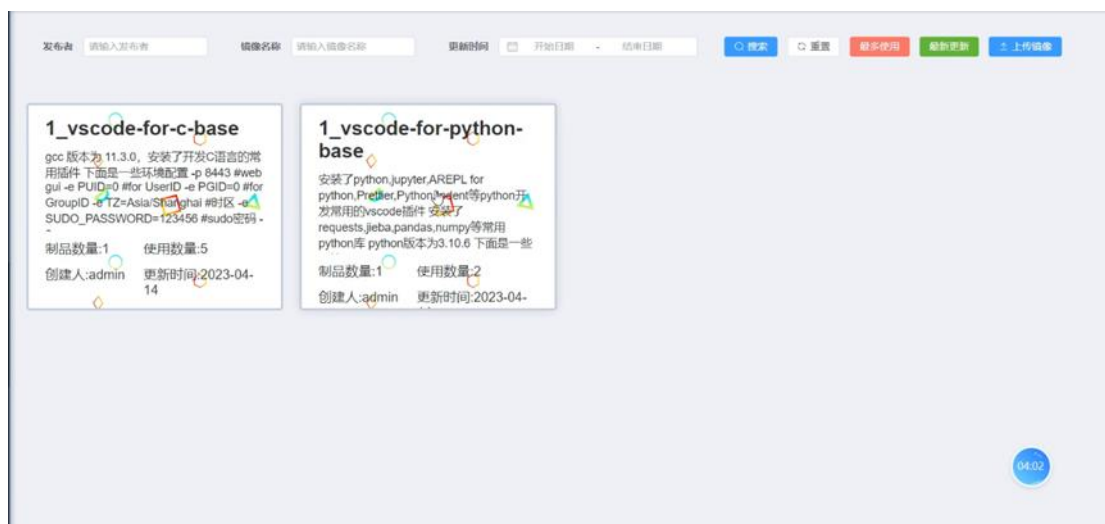


图 6-1 镜像市场页面

如图 6-2 所示，用户可以在上传镜像页面上传镜像，成功后在镜像列表显示。

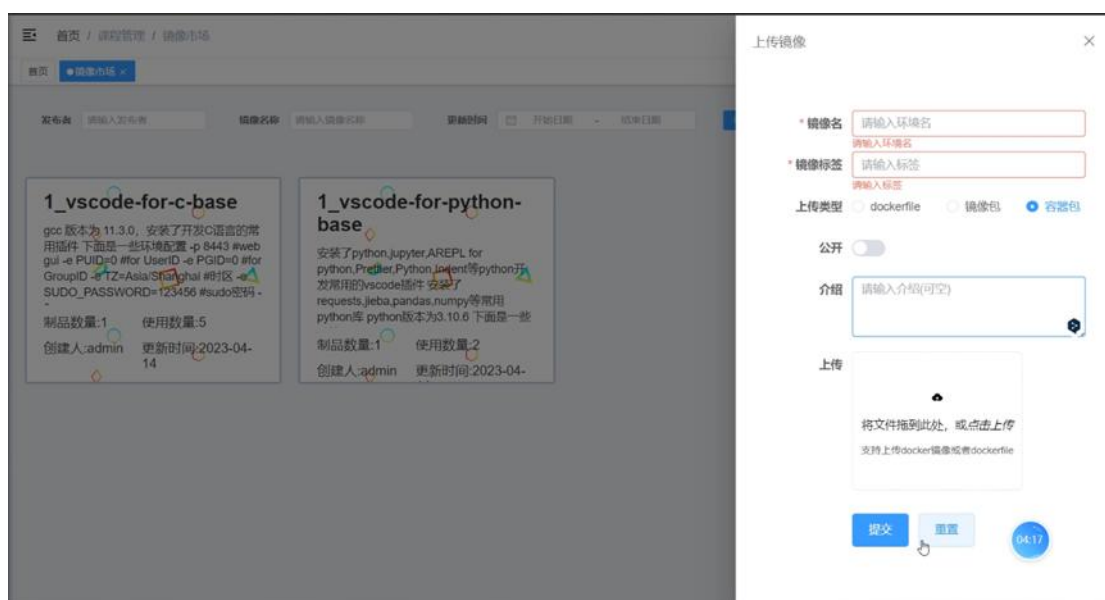


图 6-2 镜像上传

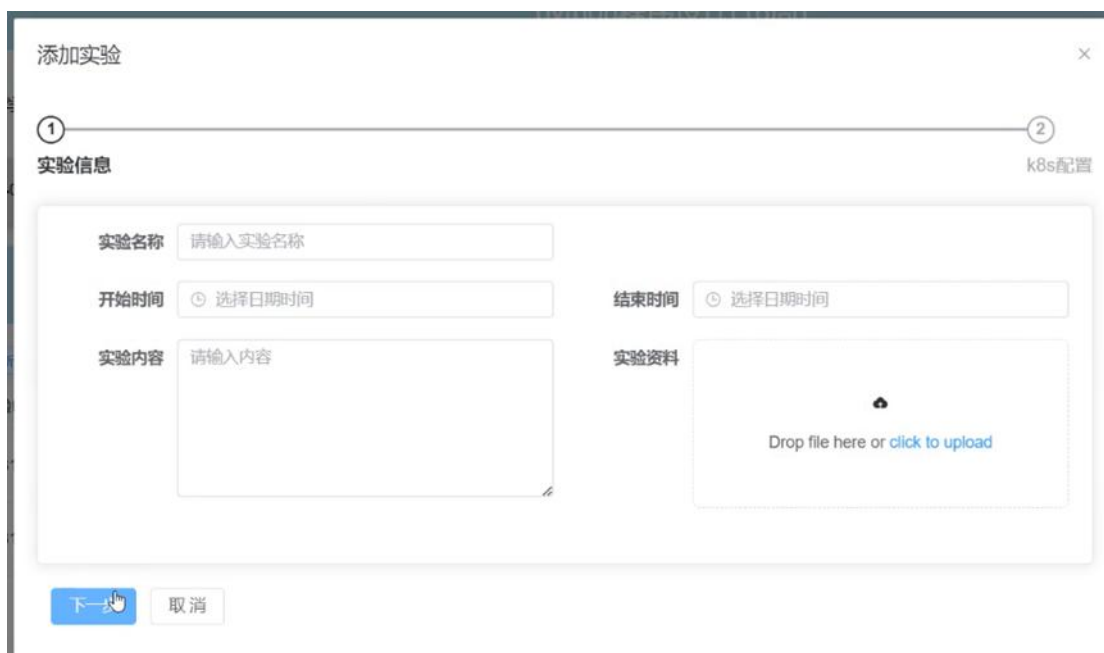
2. 实验模块测试

如图 6-2 教师可以新建课程，并去管理。



图 6-3 课程页面

如图 6-4 和图 6-5 所示学生新建实验要填写基本信息，配置实验环境。

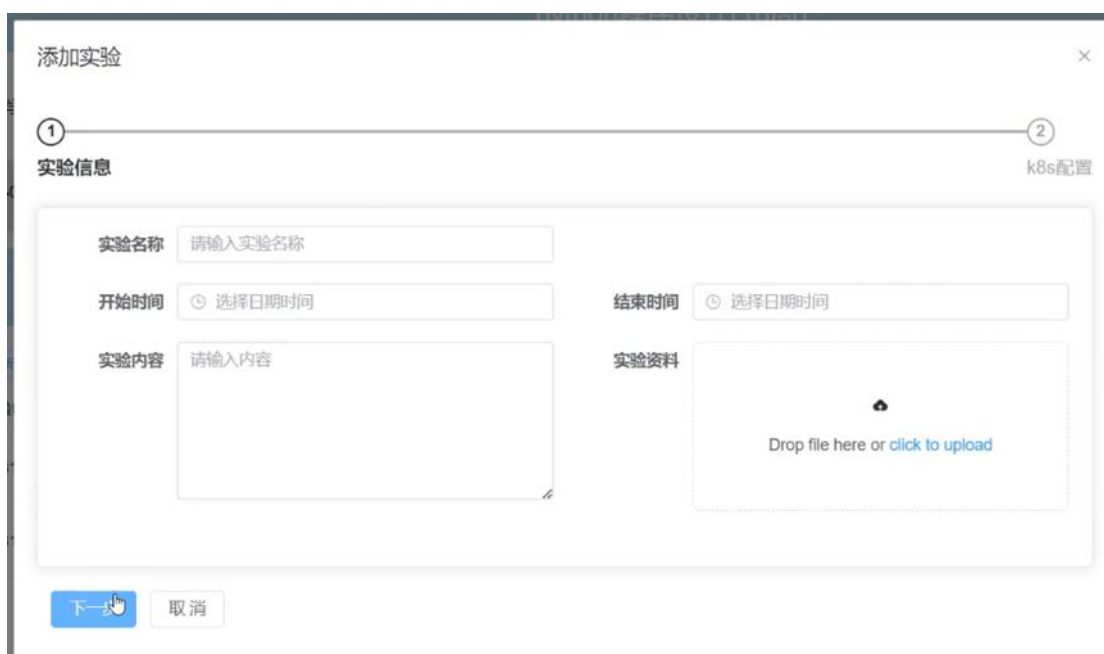


The figure shows a web form titled "添加实验" (Add Experiment) with a close button (X) in the top right corner. A progress bar at the top indicates two steps: "1 实验信息" (Experiment Information) and "2 k8s配置" (k8s Configuration). The "1 实验信息" step is currently active. The form contains the following fields:

- 实验名称** (Experiment Name): A text input field with the placeholder "请输入实验名称".
- 开始时间** (Start Time): A date and time picker with the label "选择日期时间".
- 结束时间** (End Time): A date and time picker with the label "选择日期时间".
- 实验内容** (Experiment Content): A large text area with the placeholder "请输入内容".
- 实验资料** (Experiment Materials): A file upload area with a cloud icon and the text "Drop file here or [click to upload](#)".

At the bottom of the form, there are two buttons: "下一步" (Next Step) and "取消" (Cancel).

图 6-4 实验信息



The figure shows a web form titled "添加实验" (Add Experiment) with a close button (X) in the top right corner. A progress bar at the top indicates two steps: "1 实验信息" (Experiment Information) and "2 k8s配置" (k8s Configuration). The "2 k8s配置" step is currently active. The form contains the following fields:

- 实验名称** (Experiment Name): A text input field with the placeholder "请输入实验名称".
- 开始时间** (Start Time): A date and time picker with the label "选择日期时间".
- 结束时间** (End Time): A date and time picker with the label "选择日期时间".
- 实验内容** (Experiment Content): A large text area with the placeholder "请输入内容".
- 实验资料** (Experiment Materials): A file upload area with a cloud icon and the text "Drop file here or [click to upload](#)".

At the bottom of the form, there are two buttons: "下一步" (Next Step) and "取消" (Cancel).

图 6-5 配置信息

当课程添加完成后，如图 6-6 所示，实验管理界面将会多一个实验信息列，其中有四个按钮：修改、删除、测试、管理。

实验管理						
+ 新增						
实验ID	实验名称	实验内容	开始时间	结束时间	创建人	操作
1681510792003	实验1	在jupyter中编写杨辉三角	2023-04-15 06:17:40	2023-04-20 06:17:43	admin	管理 测试 修改 删除
1681513740335	实验2	使用vscode编写汉诺塔问题	2023-04-05 00:00:00	2023-04-28 00:00:00	admin	管理 测试 修改 删除

图 6-6 实验信息列表

接着如图 6-7 所示，点击测试按钮将会进入到实验环境界面。



图 6-7 实验环境页面

在此我尝试打开一个实验，由于 jupyter 有跨域问题，此处显示了链接，如图 6-8 所示。



图 6-8 显示 jupyter 链接的实验页面

之后如图 6-9，我们可以在 jupyter 中编写代码。

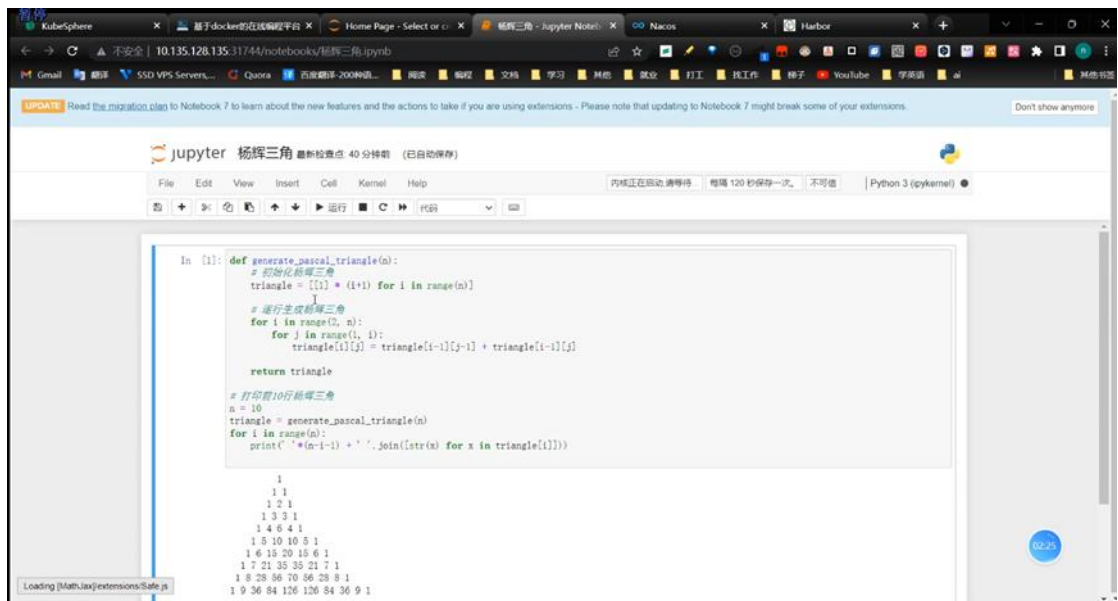


图 6-9 在 jupyter 中编写代码

此外，另一个实验中启动的是 vs code 环境，如图 6-10 所示，vscode 页面可以内嵌在网页中，也可以单独在新的页面打开。

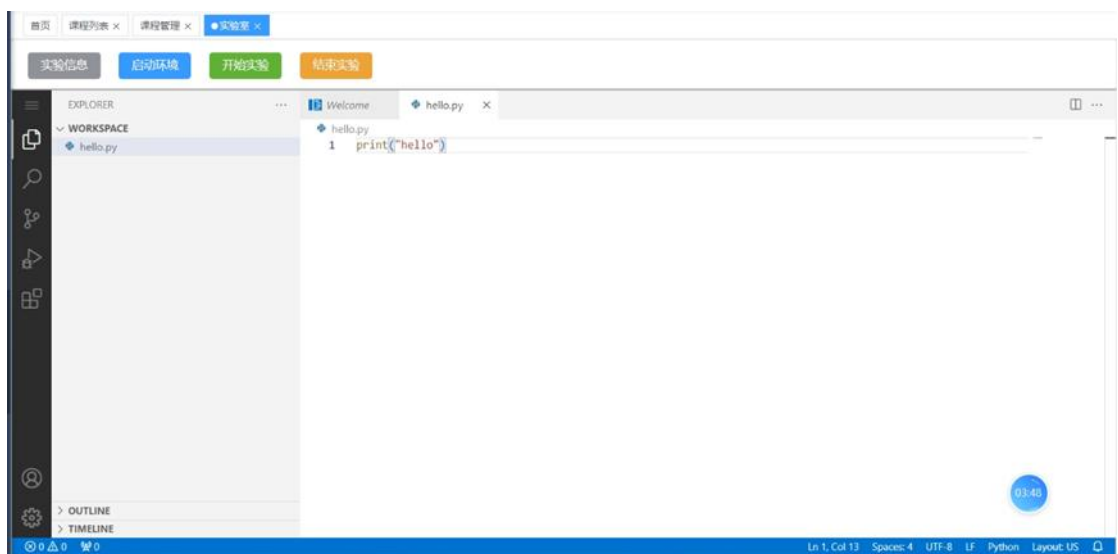


图 6-10 在 vscode 中打开实验

对于学生而言，加入到课程后，如图 6-11 所示，在我的课程里可以看到正在学习的课程。



图 6-11 学生的课程

进入后，如图 6-12 所示，我们可以看到实验列表。



图 6-12 实验列表

点击进入实验则可以进入实验界面，如图 6-13 所示。

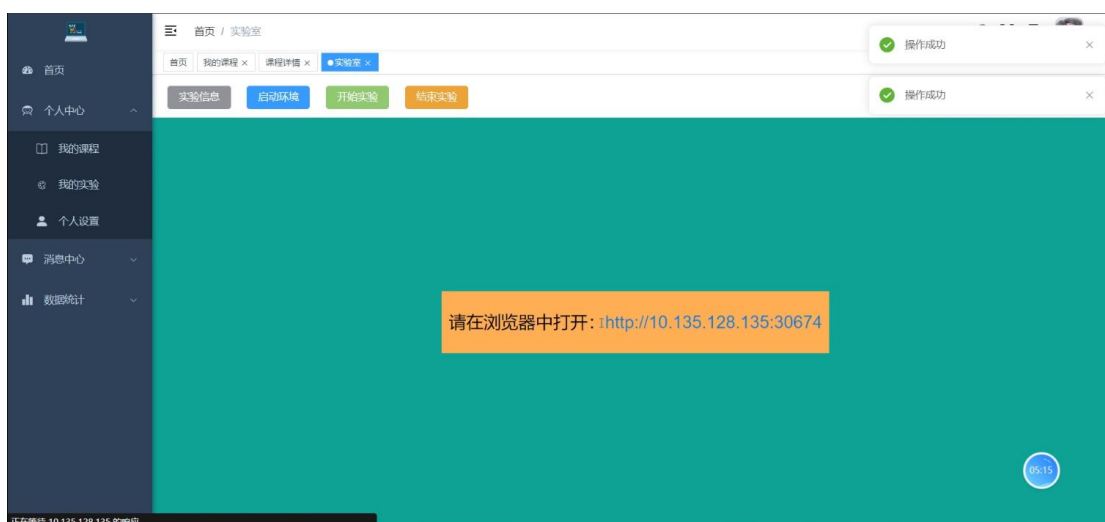


图 6-13 学生进入实验页面

实验完成后，教师可在管理页查看学生实验状态，包括学生 ID、姓名、实验时间、时长、状态和成绩等。如图 6-14，教师通过点击管理即可进入该页。页面还提供“进入实验”和“管理”两个操作按钮。

学生ID	学生姓名	课程ID	实验ID	创建时间	销毁时间	实验时长	实验状态	成绩	操作
10002	10002	3	1678943372804	2023-03-16 18:25	2023-03-16 18:26	1分钟	已结束		管理 进入实验
10004	10004	3	1681510792003	2023-04-14 23:16	2023-04-14 23:16	0分钟	已结束		管理 进入实验
10001	10001	3				0分钟	未开始		管理 进入实验
student	student	3				0分钟	未开始		管理 进入实验

图 6-14 实验管理页

点击管理，可以进入每个学生的具体管理页，如图 5-15，此处可以看见学生的实验文件并且对本次学生实验完成情况进行点评和打分。

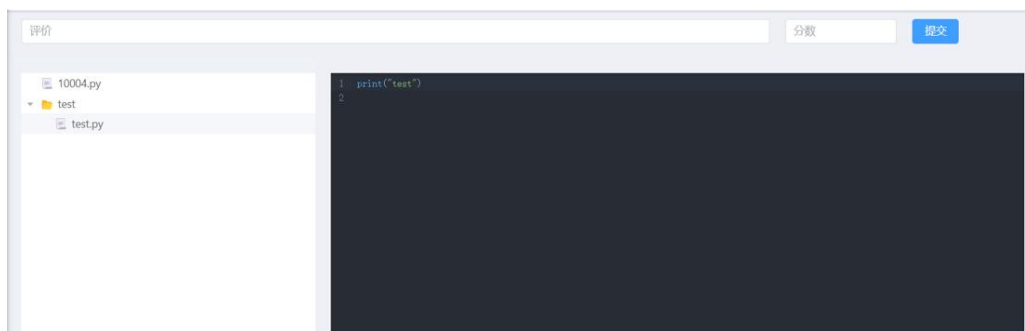


图 6-15 学生具体管理页

6.3.2 性能测试结果

如图 6-16 所示，项目环境为 16 核 cpu，47.49G 内存、193.17G 硬盘空间。



图 6-16 项目资源情况概览

之后我选择一个 vs code 的 docker 镜像进行部署，并对代码稍作修改，以便部署多个实验环境，如图 6-17 所示。

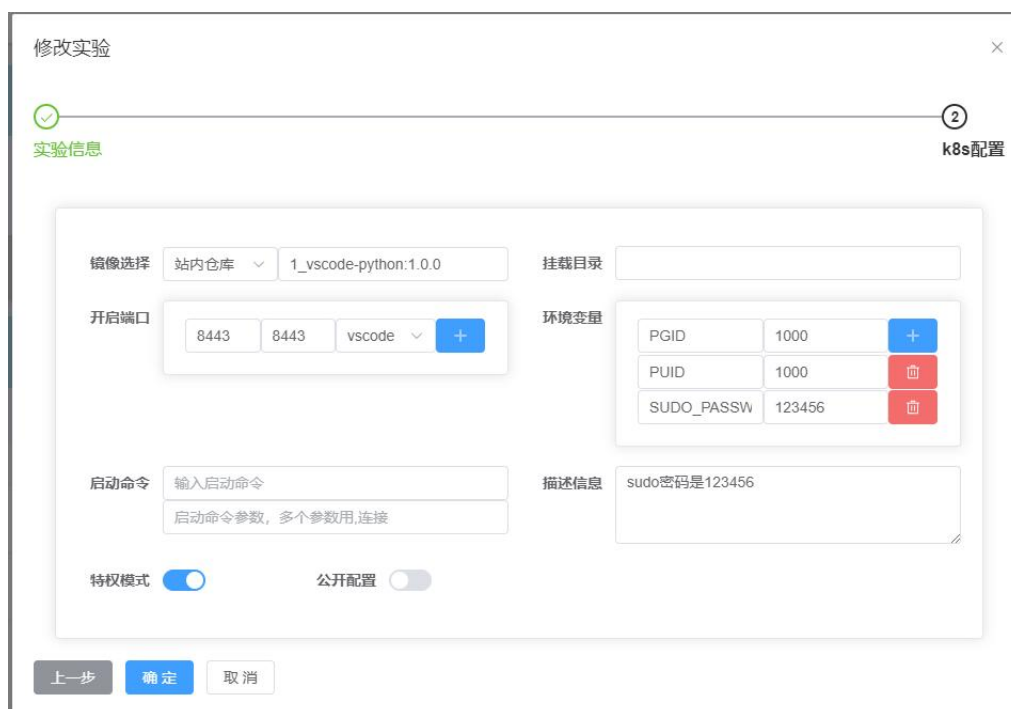


图 6-17 实验镜像配置

此后我部署了 128 个实验环境，后又为了方便增加，在其中一个部署下直接增加了 123 个容器，共计部署 250 个容器，如图 6-18 所示。

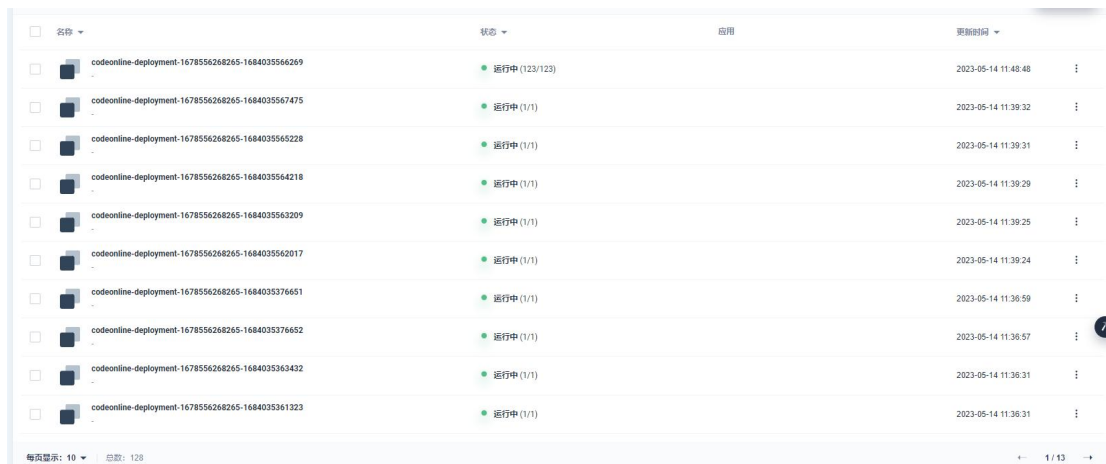


图 6-18 容器部署截图

部署后，如图 6-19 所示，内存使用增加到了 30.69G，平均每个容器占用内存不到 100M。

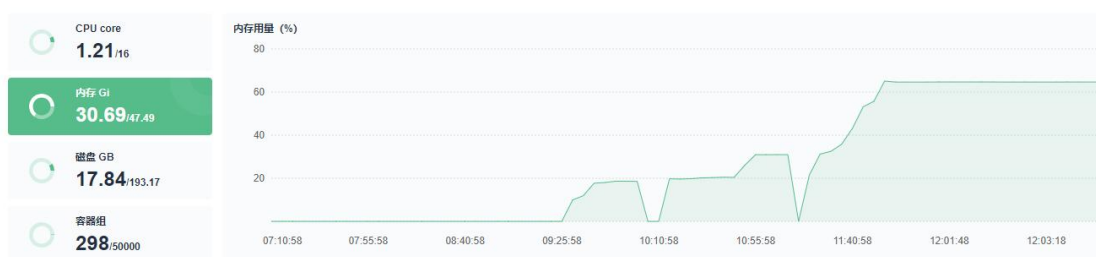


图 6-19 内存变化图

6.4 测试结果分析

根据功能测试结果，我们的在线编程平台在各种操作情境下均表现出稳定的性能，所有测试的功能均正常运行。在性能测试方面，经过大规模部署后，平台的内存消耗保持在合理范围。然而，我们需要注意到，这些数据是在无用户活动状态下测量的。在实际使用情况下，每个容器的资源消耗可能会增加，具体取决于用户行为和运行的程序。

在我的测试中，部署一个新的编程环境只需要几秒钟的时间，而且在无用户活动的情况下，它仅仅占用了几十兆的内存。这显然优于传统的虚拟机技术部署方式。这种优势不仅体现在更快的部署速度和更少的资源消耗上，还体现在它为平台提供了更大的灵活性和更高的扩展性。因此，可以说，这个基于 Docker 的在线编程平台在性能和效率上远优于传统的基于虚拟机的在线编程平台。

第 7 章 总结与展望

本文提出了一个基于 Docker 的在线编程平台设计，利用容器化技术部署实验环境，相较于传统的虚拟机技术，容器化技术更轻便且易于部署。通过该平台，用户能够获得统一的在线编程环境，从而极大地提高学习效率。在文章中，我们详细介绍了平台的设计、搭建过程以及关键模块设计等方面的内容。在实践过程中，我们积累了许多宝贵的经验，对 Docker 和 Kubernetes 有了更深入的了解，并提高了系统设计方面的认识。

尽管本文实现了一个基于 Docker 和 Kubernetes 的在线编程实验平台，但由于时间限制，本系统仅实现了核心功能，许多细节性的功能尚未完全实现。例如，在镜像市场部分，我们希望提供一个开放的镜像市场，允许大家分享自己的镜像、配置、评论和提问等。在实验部分，应该有更丰富的功能，如学生和教师在线交流、更强大的学生实验管理以及更优秀的实验数据分析等。此外，代码设计方面需要改进，例如简化的鉴权部分，这样可能导致安全问题。界面设计和交互友好性也存在一定的问题，由于这段时间忙着做其他项目，暂时没有时间去完善该系统。

在未来，我们希望对系统进行重新设计，搭建一个高可用的实验平台。教师可以方便地分享自己的实验环境和配置，简化部署难度。学生则能更轻松地进行实验和与教师沟通。此外，提供学习数据分析将有助于帮助教师和学生更好地完成教学任务。

致 谢

为了避免遗漏任何一个值得感谢的人——感谢全人类。

感谢你们的存在，让我在这个奇妙的世界里找到自己的位置。

参考文献

- [1] 李盘. 基于 Docker 的开发者实验室系统设计与实现[D].华中科技大学, 2019.
- [2] 林志煌, 张菽妍, 赵莲芬.基于 Docker 技术的高校在线编程平台设计与实现[J].现代计算机, 2022, 28(14):107-111.
- [3] 李景奇, 程樊启, 管军.基于混合云的高校网络学习空间设计[J].中国信息技术教育, 2022, No.377(02):95-98.
- [4] 崔艳敏.基于 Docker 的课程实验平台设计以及实现探讨[J].中阿科技论坛(中英阿文), 2020, No.13(03):144-145.
- [5] 岳昆, 胡矿, 袁国武, 谭明川.开源框架下在线编程平台建设与实践教学改革[J].软件导刊, 2023, 22(03):212-216.
- [6] 付成玉. 少儿在线编程学习体验影响因素研究 [D]. 贵州师范大学, 2022.DOI:10.27048/d.cnki.ggzsu.2022.000874.
- [7] 刘以. 在线编程平台直播教学系统的设计与实现[D].北京邮电大学, 2021.DOI:10.26969/d.cnki.gbydu.2021.002130.
- [8] 王彩玲, 张静.基于 Docker 的人工智能实验平台搭建[J].信息技术与信息化, 2023, No.274(01):185-188.
- [9] 周玮, 辛政华, 杨小莹等.基于 Docker 的人工智能实验教学平台研究与设计[J].鄂州大学学报, 2022, 29(03):101-103.DOI:10.16732/j.cnki.jeu.2022.03.036.
- [10] 李朋, 赵志刚, 宋婉玉.基于 Docker 的人工智能实验平台[J].信息技术与信息化, 2021, No.258(09):207-209.
- [11] 李耀, 周亚明, 罗辉等.基于 Docker 的嵌入式软件测试研究[J].电子技术与软件工程, 2020, No.182(12):67-69.
- [12] 梁进科, 陈路路, 王一, 张建廷.容器云环境下可视化编排技术[J].计算机与网络, 2021, 47(23):58-60.
- [13] 吴逸文, 张洋, 王涛等.从 Docker 容器看容器技术的发展:一种系统文献综述的视角[J/OL].软件学报:1-25[2023-05-17].<https://doi.org/10.13328/j.cnki.jos.006765>.
- [14] 李秋贤, 石云升, 周全兴.基于 Docker 容器的信息安全攻防平台[J].现代信息科技, 2021, 5(02):114-117+121.DOI:10.19850/j.cnki.2096-4706.2021.02.029.
- [15] 盛乐标, 周庆林, 游伟倩等.Kubernetes 高可用集群的部署实践[J].电脑知识与技术,

- 2018, 14(26):40-43.DOI:10.14004/j.cnki.ckt.2018.3151.
- [16] 游向东, 徐圆圆, 欧阳松.基于 Docker 的大数据 AI 教学与实验系统[J].软件, 2018, 39(08):192-197.
- [17] 田杨锋, 王振.基于 K8s 的 PaaS 架构及业界典型产品的调研分析[J].科学技术创新, 2018(06):97-98.
- [18] 应毅, 刘亚军, 任凯.基于容器云的分布式深度学习实验平台构建[J].实验技术与管理, 2022, 39(03):147-152.DOI:10.16791/j.cnki.sjg.2022.03.027.
- [19] 方中纯, 李海荣.基于 Docker 的大数据教学与实验平台的设计与实现[J].信息技术, 2022(10):7-11.DOI:10.13274/j.cnki.hdzj.2022.10.002.
- [20] 程宁.基于 Harbor 实现 Docker 私有仓库搭建的研究[J].现代工业经济和信息化, 2022, 12(09):73-75.DOI:10.16525/j.cnki.14-1362/n.2022.09.030.
- [21] 张刚刚.智慧校园体系下基于 Docker 的 Web 应用快速部署研究[J].电脑编程技巧与维护, 2022(08):125-127.DOI:10.16184/j.cnki.comprg.2022.08.008.
- [22] 黄飞.基于 Docker 的高校私有云平台建设研究[J].电脑编程技巧与维护, 2022(04):100-102+134.DOI:10.16184/j.cnki.comprg.2022.04.003.
- [23] 邱建新.基于 Docker 容器技术的 Linux 在线实验环境设计[J].信息技术, 2022(02):48-52+58.DOI:10.13274/j.cnki.hdzj.2022.02.009.
- [24] 孙波.浅谈微服务架构、Docker 和 Kubernetes[J].现代电视技术, 2022(02):100-103.
- [25] 张亮, 张丽梅.基于 Docker 技术的大数据专业综合实训平台的设计与实现[J].电脑与电信, 2022(Z1):46-48+57.DOI:10.15966/j.cnki.dnydx.2022.z1.014.
- [26] 杨碎明. 基于 Docker 的智慧教学平台研究与实践[C]//新课程研究杂志社.《“双减”政策下的课程与教学改革探索》第四辑.[出版者不详], 2022:29-30.DOI:10.26914/c.cnkihy.2022.015872.
- [27] 袁竞.基于 Docker 的个人私有云框架设计[J].信息技术与信息化, 2021(12):136-138.
- [28] 刘亚.关于 Docker 镜像仓库技术的研究[J].科学技术创新, 2021(29):76-78.
- [29] 叶惠仙, 游金水.基于 Docker 技术在高校教学平台建设中的应用[J].电脑与信息技术, 2021, 29(04):42-44+52.DOI:10.19414/j.cnki.1005-1228.2021.04.011.
- [30] 高鹏伟. 基于 Kubernetes 和 Docker 的容器云平台设计与实现[D].西安电子科技大学, 2021.DOI:10.27389/d.cnki.gxadu.2021.002684.
- [31] 谢剑刚, 肖小红, 薛云兰.基于 Kubernetes 的数据库技术课程远程实验平台[J].信息

技术与信息化, 2021(07):204-206.

[32] 兰智博. 基于 Kubernetes 的容器云平台的设计与实现[D]. 北京邮电大学, 2020.DOI:10.26969/d.cnki.gbydu.2020.000565.

[33] 马明瑞. 基于 Web 的在线编程平台的设计与实现[D]. 华中科技大学, 2021.DOI:10.27157/d.cnki.ghzku.2021.002062.

[34] 马俨纹. 在线编程虚拟实验系统设计与研究[D]. 电子科技大学, 2021.DOI:10.27005/d.cnki.gdzku.2021.004919.

[35] 江友华. 基于 Scratch3 的在线编程系统的设计与实现[D]. 北京邮电大学, 2020.DOI:10.26969/d.cnki.gbydu.2020.001919.

[36] 黄小冬. 基于 CodeMirror 的 Web 在线编程实训模块设计与实现[J]. 软件工程, 2019, 22(02):45-47.DOI:10.19644/j.cnki.issn2096-1472.2019.02.014.

[37] 窦妍. “微课堂”教育平台的在线编程评测子系统的设计与实现[D]. 南京大学, 2020.DOI:10.27235/d.cnki.gnjiu.2020.001614.

[38] 方志宁, 谢华, 张金营, 曹亚南. 基于微服务架构的统一应用开发平台[J]. 仪器仪表用户, 2023, 30(03):93-97.

[39] 董子奇, 刘淇, 高原, 刘威, 符鹏. 基于容器技术的微服务部署研究[J]. 信息技术与标准化, 2023(Z1):93-98.

[40] 庄芳, 屠臻. 基于 MyBatis 框架的多终端互通在线教学平台设计与实现[J]. 自动化技术与应用, 2022, 41(01):182-185.

附 录

项目源码: <https://github.com/woniu9524/codeonline>