基金项目：否

作者简介：韩东（1991-），男，江苏省、盱眙人，硕士，江苏科技大学计算机科学与工程学院，研究方向为软件工程，容器虚拟化技术，DevOps。

**基于kubernetes容器集群环境的持续交付平台设计与实现**

韩东

（江苏科技大学 计算机科学与工程学院，江苏 镇江 212003）

**摘要：**传统软件交付模式偏向于手动，软件从源码到编译、打包、测试、交付生产经历不同的形态和系统环境。手动模式效率较低，且不同的系统环境存在差异容易引起意想不到的错误导致软件部署失败。设计基于kubernetes容器集群环境的持续交付平台，利用容器技术隔离的特性将程序运行环境的差异屏蔽掉，利用git、docker、kubernetes的C/S架构模式，集成每个平台的客户端及相关构建工具，使得程序在拉取项目、编译、部署过程中脱离命令行模式，只需要在持续交付平台中通过表单的方式配置每个过程中所需的各项参数，将软件交付过程中所有的操作集中到统一的平台上，实现自动化交付。系统测试结果表明，持续交付平台能够自动化的完成软件的交付工作，打通运维与开发人员隔阂之墙，将DevOps理念落地。

**关键词：**容器集群；持续交付；Docker；Kubernetes；

1. **引言**

持续交付是DevOps中一个重要的方法，主要研究通过自动化的方法使得软件部署与交付变得更加便利。DevOps强调应该持续的集成与部署，做到每天进行价值交付，测试人员在软件开发的过程中就参与进来，每天进行集成测试，运维人员进行持续部署。DevOps强调通过自动化“软件交付”和“架构变更”的流程，来使得构建、测试、发布软件能够更加地快捷、频繁和可靠。矛盾在于软件部署通常是复杂的，经常出现测试环境、生产环境与开发环境不一致的情况，从而导致软件交付失败。开发人员想每天交付新的功能，而运维人员期望系统稳定，这似乎是不可调和的矛盾。在传统模式下，交付软件所花费的成本过高，导致DevOps期望的测试与运维人员及早参与到开发中来，每天进行集成测试的愿景无法真正实现。

面对持续交付中的矛盾，很多学者与从业人员给出了多种多样的解决方法，如周振兴等提出了通过 GitHub以及 IBM Urban Code Deploy（UCD）工具进行自动化部署实现持续交付，并通过虚拟化技术 Docker 对系统进行水平扩展，在系统出现资源利用率高、系统压力大的情况下，可以快速提高系统的容量，动态提高系统性能；丘晖利用了Jenkins的Pipeline功能实现向Docker容器环境下的持续交付；周雪提出基于神经网络的自调节持续集成与交付框架，利用神经网络自学习和自调节的优势为开发者预测和计划最佳的下一次构建，从而帮助开发者合理平衡开发与集成。金泽锋等认为软件交付时应注重价值交付，提出了面向完整价值交付的文档DevOps，产品文档应与软件同时交付，避免软件版本与文档不匹配的问题。

本文提出了基于kubernetes容器集群环境的持续交付平台。利用Docker容器的隔离特性屏蔽掉程序不同环境运行的差异性，引入kubernetes容器集群及应用编排技术来实现自动化的交付工作。kubernetes能够应对复杂的软件架构和应用场景。在DevOps理念广泛应用的背景下，Docker，kubernetes在软件交付中已经广泛的使用。而在目前的使用形式大多为：开发人员将编译完成的程序包发送给运维人员，由运维人员在根据需要使用命令行的方式在运行Docker引擎的服务器上生成Docker镜像，再使用kubernetes的kubectl命令行工具，将容器部署到容器集群环境中去，整个过程偏向于手动模式。本文分析了，Docker、kubernetes以及gitlab各系统架构，得出三者都是基于C/S架构模式，可以利用各平台开源的客户端，集成到持续交付平台中，在平台上集成所有操作，以表单的方式配置各流程中的参数，避免使用命令行的方式。

1. **持续交付平台设计**

持续交付平台旨在管理一个软件项目的完整的生命周期，从静态代码开始，到编译与打包，再到将程序包构建成docker镜像。Docker镜像如果只存在开发环境的机器中，则此镜像不便于共享和分发，还需要将镜像推送至镜像仓库中。最后将构建好的镜像按照软件架构的设计部署在容器集群中。整个过程中，软件项目从可编辑的源代码，到二进制程序包，再到容器镜像，整个过程中需要借助相关的平台和工具。例如，使用github作为代码版本控制工具，而docker镜像是要docker服务进行构建的。持续交付平台本身不是各项具体工作的实施者，而是交付流程的创建者，各平台的统一组织者，持续交平台通过是图形化界面的方式进行操作，以表单的方式配置各流程中的参数。

对于企业来说，首先要搭建一套基础设施服务。受企业内部规定和限制，软件项目可能只在企业内部网络使用，并不想公开在互联网上，例如企业不希望自己的项目源代码托管在github上，不希望自己的镜像发布在Docker hub中，所以一些互联网的公共服务不能使用，需要企业自己在内部网络构建一套基础设施服务。本文使用gitlab作为代码版本控制工具，使用harbor作为Docker镜像Registry服务。搭建Docker引擎服务与kubernetes服务集群。

持续交付流程是整个交付平台的核心环节，基于各基础平台的持续交付平台的交付流程如图1所示：

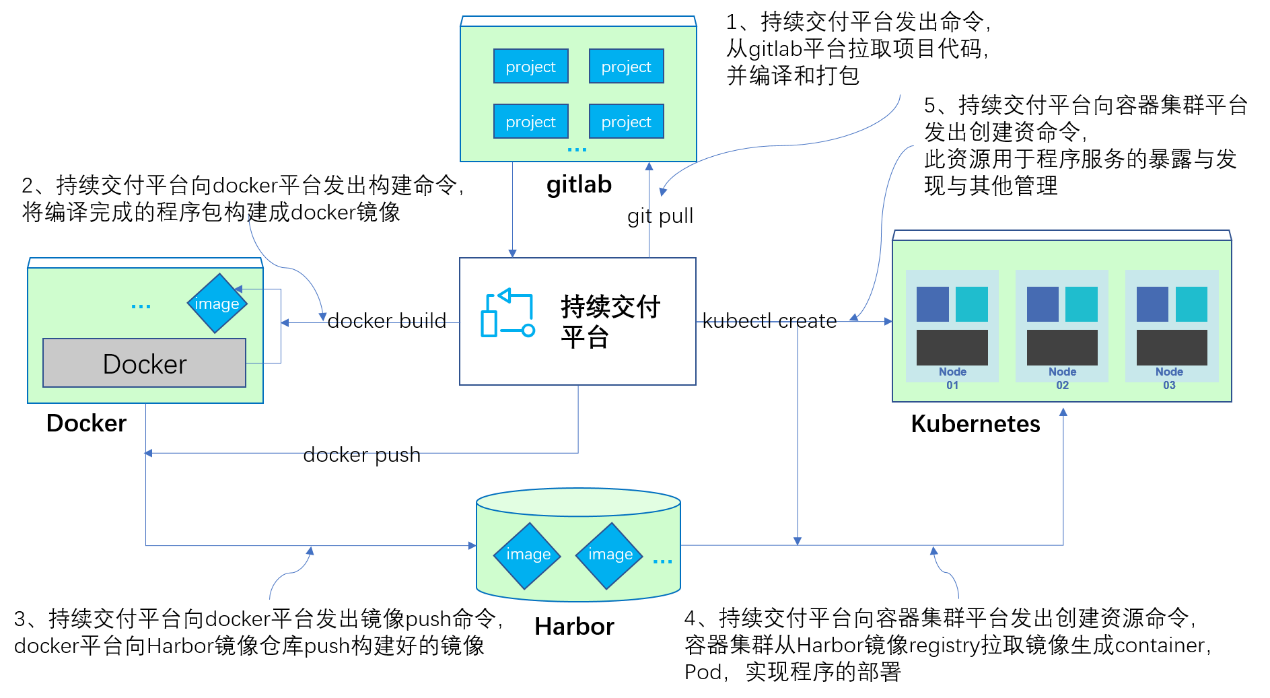


图1

1. 持续交付平台从代码托管平台gitlab上拉取指定项目的源码，并且编译和打包。
2. 持续交付平台根据项目的Dockerfile文件，向Docker平台发出构建镜像命令，docker平台会根据项目的Dockerfile进行镜像的构建。所以每个项目必须配置Dockerfile文件。
3. 镜像构建完成后，此镜像仅存在于Docker平台中，持续交付平台还需要驱动Docker平台将镜像推送至指定的镜像仓库中，等待部署与交付生产环境。
4. 持续交付平台根据项目的架构设计，创建kubernetes Deployment资源，指定相关的镜像和副本数，交付至容器集群中。
5. 如果没有其他操作，此时正在运行的软件服务只能在容器集群内部访问，持续交付平台还需要命令容器集群平台创建kubernetes Service资源，用于服务的发现。此时整个软件项目完成交付，并提供访问。
6. **持续交付平台实现**

实现持续交付平台的重点内容在如何在平台上操作各个基础服务平台完成构建打包、镜像构建与推送、容器集群中的各个资源的创建和管理。幸运的是，各个平台都提供了不同语言版本的客户端，并且开源。持续交付平台需要集成这些客户端，使用这些客户端提供的API实现平台化的操作。如何使用这些API实现编码是关键问题。

2.1、客户端选择

持续交付平台使用Java语言实现使用Maven进行项目构建，因此需要选择各个平台的Java版本的客户端。各平台的客户端Maven依赖详情如下1所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 平台 | git | docker | kubernetes |
| 客户端依赖 | groupId:org.eclipse.jgit  artifactId:org.eclipse.jgit  version:5.5.0.201909110433-r | groupId:com.spotify artifactId:docker-client version:8.16.0 | groupId:io.fabric8 artifactId:kubernetes-api  version:3.0.12 |

表1

2.2、编程实现

在编码实现环节，省略了一般性通用性功能的编码实现如平台的注册/登录，而重点聚焦持续交付流程中的核心功能的编码实现。持续交付平台需要和各个基础平台进行交互，编码的重点就在于如何利用集成的各平台客户端提供的API进行交互的实现。

1. 将项目源码克隆至本地。持续交付平台与gitlab代码托管平台进行交互，需要传递代码仓库的url和仓库的用户名，密码以及保存至本地的目录。具体编码实现如下：

public static void cloneGitRepository(GitRepository gitRepository, String localPath) throws Exception {

//设置代码仓库的url, GitRepository封装了gitlab账户的url，用户名，密码

CloneCommand cc = Git.cloneRepository().setURI(gitRepository.getUrl());

//传入仓库认证：用户名，密码

cc.setCredentialsProvider(

new UsernamePasswordCredentialsProvider(gitRepository.getUsername(),

gitRepository.getPassword()));

//设置保存至本地的目录

cc.setDirectory(new File(localPath)).call();

}

1. 创建Docker镜像。将编译完成的程序包按照Dockerfile文件构建Docker镜像。需要在操作系统的环境变量中配置docker服务所在的地址，如：DOCKER\_HOST=tcp://192.168.18.150:2375。Docker客户端读取DOCKER\_HOST的值与Docker服务建立连接。具体编码实现如下：

public String buildImage(String dockerFileDir, String tag) throws Exception {  
 //读取DOCKER\_HOST的环境变量，创建docker客户端，并建立连接  
 final DockerClient dockerClient = DefaultDockerClient.fromEnv().build();  
 //读取Dockerfile所在的目录，创建Dockerfile对象  
 File dockerFile = new File(dockerFileDir);  
 Path dockerFilePath = dockerFile.toPath();  
 //docker客户端发起build命令创建docker镜像，并给镜像打上tag  
 DockerClient.BuildParam buildParam = new DockerClient.BuildParam("t", tag);  
 //客户端向docker服务平台发出创建镜像的请求，返回ImageId  
 return dockerClient.build(dockerFilePath, buildParam);  
}

1. 推送镜像至Harbor镜像仓库。持续交付平台与Docker平台交互，将docker镜像推送至harbor镜像仓库中。一般来说在企业内部使用的是私有仓库，需要提供仓库的用户名和密码。编码实现如下：

public void pushImage(String username, String pw, String image) throws Exception {  
 final DockerClient dockerClient = DefaultDockerClient.fromEnv().build();  
 //配置Harbor系统账户  
 final RegistryAuth registryAuth = RegistryAuth.create(username, pw, null,

HarborServerAddress, null, null);

//客户端将指定的镜像推送至镜像仓库中  
 dockerClient.push(image, registryAuth);  
}

1. 创建Deployment资源实现应用程序的部署。持续交付平台与kubernetes进行交互，创建Deployment资源实例，由Deployment指挥 Kubernetes 如何创建和更新应用程序的实例。创建 Deployment 后，Kubernetes master 将应用程序实例调度到集群中的各个节点上。首先需要创建kubernetes客户端，默认读取当前系统用户文件夹下的.kube/config文件，该文件是创建kubernetes集群时生成的配置文件，config文件中描述的kubernetes master节点的地址，以及其他认证信息，然后与kubernetes master建立连接。需要指定应用程序的容器镜像以及要运行的副本数。对于私有仓库需要指定仓库的用户名，密码。具体编码实现如下：

public void newDeployment(String deploymentName, String ns, int rs, Map<String, String> labels, Map<String, String> podSelector) {

//创建客户端

Config config = new ConfigBuilder().build();

KubernetesClient client = new DefaultKubernetesClient(config);

//创建Deployment资源控制器对象

Deployment deployment = new DeploymentBuilder().build();

deployment.setKind(KubernetesConstraint.KUBERNETES\_DEPLOYMENT);

//创建Deployment Metadata对象并设置属性

ObjectMeta deploymentMeta = new ObjectMeta();

//设置该Deployment所在的命名空间

deploymentMeta.setNamespace(ns);

//设置该Deployment的名称

deploymentMeta.setName(deploymentName);

//设置该Deployment的标签

deploymentMeta.setLabels(labels);

deployment.setMetadata(deploymentMeta);

//创建Deployment的Spec属性对象

DeploymentSpec spec = new DeploymentSpec();

//设置副本数

spec.setReplicas(rs);

LabelSelector labelSelector = new LabelSelector();

//目标pod的标签选择器

labelSelector.setMatchLabels(podSelector);

spec.setSelector(labelSelector);

//通过Pod模板创建Pod

PodTemplateSpec podTemplateSpec = newPodTemplate();

spec.setTemplate(podTemplateSpec);

deployment.setSpec(spec);

client.apps().deployments().create(deployment);

}

1. 创建Service资源实现服务的暴露与发现。此时，kubernetes为Deployment管理的Pod只创建了集群内部的虚拟IP，只有集群内部才能访问。Service资源可以将程序运行的服务暴露出去，并且具有负载均衡的功能。创建Service资源需要指定Service的name属性，且需要指定需要作用于Pod的标签。具体编码实现如下：

public void newService(String serviceName, String ns, Map<String, String> labels, Map<String, String> selector) {  
 Config config = new ConfigBuilder().build();  
 KubernetesClient client = new DefaultKubernetesClient(config);  
 Service service = new Service();  
 //设置service资源类型名  
 service.setKind(KubernetesConstraint.KUBERNETES\_SERVICE);  
 //新建并设置Service meta对象  
 ObjectMeta serviceMeta = new ObjectMeta();  
 serviceMeta.setName(serviceName);  
 serviceMeta.setNamespace(ns);  
 serviceMeta.setLabels(labels);  
 //新建并设置Service spec对象  
 ServiceSpec serviceSpec = new ServiceSpec();

//设置service资源类型  
 serviceSpec.setType("NodePort");  
 serviceSpec.setSelector(selector);  
 ServicePort port = new ServicePort("http", port, targetPort, KubernetesConstraint.PROTOCOL\_TCP,

new IntOrString(80));  
 List<ServicePort> list = new ArrayList<>();  
 list.add(port);  
 serviceSpec.setPorts(list);  
 service.setSpec(serviceSpec);  
 service.setMetadata(serviceMeta);  
 client.services().create(service);  
}

1. **案例测试**

通过创建一个springboot微服务程序来验证持续交付平台的功能是否达到预期。测试程序能够读取主机名，来验证负载均衡功能是否实现。主要测试能否正确构建Docker镜像并推送至Harbor镜像仓库中；能否通过持续交付平台在kubernetes中部署该程序，并创建副本；能否通过出持续交付平台正确创建服务，实现从集群外部网络进行访问。

（1）、镜像构建与推送测试。构建docker镜像主要使用Dockerfile文件，Dockerfile文件中执行构建的命令，测试程序的Dockerfile如下所示：

# 基础镜像

FROM openjdk:8-jdk-alpine

# 对应pom.xml文件中的dockerfile-maven-plugin插件buildArgs配置项JAR\_FILE的值

ARG JAR\_FILE=target/\*.jar

# 复制打包完成后的jar文件到/opt目录下

COPY ${JAR\_FILE} /opt/app.jar

# 启动容器时执行

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/opt/app.jar"]

# 使用端口80

EXPOSE 80

使用持续交付平台将镜像打上标签：hub.devops.com.helloworld/hello-app:v1，并推送至Harbor镜像仓库中。推送成功后，Habror平台显示如图2：



图2

（2）、部署测试。通过持续交付平台创建Deployment资源，资源名称为：springboot-app-depolyment；镜像为：hub.devops.com/helloworld/hello-app:v1；标签为：app=helloworld；副本数为4。资源创建成功后，通过命令查看kubernetes集群中Deployment资源的状态如图3所示：



图3

图中显示成功创建4个副本，并且成功运行，查看Pod的状态如图4所示：

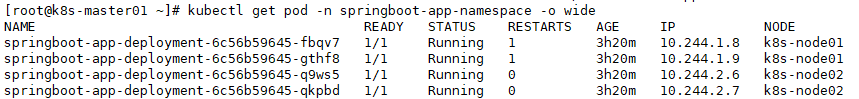


图4

图中显示4个Pod正在运行。

（3）、测试service资源能否将服务暴露出去。持续交付平台创建service资源，类型为NodePort；节点端口为：30715；目标Pod内部端口为：80；需要作用于Pod的标签为：app=helloworld。使用命令查看创建结果如图5所示：



图5

使用Linux curl命令进行服务的访问，curl每次只访问192.168.18.22:30715，结果显示每次访问的返回的主机名每4次重复，说明service资源正常工作，每次访问服务，service使用轮询算法进行负载均衡，如图6所示：

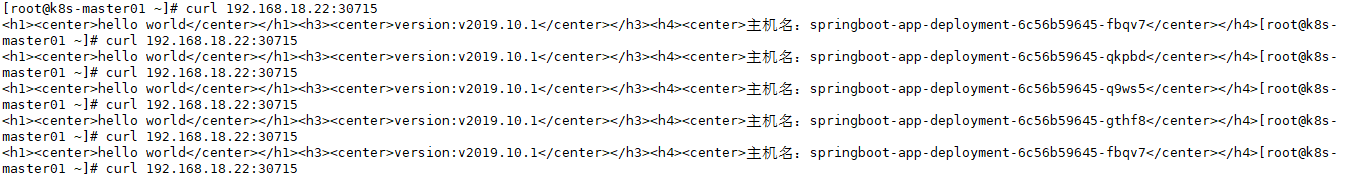


图6

1. **结语**

本文利用kubernetes容器集群技术，整合各基础服务平台的客户端，构建了自动化的持续交付平台，打通了各个环节的数据流和业务流。

**参考文献**

[1]周振兴.基于 Docker 和持续交付的项目管理系统设计与实现[D].大连:大连理工大学,2016.

[2]张文林.持续交付及其在大型项目中的应用[J].软件导刊,2017,16(10):159-161.

[3]丘晖.基于容器的持续集成和部署方法研究[J].广东通信技术,2017,10:62-66.

[4]郭雪.基于神经网络的过程自调节持续集成工具设计与实现[D].南京:南京大学,2019.

[5]金泽锋,张佑文,叶文华,张贺,邵栋.面向完整价值交付的文档DevOps应用研究[J].软件学报,2019,30(10):3127-3147.

[7]马征,缪凯,张广温.浅析 Kubernetes 容器虚拟化技术[J].应用技术,2019,10(2):63-64.

[8]郑冰.基于Kubernetes的企业级容器云平台设计[J].数字技术与应用,2019,37(6):138-141.

[9]翁湦元,杏花,志远,王雪峰.于Kubernetes的容器云平台设计与实践[J].计算机应用,2019,28(12):49-53.

[10]基于Docker的资源调度及应用容器集群管理系统设计与实现[D].山东:山东大学.2017.

[11]Ivan Merelli,Federico Fornari,Fabio Tordini,Daniele D’Agostino,Marco Aldinucci,Daniele Cesini.Exploiting Docker containers over Grid computing for a comprehensive study of chromatin conformation in different cell types[J]. Elsevierjournal,2019,134: 116-127.

[12]杨保华，戴王剑，曹亚仑.Docker技术入门与实践[M].北京:机械工业出版社,2018:3-9.

[13]Marko Luksa.Kubernetes in Action[M].北京:电子工业出版社.2019:21-20.

[14]小枣菌.DevOps到底是什么意思[EB/OL].<https://zhuanlan.zhihu.com/p/91371659>.

[15]梁惠惠.对软件开发模式变迁的研究[J].现代信息科技,2019,3(22):1-8.

[16]刘博涵,张贺,董黎明.DevOps中国调查研究[J].软件学报,2019,30(10):3206-3226.

[17]牛晓玲,吴 蕾.DevOps发展现状研究[J].电信网技术,2017,10(10):48-51.

[18]李超,花磊,宋云奎.OpsFlow\_一种面向DevOps的应用自动化部署引擎简[J].计算机与数字工程,2019,47(1):190-194.

[19]付大亮.拉动DevOps持续交付的三匹马——快速交付实践总结与思考[J].应用技术,2018,3(12):58:60.

[20]高栋,王殿胜,张思琪,张晓明. DevOps平台建设分析[J].中国科技信息,2019.10(24):39-40.