软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006277]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种智能合约微服务化框架*

张富利[1,2], 侯培宇[1,2], 李杉杉[1,2], 荣国平[1,2], 李质颖[1,2], 丁梦洁[1,2]

[1](南京大学 软件学院,江苏 南京 210023)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 荣国平, E-mail: ronggp@nju.edu.cn

摘 要: 区块链具有分布式、不可篡改、去中心化、历史可追溯等特点,但难以落地.智能合约的引入有效解决了这一难题.然而智能合约的开发和运维存在部署效率低、监控工具不成熟等问题.受到 DevOps 自动化工具支持微服务持续交付、持续监控的启发,针对上述问题,本文提出了一种用于智能合约微服务化改造的框架.随后本文结合支持 DevOps 的工具设计自研原型平台 Mictract 完成智能合约的部署和监控.在 Hyperledger Fabric 官方链码 Marbles 上的案例研究表明,本文提出的框架和自研原型平台能够显著提升智能合约部署和监控的自动化水平.

关键词: 区块链;智能合约;微服务;DevOps;自动化

中图法分类号: TP311

中文引用格式:张富利,侯培宇,李杉杉,荣国平,李质颖,丁梦洁.一种智能合约微服务化框架.软件学报. http://www.jos.org.cn/1000-9825/6277.htm

英文引用格式: Zhang FL, Hou PY, Li SS, Rong GP, Li ZY, Ding MJ. A Framework to Support Refactoring Smart Contracts into Microservices. Ruan Jian Xue Bao/Journal of Software, (in Chinese). http://www.jos.org.cn/1000-9825/6277.htm

Framework to Support Refactoring Smart Contracts into Microservices

ZHANG Fu-Li[1,2], HOU Pei-Yu[1,2], LI Shan-Shan[1,2], RONG Guo-Ping[1,2], LI Zhi-Ying[1,2], DING Meng-Jie[1,2]

[1](Software Institute, NanJing University, Nanjing 210023, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Blockchain has the advantages of distribution, immutability, decentralization and traceability, but short of implementing. Smart contract is a good solution to make up for this deficiency. However, smart contracts also struggle in deploying and monitoring. Inspired by the DevOps tools that support continuous delivery and continuous monitoring for microservices, we proposed a framework to microservice-enable smart contracts. Besides, we implemented a prototype platform (Mictract) in which DevOps tools were aggregated to support smart contracts deploying and monitoring. The case study performed in Marbles of Hyperledger Fabric shows that the proposed framework and the prototype platform significantly improve the automation level to deploy and monitor smart contracts.

Key words: Blockchain;Smart contract;Microservices;DevOps;Automation

智能合约[1](Smart Contract)最早由 Nick Szabo 提出,其被定义为实现合同条款的计算机程序.智能合约允许在没有可信第三方情况下进行可信的、可追踪的、不可逆转的交易[2].受当时技术所限,智能合约很难在没有可信第三方情况下准确执行,相关的理论一直领先于应用实践.近年来,区块链(Blockchain)的迅速发展为智能合约的落地提供了可能.区块链网络具有不可伪造、无法删除、无法修改、历史可追溯等特点,是执行智能合约的理想平台.以太坊[3]将智能合约定义为区块链网络中的执行业务逻辑的代码片段,其在满足一定的限制条件后自动执行.随着区块链发展以及推广,区块链以及智能合约技术正在给金融、医疗、供应链等领域带来重大

* 基金项目: 国家自然科学基金(62072227, 61802173); 国家重点研发计划(2019YFE0105500); 江苏省政府间双边创新项目(BZ2020017); 南京大学计算机软件新技术国家重点实验室创新项目(ZZKT2019B01)

收稿时间: 2020-09-15; 修改时间: 2020-10-26; 采用时间: 2020-12-15; jos 在线出版时间: 2021-08-03

变革.然而,智能合约的开发和运维仍存在部署效率低、监控工具不成熟等诸多问题[2][4][5][6][7].典型地,目前区块链网络搭建、智能合约编写部署仍依赖手工或半自动化脚本方式.与此同时,随着 DevOps(Development and Operations,简称 DevOps)理念的兴起,DevOps 与微服务架构[8](Microservices Architecture,简称 MSA)以其快速响应需求变更、持续部署、持续交付、持续监控等特点逐步取代单体架构[9] (Monolithic Architecture),引起了很多关注. DevOps 强调使用自动化工具来更快、更频繁地交付更稳定的软件.经过不断发展,DevOps 观念不断获得认同,支持 DevOps 的自动化工具不断增多[10].这些自动化工具都能很好的支持微服务的开发、测试、部署、监控,而这种自动化工具与能力正是智能合约开发和运维过程所欠缺的.

为提升智能合约部署与监控的自动化水平,本文提出了一种智能合约微服务化框架.具体地,本文将智能合约与下层共识网络节点解耦并进行容器化编排托管给 Kubernetes 集群.解耦后的智能合约可以实现在区块链网络上的“即插即用”.智能合约经微服务化改造后,各种支持 DevOps 的自动化工具能够应用于智能合约领域,扩充智能合约开发运维工具库.最后,本文在智能合约微服务化框架的基础上结合自动化工具构建了原型平台--Mictract[1],帮助工程师从零开始搭建区块链网络、部署、操作、升级和监控智能合约.

本文第 1 节介绍背景、相关工作.第 2 节阐述智能合约微服务化框架以及改造后的智能合约开发运维流程.第 3 节介绍 Mictract 原型平台的设计与实现并且进行案例研究与分析.第 4 节讨论本文的局限性.第 5 节总结全文,并给出下一步工作展望.

1 背景及相关工作

1.1 区块链与智能合约

区块链是一种按照时间顺序链式存储数据块的特定数据结构,被视为分布式账本.如图 1 所示,当新交易生成时,交易被写入新区块.网络中的所有节点通过共识协议参与验证区块,新区块一旦被验证通过,即被追加进图 1 所示的链条且不能被删除,从而保证区块链中数据的不可伪造、不可篡改、可追溯.典型的共识协议包括工作量证明(Proof of Work,简称 PoW)、股权证明(Proof of Stake,简称 PoS)和实用拜占庭共识协议(Practical Byzantine Fault Tolerance,简称 PBFT),这些协议可以保证在没有第三方公正平台参与的情况下完成交易,从而节约交易成本.参与验证的节点被称为共识节点或矿工,每一个矿工都保存有区块链的完整副本.此外,用户使用虚拟身份进行交易,使用户隐私得到保护.如图 1 所示,每一个区块链节点都储存着一份完整的账本以及相同的智能合约.每当有新的交易请求时,区块链网络中的每个节点都会分别运行智能合约,将运行结果广播到网络中.当结果达成共识后会被更新到区块链账本内.

现有的区块链主要分为公有链、联盟链和私有链.其中,公有链去中心化的程度最强.在公有链中,全世界任何组织和个人都可以自主创建密钥对,不需经过其他人的认可即加入区块链网络,并且可以自由读取数据、发送交易、竞争记账权.公有链中的数据由大家共同记录,比特币、以太坊就是典型的公有链.联盟链的去中心化程度适中.在联盟链中,公用账本、数据只对联盟内部成员开放,新节点加入要经过联盟的认可.典型地,微软的 Bletchey 以及 IBM 开源的 Hyperledger Fabric 主要侧重支撑构造企业联盟区块链系统.私有链是完全中心化的区块链.私有链由特定组织维护,只有被授权的节点才能操纵数据.

[1] https://github.com/doporg/Mictract.git

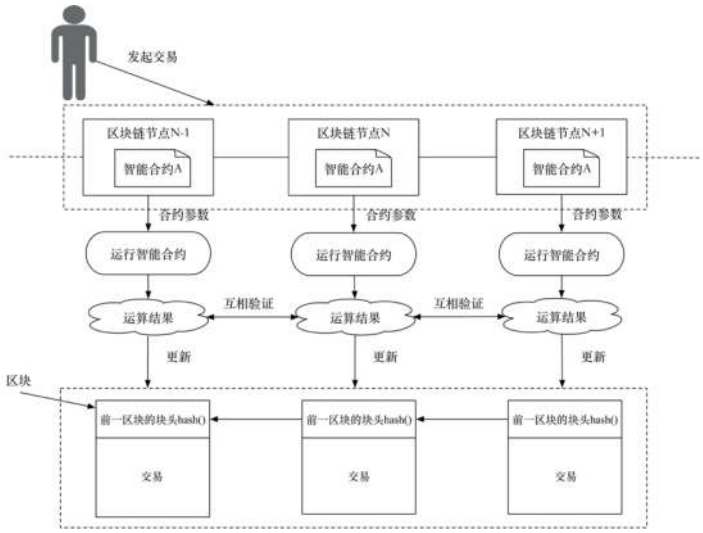

Fig.1 The schematic of Blockchain

图 1 区块链示例图

自以太坊白皮书与黄皮书发布以来,智能合约被描述为图灵完备的[11]、部署于区块链网络中的合同条款代码.图灵完备意味着智能合约可以执行复杂逻辑.部署于区块链网络意味着传统的合同条款可以进入到实体计算机中,并在区块链网络去中心化、不可伪造、不可篡改的特性下执行.智能合约由区块链中参与该合约的多个用户共同编写,智能合约明确参与者的权利与义务.智能合约由区块链全部网络节点共同参与和监督,不可被删除,在满足触发条件后自动触发执行.目前,不同区块链网络智能合约开发语言并未统一,常见的,如以太坊的Solidity、Hyperledger Fabric 的链码(chaincode).

1.2 智能合约研究现状

近年来,智能合约极大地丰富和扩展了区块链应用场景,形成了各式各样的去中心化应用(Decentralized application,简称 Dapp).在应用场景方面,McCorry 等人[12]将区块链应用在电子投票领域,实现了一个基于以太坊的去中心化互联网公开投票协议.这是第一个实现的不依赖任何可信的权威机构来计数和保护选民隐私的去中心化应用.Tonelli 等人[13]结合了智能合约与微服务的相似性,提出了一种架构方法,给出了一个电子商务平台实现的例子.该例用智能合约将微服务体系结构进行重构,每个微服务的角色和服务运行到以太坊区块链上.随后,Tonelli 等人[14]进行了一个案例研究,将微服务架构环境通过一组等效的智能合约进行复制和实现,证明了利用智能合约实现一个简单的基于微服务的系统是可行的.Zhang 和 Wen[15]提出了一种物联网电子商务模型,旨在重新设计传统电子商务模型中的许多元素,并借助区块链和智能合约技术在物联网上实现智能财产和付费数据的交易.Chang 和 Chen[16]在供应链领域进行了系统文献综述,表明传统的供应链活动涉及多个中介、信任和性能问题,利用区块链技术的潜力重建传统供应链商业的运作的方式,以便获得更好的性能、分布式治理和过程自动化.Leka 等人[17]相信区块链技术将是下一个技术革命,同时表明区块链研究现阶段在物联网[18]、医疗、教育、政府各个领域都有涉及.

目前,区块链应用尚处于初级阶段.研究的重点是利用区块链技术特性应用在不同领域[17],然而区块链应用仍然未大规模落地.应用区块链技术,很大程度上就是利用区块链编写智能合约[6].然而,智能合约的开发属全新的开发领域,业界没有形成标准的开发流程.教育领域实践经验少[19],使用门槛高.

其次,当前业界对于智能合约的部署研究探索较少,缺乏提高部署效率、缩短产品交付周期的有效工具与框架[20].智能合约的部署通常采用的纯手工或脚本方式[21][22],消耗大量的时间成本,给智能合约的部署和升级带来了严重的负担.由于智能合约的不可修改性,一旦区块链上的智能合约出现漏洞,只能升级智能合约.另一

方面,集成开发环境(Integrated Development Environment,简称 IDE) 对错误信息的支持差[7]使得智能合约调试困难.工程师开发无漏洞智能合约具有挑战性.同时部署升级效率的低下使得无法及时更新链上存在漏洞的智能合约.这些漏洞一旦被利用就会给客户带来财产损失,动摇客户对智能合约的信心[23].

此外,当前系统缺乏一套涵盖不同层面的标准方法来监控区块链系统[5],尤其是智能合约层面.业务逻辑中的请求、交易涉及多个智能合约,这些智能合约通常由多个团队开发.当出现问题或者性能不佳的时候,不仅执行过程无法中断而且缺乏错误日志[24],这使得开发人员很难从错综复杂的服务调用网络中找到问题根源,故障难以检测、定位和修复.智能合约与共识节点耦合,对于智能合约的运行状态、消耗物理资源的情况现阶段不可知.

综上,智能合约学习使用门槛高、部署效率低、监控标准不完善,严重限制区块链技术的推广使用和大规模落地.目前,亟需一体化、自动化的解决方案.

1.3 智能合约微服务化探索

为解决以智能合约为底层技术的区块链应用的一体化和自动化部署问题,学术界和工业界都进行了积极的探索.Porru 等人[23]将区块链实现的软件定义为“面向区块链的软件(Blockchain-Oriented Software,简称 BOS)”,并指出需要为区块链应用设计开发相应工具.考虑到区块链应用的独特性,Porru 认为软件工程师可以从 BOS 开发实践应用中受益,这些实践将构成“面向区块链的软件工程(Blockchain-Oriented Softare Engineering,简称 BOSE)”.随后 Porru 总结了 BOSE 的挑战及其原因,提倡重点关注的大型团队之间的协作、测试活动和用于创建智能合约的专门工具.虽然,区块链即服务[25] (Blockchain as a Service,简称 BaaS)平台白皮书[1]以云计算为基础给出了通用 BaaS 平台的解决方案,但很难有效利用已有工具.

区块链应用的开发与运维涉及全栈技术,但大多数工程师只专注于一个层次的专业知识,鲜有熟练的全栈工程师[26],采用区块链领域的新工具会给工程师们带来学习负担.以微服务为核心的 DevOps 在持续协作、持续测试、持续集成、持续交付、持续监控等方面吸引了越来越多研究者与实践者的注意[5],自动化工具也在寻求在其他领域的应用[10].如表 1 所示,微服务与智能合约在设计原则上有很大的相似性[6][27].微服务作为一组独立自治的小型应用程序,业务目标明确且定义良好,它们很好地实现了模块化结构;智能合约也会根据业务需求进行良好的定义并实现简单的功能,每个智能合约都运行在各自环境中,具有高度的模块化,通过特定的通信方式与区块链网络进行通信.这种相似性为智能合约微服务化改造提供了可能.

Hu 等人[28]提出一种建立智能合约的微服务化方法,提出将智能合约容器化,并将其发布在 Kubernetes 集群中,使其满足可组合性、可扩展性,但是这种方法停留在理论阶段,无法对区块链各网络节点以及智能合约容器进行统一的管理.Wang 等人[29]将基于区块链智能合约与云技术相结合,研究智能合约的可扩展性和性能问题,提出了一种新型的智能合约体系结构,并对其各层的关键技术进行了研究,Wang 等人的实验部分针对的是智能合约的可扩展性与性能,并未提升智能合约部署、监控自动化水平.Sousa 等人[30]提出了一种基于微服务和区块链技术的公证机构,使公证处和其他机构能够结合起来,确保各方之间信息交流的安全和迅速.并详细介绍了一种针对所提议的业务模型的框架.该框架将核心出生证书文件上链,其他业务场景在链下执行,并使用自动化工具进行构建.Li 等人[5]进行了典型案例研究,系统分析了当前区块链应用技术领域存在的挑战以及可能的解决方法,并给出了区块链 DevOps 化、区块链微服务化的研究方向.

受 Li 等人区块链微服务化工作的启发,针对目前智能合约开发、部署以及监控等环节自动化工具不健全等问题,本文提出一种智能合约微服务化框架.基于该框架构建起的支持平台 Mictract 能够有效整合各类支持 DevOps 的自动化工具,提升智能合约开发运维的自动化水平.

[1] http://www.caict.ac.cn/kxyj/qwfb/bps/201901/t20190111_192631.htm

Table 1 The design principles of microservices and smart contracts

表 1 微服务与智能合约设计原则

设计原则	微服务	适用智能合约	理由
高内聚低耦合	服务体量小 单一性职责 轻量级通信	√	智能合约拥有特定的范围和职责
高度自治	独立开发 独立部署 独立发布 进程隔离	√	智能合约独立运行在沙盒环境中
分布式 去中心化	分布式体系结构 无中心化数据库	√	智能合约部署在分布式网络中
以业务为中心	微服务即代表业务 快速响应业务变更 根据业务组织团队 团队仅对业务负责	×	智能合约无法快速响应业务变更, 团队需要兼顾智能合约开发运维和区块链网络维护

2 智能合约微服务化

2.1 智能合约微服务化框架

针对第 1 节中智能合约在开发运维中存在的问题,本节重点探寻智能合约微服务化改造并利用支持 DevOps 的自动化工具简化智能合约的开发运维工作,提升智能合约开发运维的效率.目前,SpringBoot[1]是构建微服务应用的流行框架.SpringBoot 的部署方式主要分为两种:(1)Java 档案文件(Java Archive File,俗称 jar 包)形式,开发者直接将 jar 包运行在服务器的某端口上对外提供服务;(2)容器化形式,开发者将 jar 包进一步打包成容器,然后将该容器部署于集群,由集群管理容器并对外发布服务.如图 2 所示,展示了采用微服务容器化部署的流程.该流程拥有多种可选的自动化工具.

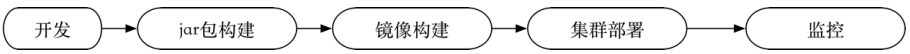

Fig.2 The deployment of microservices

图 2 微服务容器化部署

由于智能合约与微服务在设计原则上的相似性,将智能合约进行解耦与容器化打包是智能合约利用支持 DevOps 的自动化工具的关键.如图 1 所示,智能合约运行在区块链网络中的节点中,智能合约与节点是强耦合关系.这种强耦合关系使我们很难利用现有工具单独部署、监控智能合约.为此,本文提出智能合约微服务化框架,如图 3 所示.

[1] https://spring.io/projects/spring-boot

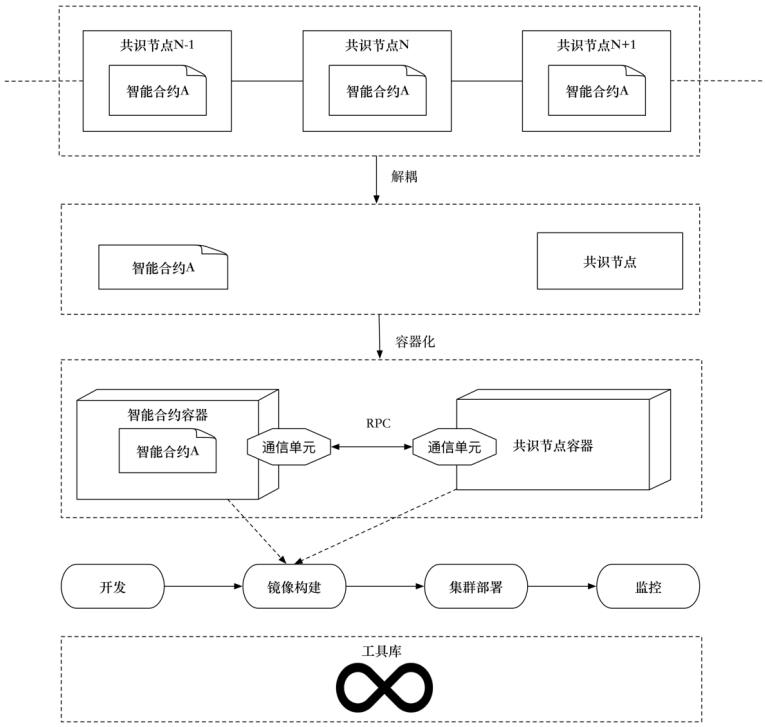

Fig.3 The framework of microservices for smart contracts

图 3 智能合约微服务化框架

(1) 容器化

容器化技术有效隔离了资源,不仅能提高应用的安全性和可移植性,而且能够加速应用管道自动化和应用部署.智能合约微服务化的目的是解耦,让系统更容易拓展、更富有弹性.将区块链网络的共识节点及智能合约解耦并以容器化的形式打包交给集群统一管理.与区块链网络的节点解耦后,智能合约变成无状态.智能合约容器提供智能合约的运行环境,对外提供更加明确的接口,对网络节点提供更便捷的插拔式服务.合约本身并不提供数据的维护管理,不关心数据的一致性、网络安全性等问题,分散、冗余且经过验证的数据管理交给下层网络节点.

本文将智能合约和下层网络节点进行容器化改造,具有以下优势:

- 单一职责:每个智能合约容器仅需完成自身业务功能,智能合约独立开发、独立运营、独立部署、独立升级;
- 高可扩展性: 智能合约及其网络节点进行容器化编排后,对内封装编程语言,开发者可以自由选择编程语言;对外暴露服务,当有新的智能合约需求或者对区块链网络要求升级时,仅需向集群内部增加新的智能合约或者网络节点容器;
- 高可用性: 集群层面的高可用性策略可以应用于区块链领域,增加区块链层面的的高可用性.

值得注意的是,智能合约微服务化需要额外考虑区块链网络与微服务架构在数据处理响应时间方面的区别.链上新数据的更新要经过多个节点同时需要消耗大量的计算资源,限制了区块链网络每秒处理事务的能力,影响了区块链网络的响应时间.比特币平均每秒执行 7 笔交易,以太坊平均每秒执行 15 个事务,虽然优化的 Hyperledger Fabric 可以达到每秒 20000 交易的吞吐[31],但这样的表现与微服务架构的每秒百万事务相比不可接受.所以,对于大流量数据与实时性要求比较高的数据并不建议使用链上代码处理.

(2) 通信方式

现阶段智能合约在业务上的是高内聚的,智能合约之间调用的情况并不多见,更常见的是解耦后的智能合约与下层网络节点之间的通信调用.为了实现智能合约的可扩展以及高利用率,就要在智能合约与下层网络之间建立良好的通信机制.智能合约容器与网络节点彼此有效通信是智能合约部署运行成功与扩展的基础,有效的通信方式才能保证智能合约微服务化的运行正确性.

如表 2 所示,目前主要有两种方式实现微服务间的通信,一种是使用远程过程调用(Remote Procedure Call,简称 RPC)的框架,另外一种是通过 HTTP 协议定义的表述性状态传递[32] (Representational State Transfer,简称 REST).RPC 在性能上表现出色,区块链网络中通常使用 RPC 作为 P2P 节点通信.而 REST 虽然在性能上不如 RPC,但是 REST 接口在理解、测试、调用方面更加简单方便.对区块链网络之外提供智能合约服务时,建议使用 REST 风格接口.

Table 2 The comparison of two communications

表 2 通信方式对比

	远程过程调用	表述性状态传递
场景	调用者只需要函数运算的结果，却不需要实现函数的具体细节，主要用于函数方法的调用	适合对外部提供资源接口时使用
优/缺点	性能表现出色，但耦合度较强	耦合度低，在理解、测试调用方面更加简单

2.2 智能合约微服务开发运维流程

本节希望为智能合约微服务化提供一套从需求、开发、构建、测试、部署到监控的流程指导.如图 4 所示,本文设计了智能合约微服务化通用开发运维流程.图 4 在已经搭建好的区块链网络中,使用现有 DevOps 自动化工具无侵入的支撑智能合约完整生命周期.本文目前已完成的工作在图 4 中使用六边形边框,椭圆边框部分将在未来工作中进行补充.

(1) 需求

需求收集与分析是智能合约微服务开发流程的开端.智能合约微服务化的目标是解耦,让系统更容易拓展、更富有弹性.虽然智能合约可以省去许多中间业务场景,简化操作流程、节约成本,但受区块链技术本身的特性所限,并不支持全部业务上链.在考虑区块链性能的同时,可以结合领域驱动设计(Domain-Driven Design，简称 DDD)[33],技术人员与业务人员共同协作完成智能合约与微服务划分.

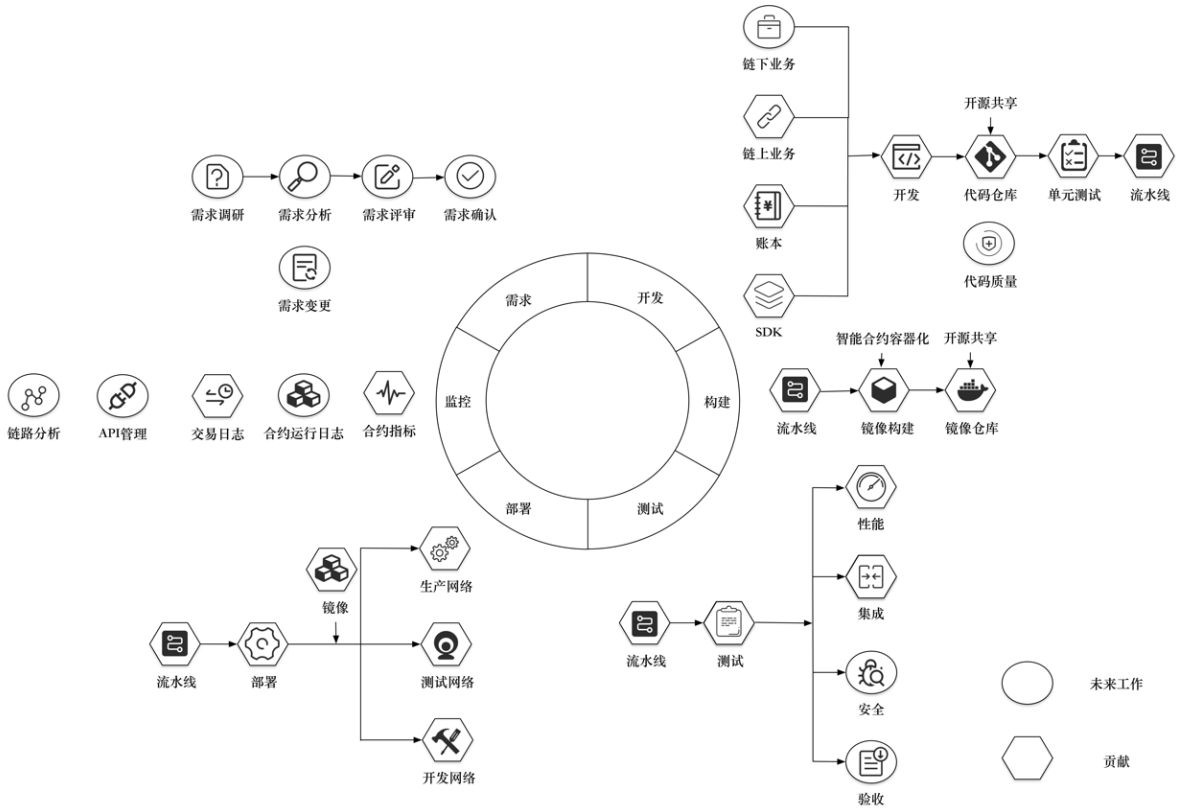

Fig.4 The process of developing and operating microservices for smart contracts

图 4 智能合约微服务化开发运维流程

(2) 开发

开发过程中,智能合约与其他业务代码隔离,彼此之间可以使用不同的编程语言.可以将智能合约开源,帮助工程师更快了解智能合约内部的工作原理.开发的代码仓库可以对接持续集成(Continuous Integration,简称 CI)工具(例如 Jenkins、Travis CI 等),做到合并代码后自动提交部署.

(3) 构建

智能合约的开发和升级过程都比较繁琐,所以智能合约的手动部署、版本更迭以及升级过程都容易出错,这有可能会导致系统漏洞,甚至升级失败,造成数据和经济损失.利用已有 CI 工具实现基础设施自动化,利用容器化工具(如 Docker)打包编写完成的智能合约,衔接测试和部署.同样,将合格的智能合约镜像开源可以方便其他开发者直接部署于自己的区块链网络.

(4) 测试

常见的智能合约漏洞包括重入攻击[34]、事务顺序依赖、时间戳依赖、处理异常、整型溢出等.尽管业界已有各种理论与工具[35][36],但自动化程度较低.本文利用持续集成工具提升智能合约测试的自动化水平,将“可插拔”、“开箱即用”的测试策略配置进入持续交付流水线.以便在智能合约上链前对其进行全方位测试.

(5) 部署

本文提倡从一开始不断的集成,将拆分后的智能合约以及区块链共识节点组合起来.每个过程都由自动化的构建(包括测试)进行验证,以尽可能快地检测集成的错误.智能合约可交付镜像可以在开发、测试、生产环境

之间无障碍流通,打破三者之间的隔阂.开发完成的镜像可以直接交给测试网络测试,测试完成的镜像可以直接在生产环境中部署.

(6) 监控

合约指标: 在智能合约微服务运行时采用微服务监控领域相结合的手段与工具(例如 Hyperledger Caliper、Promethus 等)全面采集智能合约容器以及下层网络节点的性能指标数据,提供一个标准和定制的性能框架.该框架可以集成数据异常检测[37]、报警及时反馈和智能决策分析,提供有效的可视化.可以监控的指标有:

- 响应时间: 每个事务的响应时间,包括平均响应时间、最大响应时间和最小响应时间;
- 吞吐量: 每秒成功事务的数量;
- 成功率: 成功交易的数量与全部交易数量的比值;
- 资源消耗: CPU、内存、磁盘和网络 I/O 等.

容器运行日志: 目前智能合约开发领域对日志采集分析几乎没有涉及,但微服务架构在大规模分布式系统中的日志采集、收集、分析方面已经做得十分成熟,可以借鉴微服务架构中的工具(例如 EFK、ELK),进行简单的配置,全面采集智能合约容器运行产生的日志,提高智能合约运行过程中对于日志的实时采集、分析、展示、以及故障定位的能力.

交易日志: 智能合约的运行日志外,用户的交易行为、资金和资产的流向以及流通过程,蕴藏大量有价值的信息.可以进行交易性日志的记录与追踪.

API 管理: 结合集群网关管理工具不用考虑安全控制、流量控制、审计日志等问题,网关层还可以提供 API 发布、管理、维护等主要功能,开发者只需要简单的配置操作即可把自己开发的合约发布出去,同时置于网关的保护之下.

链路分析: 智能合约微服务的全链路调用分析,借鉴 Google 的 Dapper[38],在软件和通信协议中注入探针,跟踪运行时智能合约微服务的有效调用路径及响应时间.帮助理解系统行为、用于分析性能问题的工具,以便发生故障的时候,能够快速定位和解决问题,在出现性能瓶颈的时候,精准合理地扩容.

3 智能合约微服务化开发运维原型平台的设计与实现

本节为第 2 节提出的智能合约微服务化框架与流程提供一种可行的支撑平台,使用三台 Centos 7.x 组成的 Kubernetes 集群,一台 Mater 节点(dop-node1),两台 Node 节点(dop-node2、dop-node3).Master 节点作为 NFS 服务端.公有链允许任何节点随意进出的特性无法适用于企业级区块链应用[39],本文将使用 Hyperledger Fabric 进行智能合约微服务改造,这可以避免了公共区块链的运营成本,并确保更好地控制隐私[13].

3.1 原型平台

Mictract 作为 BaaS 平台能够为构建区块链应用程序的人或组织提供基于云的第三方服务,允许用户在云平台上构建、托管、运行、监控自己的区块链应用,其架构如图 5 所示.UI 层使用 React 和 ICE 实现,为用户提供简单易操作的可视化界面.用户可以在网络管理页面根据业务与组织需要搭建定制化的网络,在通道管理页面对搭建的网络进行通道配置,在链码管理页面为新建的通道自动化安装链码、升级链码以及操作链码,在安全管理界面对链码进行自动化单元测试、性能测试,在监控管理与日志管理界面对运行的链码与网络进行各种性能指标数据的收集,同时收集账本信息.账本信息中包含当前通道的区块信息以及交易信息.业务层通过 SpringBoot 实现,暴露 API 以支撑上述操作.工具层利用 DevOps 以及 Hyperledger Fabric 领域相关的工具对上述网络搭建、自动化部署、安装、升级链码、监控与日志收集、镜像容器的打包存储提供底层工具支持.持久层利用 NFS 共享证书、账本文件.最后整个平台与部分工具部署于 Kubernetes.Mictract 将上述功能与工具形成工具链支撑智能合约微服务化的开发运维工作.

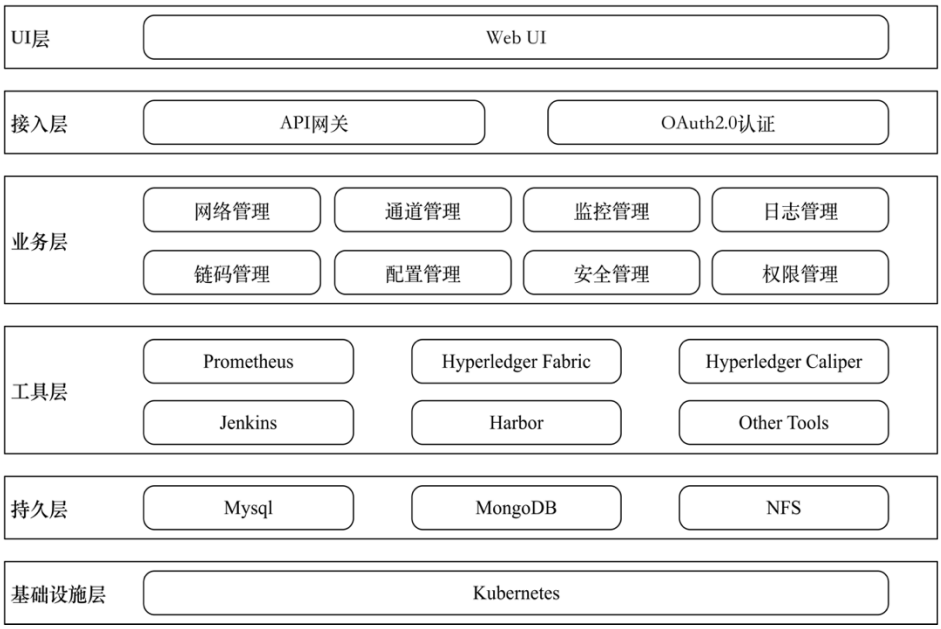

Fig.5 The design architecture of Mictract

图 5 Mictract 体系结构

伪代码 1 智能合约-Dockerfile

传入: 智能合约代码

结果: 暴露端口提供服务

```
1: FROM <image:tag>
2: COPY <src> <dest>
3: WORKDIR <dest>
4: RUN <build-command>
5: USER <uid>
6: EXPOSE <port>
7: CMD <run-command>
```

伪代码 2 外部链码-Kubernetes deployment

传入: 链码镜像地址，命名空间，链码包ccid

结果: 暴露端口提供服务

```
1: namespace <namespace>
2: containers:image <image-uri>
3: image:imagePullPolicy <IfNotPresent>
4: env:CHAINCODE-CCID <ccid>
5: env:CHAINCODE-ADDRESS <defaultAddress>
6: container:port <defaultPort>
```

将官方 Hyperledger Fabric 的网络节点镜像以及链码打包成 Docker 镜像并将这些镜像在 Kubernetes 集群中成功部署是智能合约微服务化的基础也是阻碍智能合约微服务化的主要问题.Hyperledger Fabric 2.0 引入了链码新生命周期,链码包的格式从序列号协议缓冲消息变为了由 gzip 压缩的 POSIX tape 归档,通过新生命周期命令可以打包新类型链码包,并生成一个链码包的 id(CCID),作为环境变量注入到 Kubernetes 的 pod 容器中.利用伪代码 1 将链码打包成 Docker 镜像.当需要部署链码时,利用伪代码 2 所示配置链码的部署文件,拉取链码镜

像并为其配置环境变量(如 CCID,容器内链码运行地址),最终生成相应的 Kubernetes pod 容器作为独立的运行单元.

如下表 3 所示,将官方给出的搭建网络、外部链码部署教程进行总结与 Mictract 启动网络、部署链码原理进行对比.智能合约容器化改造过程存在三个难点:(1)Kubernetes 部署 Deployment 时,Kubernetes 在非人为指定下会选择出集群中适合运行当前 pod 资源的一个节点,这种资源调度策略很可能使 pod 与证书文件、初始区块或者通道文件处于不同节点.所以采用 NFS 文件服务器的方式将上述配置文件共享;(2)官方教程启动容器镜像需要注入特定的环境变量以及配置文件.Mictract 将环境变量以及共享的配置文件以 Deployment 容器变量的形式传入;(3)将 peer 声明为外部链码构建形式并为其传入外部链码构建器启动器.同时,将 peer 节点生成外部链码标识符传入外部链码的 Deployment 容器.

Table 3 The comparison of two kinds of deployments

表 3 两种部署方式对比

步骤	官方部署	Mictract 启动网络、部署链码原理
S1	创建证书,生成证书文件; 创建创世区块,生成创世区块文件; 创建通道,生成通道文件;	创建证书,生成证书文件; 创建创世区块,生成创世区块文件; 创建通道,生成通道文件; NFS 共享上述配置文件;
S2	修改节点 Dockerfile 挂载步骤 S1 生成文件;	生成 peer 节点 yaml 文件,包含 Deployment 和 Service,配置 Docker 启动环境变量; 修改 yaml 挂载步骤 S1 生成的文件;
S3	打包链码; 修改 peer 节点 Dockerfile 文件声明为外部链码; 修改 peer 节点 Dockerfile 文件挂载外部构建器启动器脚本; 启动 peer 节点;	修改 peer 节点 yaml 声明为外部链码形式; 修改 peer 节点的 yaml 挂载外部构建器启动器脚本; 启动 peer 节点的 Deployment 与 Service;
S4	依次在每个 peer 内部将所属 peer 加入步骤 S1 创建的通道;	在 master 节点中将 peer 节点加入步骤 S1 创建的通道;
S5	为 peer 安装外部链码,生成外部链码标识符; 修改链码代码,添加外部链码标识符; 部署链码;	为 peer 安装外部链码,生成外部链码标识符; 修改链码代码,使其从环境变量中读取外部链码标识符; 为链码构建 Dockerfile,将其打包成 Docker 镜像; 为链码构建 yaml 文件,将外部链码标识符作为环境参数传入; 启动链码 Deployment 与 Service;
S6	实例化链码; 链码操作;	实例化链码; 链码操作;

在测试方面, Mictract 提供对 Go 语言编写链码的测试.Mictract 借鉴 Hyperledger Fabric MockStub 类提供的 MockInit 函数在测试环境下调用智能合约中的 Init 接口,完成账本初始化工作. Mictract 采用 MockStub 类调用智能合约的 Invoke 接口进行单元测试.Mictract 借助 MockStub 类完成智能合约的性能测试,利用 Go 语言提供 pprof 工具完成性能剖析工作.pprof 工具通过采集测试过程中的数据信息,得到一组 profiling 数据.profiling 包含 cpu profiling、heap profiling 以及 block profiling 三种性能指标.cpu profiling 指标用于标识测试过程中使用 cpu 最多的函数,通过对 cpu 上执行的线程进行定期中断来记录一个性能剖析事件;heap profiling 指标用于标识分配内存最多的语句;block profiling 指标用于标识阻断协程时间最长的操作.性能测试结果最终被写入一个文件中,利用 pprof 工具和 GraphViz 工具可以将测试结果转换为 svg 的可视化图,直观的展示智能合约的性能消耗情况.

Mictract 目前主要对运行中的智能合约容器进行两方面监控:(1)资源消耗,利用 Prometheus 对智能合约及

其网络节点容器进行资源节点收集.采用 Prometheus 提供的强大的查询能力以及灵活查询接口节点的资源消耗如:CPU、内存、磁盘、网络 I/O 进行收集,时刻监控智能合约的运行状态;(2)对于交易过程监控,集成 Hyperledger Fabric 原生采集区块信息的方式,查询链上区块信息,如:区块高度、某区块具体信息、某区块包含的交易等.

3.2 案例研究

为了验证本文提出的智能合约微服务化框架及原型平台 Mictract 可行性.本文需要选取合适的案例,使用 Mictract 对其进行托管、运行、监控,并保证其运行的正确性.本文最终选取官方给出的示例链码 Marbles[1]进行操作,其支持 levelDB 的功能如下表 4.

Table 4 The functions of Marbles

表 4 Marbles 链码功能展示

编号	名称	参数	作用
F1	initMarble	marbleName, color, size, owner	为 owner 分配一颗名为 marbleName 大小为 size 颜色为 color 的弹珠
F2	transferMarble	marbleName, newOwner	将名为 marbleName 的弹珠移交给 newOwner
F3	transferMarblesBasedOnColor	color, newOwner	将颜色为 color 的全部弹珠交给 newOwner
F4	delete	marbleName	销毁名为 marbleName 的弹珠
F5	readMarble	marbleName	获取 marbleName 的相关信息
F6	getMarblesByRange	marbleName1, marbleName2	获取名称位于 marbleName1 与 marbleName2 之间的弹珠情况
F7	getHistoryForMarble	marbleName	获取 marbleName 的资产转移情况

本文选择了 5 名志愿者使用 Mictract 搭建区块链网络并部署链码.选取的 5 名志愿者具备基本软件开发的经验和能力,且有区块链网络搭建和区块链应用的开发经验.由于目前基于 Hyperledger Fabric 的开源 BaaS 平台基本处在研发当中(例如 Cello[2]),功能都不完备.所以本次案例研究仅考虑官方推荐的脚本方式部署以及 Mictract 对区块链网络及链码分别部署、升级.本文考虑并排除了一些影响公平性的因素,如:在实验中使用相同的机器并且 Hyperledger Fabric 所必需的网络节点、证书节点、客户端节点镜像提前下载完成.本文希望志愿者们自由搭建不同规格的区块链网络,但规定每一个网络只创建唯一通道.通道(Channel)是链码运行的具体环境,相同的通道就意味着拥有相同的账本.这 5 名志愿者中最终使用 Mictract 搭建了 5 种不同结构的区块链区块链网络,网络规格如表 8 所示.

志愿者操作 Mictract 过程如下:(1)利用网络管理的界面搭建网络.志愿者们可以在网络管理界面定制不同规格的网络结构,即选取不同数量的 order 节点个数、组织(Org)个数、peer 节点个数.(2)利用通道管理界面创建通道.志愿者们需选择要创建通道的网络及相关节点.(3)利用链码管理界面安装链码.本文按照表 2 中 S5 的描述修改 Marbles 链码并编写好 Dockerfile 与部署的 yaml 模版,将其推送至代码托管平台,并将镜像地址交给志愿者们.在链码管理界面,志愿者们只需要选择需要安装链码的通道并填写好链码地址就即可完成链码的安装.

图 6 所为志愿者 A 搭建的区块链网络架构图,该网络拥有 2 个 order、2 个组织、2 个 peer 节点.每个 peer 节点都含有一个相同的名叫 mychannel 的通道,志愿者 A 将 Marbles 链码安装在该通道内.表 5 为志愿者 A 在网络搭建完成后 Kubernetes 集群所拥有的 Pod 情况.由于表格长度限制,故将信息简略描述.为体现

[1] https://github.com/hyperledger/Fabric-samples.

[2] https://github.com/hyperledger/cello.

chaincode-marbles-org1 的高可用,表 5 中将 chaincode-marbles-org1 设为 3 个副本.Pod 是 Kubernetes 系统的基础单元,是用户所创建或部署的最小组件.表 5 中,“pod name”描述的 Pod 的名称;“ready”表示准备好的副本数量;“status”表明 Pod 所处的生命周期,“Running”表示该 Pod 已经被成功调度运行;“node”表示 Pod 所在的集群节点的名称,Pod 所在 dop-node2、dop-node3 两个节点是随机的.

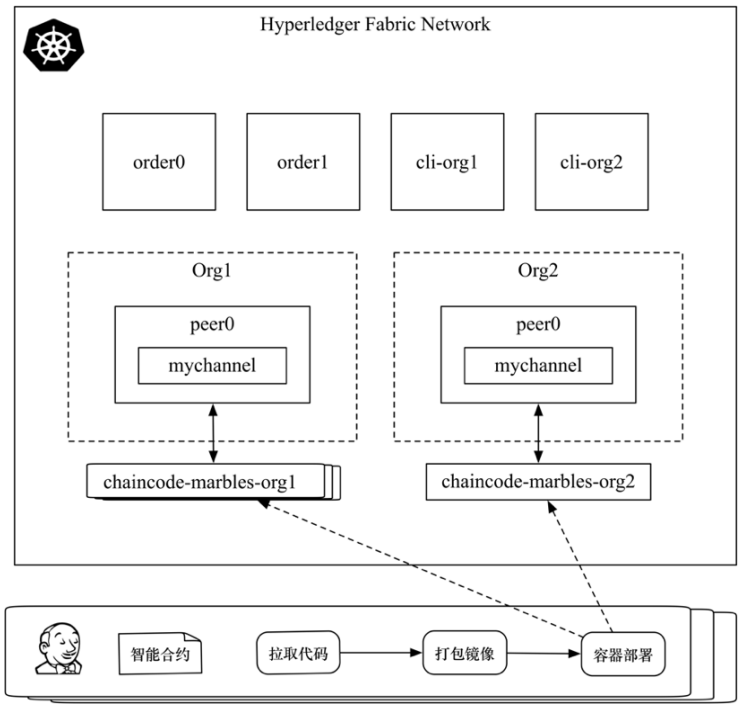

Fig.6 The deployment architecture of Marbles

图 6 Marbles 部署架构

Table 5 The pods in Marbles network

表 5 Marbles 网络的 pod 展示

pod name	ready	status	node
ca-org1	1/1	Running	dop-node3
ca-org2	1/1	Running	dop-node2
chaincode-marbles-org1	1/1	Running	dop-node3
chaincode-marbles-org1	1/1	Running	dop-node2
chaincode-marbles-org1	1/1	Running	dop-node2
chaincode-marbles-org2	1/1	Running	dop-node3
cli-org1	1/1	Running	dop-node2
cli-org2	1/1	Running	dop-node2
order0	1/1	Running	dop-node2
order1	1/1	Running	dop-node2
peer0-org1	1/1	Running	dop-node3
peer0-org2	1/1	Running	dop-node3

Mictract 可以对运行时链码进行操作,并且每当执行一次交易 Mictract 能同时捕获到当前通道中的区块信息以及交易性数据,不需要专业人员参与,这大大降低了使用智能合约的使用门槛.表 6 为 Mictract 日志管理展示的部分区块链块头信息.图 7 展示的为 prometheus 监控上述在 Kubernetes 部署的 chaincode-marbles-org2 容器 CPU 使用率.

Table 6 The header of Blockchain

表 6 区块链块头信息

number	previous_hash	data_hash
4	Mp35UXUZ+1/rAE0QF0GEVU noDadYDhCnA5KXtmRtXkQ=	p9kYGq5KNiE1V1lzbkIQXD gJlfgkF95qkWYcVE5G2l8=
3	qBmKLqlAc/Mf6xwkDkPw0E JfbSVbT9c6I0yYOtuMzi8=	4YrDS9AAi6Ej9pZuxP1iw6 lI/J3RQICMUetCBet1aRU=
2	YzkoeQNiDI9hWenYFJPb/cN UF9MizCMK7mK6fuw6Asg=	X7cfTkm9puNOpZ5rJiWRTU W4R/SCSvnvm3vo/suBCfE=
1	mL0WJIrLUrL0XujuVMxV03 LXK4fLcmiLBKO1gtci+NQ=	HG6laE2CUPUY+gANGzFzPAj 4ScluSPUcpM3YRs/cYmc=
0	null(创世块)	oV8RSAFEaJl8h827+21IlivM SuRm3p+q/r+FU5SwJL8=

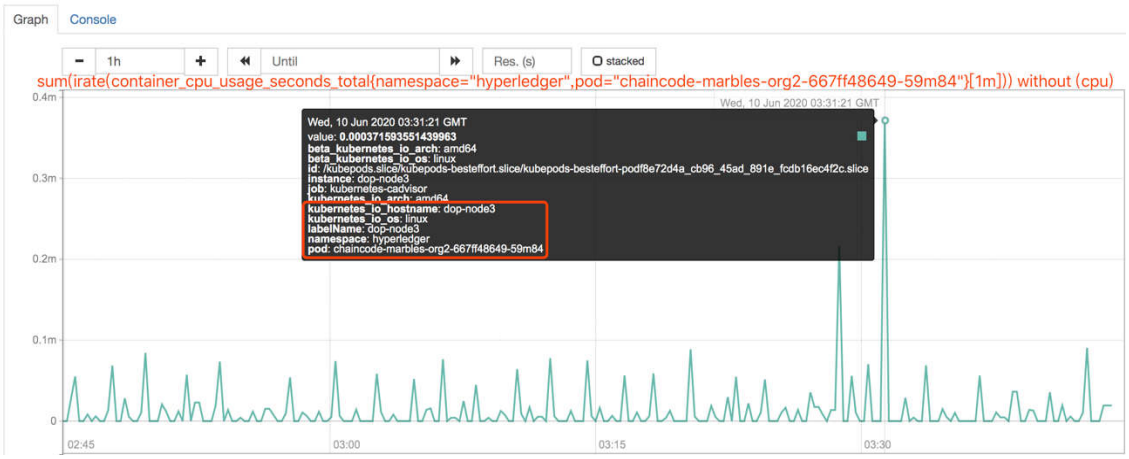

Fig.7 The CPU's usage of chaincode-marbles-org2 container

图 7 chaincode-marbles-org2 容器 CPU 使用率

3.3 结果分析

本节旨在从两个方面对所提出来的智能合约微服务化框架及原型平台 Mictract 进行分析.分别是可行性与部署效率.

(1) 可行性

可行性是本文首要关心的问题,智能合约微服务化框架需保证智能合约运行无误.为此本文将 Marbles 进行微服务化改造,验证其是否满足预期功能.操作过程如下表 7 所示,表 7 为表明智能合约微服务化框架确保智能合约正确运行,并不是为了展示 Marbles 链码本身的功能,故未进行全部功能的调用.结果表明,智能合约微服务化改造满足预期,可以保证智能合约运行的正确.

(2) 部署效率

目前,缺乏智能合约微服务化的有效评估指标.由于本文的目的是利用已有的工具定制化区块链网络、快速部署、操作、监控智能合约,避免手工及脚本方式,以此来提升智能合约开发运行自动化水平.故本文对选取部署效率作为分析指标.参与案例研究的 5 名志愿者已经熟练掌握并能够修改 Hyperledger Fabric 部署网络所

用到的脚本文件.表 8 展示了 5 名志愿者搭建网络及其用时的情况.由于 peer 节点的数量、组织的数量以及通信效率的影响,同一规格的网络使用 Mictract 部署网络的时间可能会略有差别.在环境完备且中间不出现 Bug 的前提下采用脚本方式比 Mictract 方式平均多使用 225.6 秒.对于刚接触 Hyperledger Fabric 的工程师而言,理解修改 Hyperledger Fabric 网络配置文件启动网络往往需要一周左右的时间.从结果得知,与官方的部署方式相比,Mictract 具有对于入门区块链的工程师而言具有极强的易用性,对于经验丰富的工程师而言部署效率也能得到提升.

Table 7 The operations of Mabrles by Mictract

表 7 Mictract 操作 Mabrles 过程

步骤	操作	结果	满足预期
S1	{"Args":["initMarble","marbleDop","blue","20","Mictract"]}	INFO001 Chaincode invoke successful.result:status:200	√
S2	{"Args":["readMarble"," marbleDop "]}	{"docType":"marble","name":"marbleDop", "color":"blue","size":20,"owner":" Mictract "}	√
S3	{"Args":["transferMarble","marbleDop","test"]}	INFO001 Chaincode invoke successful.result:status:200	√
S4	{"Args":["readMarble"," marbleDop "]}	{"docType":"marble","name":"marbleDop", "color":"blue","size":20,"owner":" test "},	√
S5	{"Args":[" getHistoryForMarble"," marbleDop "]}	[{"TxId":" baa15e16e6461a4c0f35371c3d3caf124c 898155325f0 f923ac736c52890ade4", "Value":{"docType":"marble","name":"marbleDop", "color":"blue","size":20,"owner":"Mictract"}}, {"TxId":"5c2f405889eaee25d71d36eb3d35420af83 3a20755d581eba2f53a3eefd", "Value":{"docType":"marble","name":"marbleDop", "color":"blue","size":20,"owner":"test"}}, {"TxId":"8dde5231945e56a89262555e04f14788e8 d98e33e28b1032d8132eef366247ff", "Value":}]	√

Table 8 The results of experiments

表 8 实验结果

志愿者	总 Peer 数量	Order 数量	组织数量	脚本方式	Mictract
A	2	2	2	5 分 42 秒	2 分 16 秒
B	4	3	2	7 分 20 秒	3 分 05 秒
C	2	3	2	5 分 58 秒	2 分 26 秒
D	3	4	3	8 分 03 秒	3 分 22 秒
E	1	2	1	5 分 06 秒	2 分 12 秒

4 局限

本文在三台 centos7.x 组成的 Kubernetes 集群上搭建了 Mictract 平台并对智能合约微服务化框架进行了初步的验证.虽然 Mictract 可以在智能合约正确运行的前提下提升智能合约的部署效率,但是 Mictract 尚有诸多的不足之处.(1)缺乏系统化的研究方法支持智能合约微服务化框架的评估;(2)受到 Hyperledger Fabric 外部链码安装部署所限,对部署相同链码的每个 peer 都要启动一个链码容器;(3)共识算法支持有限;(4)本文虽然搭建了智能合约开发运维工具链,但集成工具有限;(5)对于监控方面,未对容器性日志进行有效收集,且未深入挖掘性能指标与交易日志的数据价值;(6)本文选取官方示例链码进行案例研究,虽然该案例表示 Mictract 可以帮助工程师快速理解落地智能合约原型应用,但是当前阶段的案例并非企业中完整实际案例.

5 总结与展望

针对智能合约在开发运维过程中遇到的部署效率差、监控工具不完善等问题,本文对比分析了微服务与智能合约在设计原则上的相似性.由于微服务与智能合约在设计原则上有很大的相似性,这为本文智能合约微服务化改造奠定了良好基础.随后,本文提出了一种智能合约微服务化框架.该框架将传统的智能合约运行方式在保证运行无误的前提下将智能合约进行改造,改造后可以利用支持 DevOps 的自动化工具对智能合约进行进行持续集成、持续部署与持续监控.具体来讲,通过将智能合约与共识节点进行解耦,解耦后的智能合约以容器化的方式的插拔于下层网络节点.在此基础之上,利用 DevOps 支持微服务持续集成、持续监控的实践与工具,在流水线中将智能合约进行容器化编排以“定制化”、“插拔化”的方式进行智能合约的持续部署,利用 DevOps 监控工具对运行中的智能合约进行持续监控.本文还介绍了原型平台 Mictract 的架构与实现原理.Mictract 的核心优势是具备 BaaS 平台基本功能的前提下,屏蔽了底层方法原理与工具,形成了支撑智能合约开发运维的工具链.为验证智能合约微服务化框架及其原型平台的可行性及部署效率,本文选取了 Hyperledger Fabric 官方链码 Marbles 进行案例研究.结果表明,Mictract 能够保证智能合约改造后功能运行无误,且在智能合约部署、升级效率以及智能合约的操作监控方面取得了较为满意的效果.

此外,本文提倡在智能合约微服务化的基础上以代码库或镜像库的形式发布分享有用且经过市场检验的智能合约.开源共享[40]可以促进业界的交流合作,减少重复开发智能合约浪费的精力.为希望参与到开源智能合约的编写工作的开发者提供良好的平台与交流机会;激励开发者用类似的方式解决相似的问题,扩宽开发者的思维,为相同业务采用不同方法敞开了大门,同时可以减少因智能合约漏洞所产生的严重的经济财产损失.

未来,本文的后续工作将从以下几个方面展开深入研究.如图 4 所示,(1)在需求分析阶段,本文将基于 DDD 方法对区块链领域进行建模分析,构造针对区块链领域的 DDD 特定领域建模语言及工具;(2)本文将对现阶段支持 DevOps 的自动化工具进行全面调研,寻找适用于智能合约微服务化开发运维领域的工具,针对智能合约微服务化开发运维特殊过程进行工具自研或改造;(3)激发 Kubernetes 潜能,在智能合约层面尝试更多的 Kubernetes 可用性策略配置与 API 托管服务;(4)进行容器日志的收集工作,搭配 Kubernetes 下日志收集工具(例如 ELK、EFK)进行智能合约微服务运行过程日志的收集汇总;提升性能指标的利用率,如性能数据异常检测、报警及时反馈,并根据性能指标反馈进行区块链网络性能优化.除此之外,(5)在数据可扩展性方面,增加对区块链网络的数据可插拔性,对 Hyperledger Fabric 推荐的账本数据库 CouchDB 进行改造;(6)立足于未来,服务于开发者.进行企业级案例研究进行评估,收集并整理开发者和专家对智能合约微服务化框架的评价,并围绕企业在智能合约开发运维过程中不断涌现的新的挑战进行改造升级.

References:

[1] N. Szabo. Smart Contracts: Building Blocks for Digital Markets, 1996. [Online]. Available: http://www.fon.hum.uva.nl.

[2] Hassan UF, Ali A, Latif S, Qadir J, Kanhere S, Singh J, Crowcroft J. Blockchain And the Future of the Internet: A Comprehensive Review. ArXiv, 2019.

[3] V. Buterin. A next-generation smart contract and decentralized application platform. 2017. [Online]. Available: http://github.com/ethereum/wiki/wiki/White-Paper/.

[4] Andonia M, Robua V, Flynna D, Abramb S, Geachc D, Jenkinsd D, McCallumd P, Peacockd A. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. Renewable and Sustainable Energy Reviews,2019,100:143–174.

[5] Li SS, Xv QW, Hou PY, Chen XD, Wang YZ, Zhang H, Rong GP.Exploring the Challenges of Developing and Operating Consortium Blockchains:A Case Study. EASE'20: Evaluation and Assessment in Software Engineering, 2020. 398-404.

[6] Sun Y, Fan LJ, Hong XH. Technology Development and Application of Blockchain: Current Status and Challenges. Strategic Study of CAE, 2008,20(2):27-32 (in Chinese).

[7] Zou WQ, Lo D, Kochhar PS, Le XBD, Xia X, Feng Y, Chen ZY, Xu BW. Smart Contract Development: Challenges and Opportunities,IEEE Transactions on Software Engineering, 2019.

[8] Newman S, Wrote; Cui LQ, Zhang Y, Trans. Building Microservices: Designing Fine-grained Systems. 2nd ed., O'Reilly Media, 2015. 2–3, 9, 16, 32, 47 (in Chinese).

[9] Kalske M, Mäkitalo N, Mikkonen T. Challenges when moving from monolith to microservice architecture. In: Proc. of the Int'l Conf. on Web Engineering (ICWE 2017): Current Trends in Web Engineering. 2018. 32–47.

[10] Huang H, Zhang H, Shao D. Practical impacts of automation tools in support of DevOps in China. Ruan Jian Xue Bao/Journal of Software, 2019,30(10):3056-3070 (in Chinese).

[11] Chatterjee A, Pitroda Y, Parmar M. Dynamic Role-Based Access Control for Decentralized Applications. 2020 IEEE International Conference on Blockchain and Cryptocurrency, 2020,12404:185-197.

[12] Patrick MC, Siamak F.S, Hao F. A smart contract for boardroom voting with maximum voter privacy. International Conference on Financial Cryptography and Data Security, 2017. 357-375.

[13] Tonelli R, Pinna A, Baralla G, Ibba S. Ethereum Smart Contracts as Blockchain-oriented Microservices. XP '18:Proceedings of the 19th International Conference on Agile Software Development, 2018,21:1–2.

[14] Tonelli R, Lunesu MI, Pinna A, Taibi D, Marchesi M. Implementing a Microservices System with Blockchain Smart Contracts. 2019 IEEE International Workshop on Blockchain Oriented Software Engineering, 2019. 22-31.

[15] Zhang Y, Wen J. The IoT electric business model: Using blockchain technology for the internet of things. Peer-to-Peer Networking and Applications, 2017,10(4):983-994.

[16] Chang SE, Chen Y.When Blockchain Meets Supply Chain:A Systematic Literature Review on Current Development and Potential Applications. IEEE Access, 2020,8:62478-62494.

[17] Leka E, Selimi B, Lamani L. Systematic Literature Review of Blockchain Applications:Smart Contracts. 2019 International Conference on Information Technologies, 2019. 1-3.

[18] Christidis K, Devetsikiotis M. Blockchains and smart contracts for the internet of things. IEEE Access, 2016,4:2292-2303.

[19] Yang XM, Li X, Wu HQ, Zhao KY. The Application Model and Challenges of Blockchain Technology in Education. Modern Distance Education Research, 2017,2:34-44(in Chinese).

[20] Rodler M, Li WT, Karame G, Davi L. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. ArXiv, 2019.

[21] Li TS, Liu Y. Design and Implementation of Second-hand goods renting System Based On Ethereum Smart Contract. International Conference on Intelligent Information Processing, 2019. 346–351.

[22] Daniel B, Arti M, Brett D, Jacy R. A blockchain use case in food distribution: Do you know where your food has been? International Journal of Information Management, 2020,52:102008.

[23] Han Xuan, Yuan Yong, Wang Fei-Yue. Security Problems on Blockchain: The State of the Art and Future Trends. Acta Automatica Sinica, 2019, 45(1): 206-225.(in Chinese).

[24] Zhao Y, Yu Y, Li Y, Han G, Du X. Machine learning based privacy-preserving fair data trading in big data market. Information Sciences, 2019,478:449–460.

[25] Zhu YJ, Yao JG, Guan HB. Blockchain as a service: Next generation of cloud services. Ruan Jian Xue Bao/Journal of Software, 2020,31(1):1-19 (in Chinese).

[26] Li Z, Zhang Y, Liu Y. Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications. Tsinghua Science and Technology, 2017, 22(01):1–9.

[27] Weber I. Blockchain and Services-Exploring the Links. Service Research and Innovation, 2019,367:13-21.

[28] Hu K, Yu W, Luo K, Ding Y. A method of establishing microservice of smart contract. Patent for invention, CN108989389A, 2018(in Chinese).

[29] Wang S, Zhang X, Yu W, Hu K, Zhu J. Smart Contract Microservitization. 2020 IEEE 44th Annual Computers, Software, and Applications Conference, 2020. 569-1574.

[30] Sousa PSd, Nogueira NP, Santos RCd, Maia PHM, Souza JTd. Building a prototype based on Microservices and Blockchain technologies for notary's office: An academic experience report. 2020 IEEE International Conference on Software Architecture Companion, 2020. 122-129.

[31] Rodler M, Li WT, Karame G, Davi L. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. ArXiv, 2019.

[32] Fielding, Roy Thomas. Chapter 5: Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures [Ph.D. Thesis]. University of California, Irvine. 2000.

[33] Gorski T.Verification of Architectural Views Model 1+5 Applicability. Computer Aided Systems Theory-EUROCAST 2019, 2020. 499–506.

[34] Rodler M, Li WT, Karame G, Davi L. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. ArXiv, 2019.

[35] Luu L, Chu DH, Olickel H,Saxena P,Hobor A. Making Smart Contracts Smarter. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016. 254–269.

[36] Zhao Y, Yu Y, Li Y, Han G, Du X. Machine learning based privacy-preserving fair data trading in big data market. Information Sciences, 2019,478:449–460.

[37] Ding R, Wang Q, Dang YN, Fu Q, Zhang HD, Zhang DM. YADING: fast clustering of large-scale time series data. Proceedings of the VLDB Endowment, 2015. 473–484.

[38] Sigelman BH, Barroso LA, Burrows M,et al. Dapper, a large-scale distributed systems tracing infrastructure. Google Research,2010.

[39] Shao QF, Zhang Z, Zhu YC, Zhou AY. Survey of enterprise blockchains. Ruan Jian Xue Bao/Journal of Software, 2019,30(9):2571−2592 (in Chinese).

[40] Hippel E, Krogh G. Open Source Software and the “Private-Collective” Innovation Model: Issues for Organization Science. Organization Science, 2003,14(2), 209–223.

附中文参考文献:

[6] 孙毅,范灵俊,洪学海.区块链技术发展及应用:现状与挑战.中国工程科学,2018,20(2):27-32.

[8] Newman S,著;崔力强,张骏,译.微服务设计.北京:人民邮电出版社,2015.2-3,9,16,32,47.

[10]黄璜,张贺,邵栋.自动化工具对中国 DevOps 实践的影响.软件学报,2019,30(10):3056-3070.

[19]杨现民,李新,吴焕庆,赵可云.区块链技术在教育领域的应用模式与现实挑战.现代远程教育研究,2017,2:34-44.

[23]韩璇,袁勇,王飞跃.区块链安全问题:研究现状与展望.自动化学报, 2019, 45(1): 206-225.

[25]朱昱锦,姚建国,管海兵.区块链即服务:下一个云服务前沿.软件学报,2020,31(1):1-19.

[28]胡凯,余维,罗戡,丁毅.一种建立智能合约微服务化的方法.发明专利,CN108989389A,2018.

[39]邵奇峰,张召,朱燕超,周傲英.企业级区块链技术综述.软件学报,2019,30(9):2571−2592.