

研究生毕业论文

(申请硕士学位)

论 文 题 目 面向 Hyperledger Fabric 的
区块链云化框架的研究与实现

作 者 姓 名 _____

学 科、专 业 名 称 _____

研 究 方 向 _____

指 导 教 师 _____

2022 年 5 月 23 日

学 号：

论文答辩日期：xxxx 年 xx 月 xx 日

指 导 教 师： (签字)

Research and Implementation of A Blockchain Cloudification Framework for Hyperledger Fabric

by

Supervised by

A dissertation submitted to
the graduate school of
in partial fulfilment of the requirements for the degree of
MASTER
in

May 23, 2022

研究生毕业论文中文摘要首页用纸

毕业论文题目：面向 Hyperledger Fabric 的
区块链云化框架的研究与实现
专业 级硕士生姓名：
指导教师（姓名、职称）：

摘 要

区块链具有去中心化、不可篡改的特性，被誉为新型生产关系。区块链的发展为传统领域带来了巨大变革，去中心化应用逐步从理论论证阶段转变为工程化实践阶段。BaaS 作为一种构建、管理、托管和运维区块链网络及去中心化应用的云服务平台，在一定程度上降低了去中心化应用落地的门槛。

然而，当前区块链的发展依旧存在诸多问题。首先，区块链的基础商业应用工具不完善。BaaS 平台尚在初级阶段，研发投入巨大。市场上 BaaS 平台被头部云厂商垄断，行业马太效应明显。不同 BaaS 平台缺乏统一的顶层规划，形成数据孤岛。其次，BaaS 平台利用云能力赋能区块链乏力。BaaS 平台仅提供一种基于开源区块链平台的自动部署方案，未深入区块链与云基础设施底层。因此，BaaS 平台无法挖掘云原生技术的潜力，提供安全、可扩展、高可用的区块链服务。最后，智能合约缺少标准的开发运维流程，缺少促进去中心化应用价值交付的有效手段。

因此，针对上述挑战，本文综合考虑区块链网络节点以及智能合约的开发运维两个维度，提出了面向 Hyperledger Fabric 的区块链云化框架。首先，本文通过快速评审的方法，调研学术界如何利用 Kubernetes Operator 赋能传统领域，最终整理出 11 条赋能策略集。本文结合区块链云化的原则和区块链本身特性筛选并获得区块链网络节点云化实施方案。其次，本文对比了智能合约与微服务在设计原则上的相似性，进行了智能合约微服务化改造，并提出了智能合约微服务化开发运维流程。然后，基于区块链网络节点云化实施方案以及智能合约微服务化开发运维流程，本文设计实现了面向 Hyperledger Fabric 的区块链云化框架及其原型工具。

本文对框架及原型工具进行了基础功能自证、成熟度衡量、对比分析三维度的测试与评估。在基础功能自证方面，通过典型案例对原型工具进行了功能

可行性测试,利用 SAAM 架构评估方法验证质量属性。在成熟度衡量方面,本文结合 Kubernetes Operator 五层成熟度模型进行了定性评估。在对比分析方面,本文与官方 BaaS 平台 Cello 对比进行了定量评估。结果表明,本文提出的框架及原型平台在兼容 BaaS 基本功能前提下充分利用云的特性提升了 Hyperledger Fabric 网络的可移植性、可靠性、易用性、可扩展性以及安全性,其基本满足五层成熟度的能力要求并且拥有更优的网络部署时间和链码交付效率。

关键词: 区块链, 云原生, BaaS, Kubernetes Operator

研究生毕业论文英文摘要首页用纸

THESIS: Research and Implementation of
A Blockchain Cloudification Framework for Hyperledger Fabric
SPECIALIZATION:
POSTGRADUATE:
MENTOR:

Abstract

Blockchain has the advantages of decentralization and immutability. It is known as the next production relationship. The development of blockchain has brought great changes to the traditional field, and decentralized applications have gradually shifted from the theoretical demonstration stage to the engineering practice stage. As a cloud service platform for building, managing, hosting and operating blockchain networks and decentralized applications, BaaS greatly reduces the threshold for implementation of decentralized applications.

However, there are still many problems in the current development of blockchain-based systems. First, the commercial tools of blockchain are not perfect. The BaaS platform is still in its infancy, with huge R&D investment. In the market, the BaaS platform is monopolized by the top cloud manufacturers, and the Matthew effect of the industry is obvious. Different BaaS platforms lack unified top-level planning, resulting in isolated islands of data. Secondly, the BaaS platform is weak in empowering the blockchain with cloud capabilities. The BaaS platform only provides an automatic deployment solution based on the open source blockchain platform, without going deep into the principles of the blockchain and cloud infrastructure. Therefore, the BaaS platform cannot tap the potential of cloud-native technology to provide secure, scalable and highly available blockchain services. Finally, smart contracts lack standard development and operation processes, and lack effective means to promote the value delivery of decentralized applications.

Therefore, in response to the problems, this thesis proposes a cloudification framework for blockchain based on Hyperledger Fabric by comprehensively considering

the blockchain network nodes and smart contracts. Firstly, this thesis surveys how academia uses Kubernetes Operator to empower traditional fields through a rapid review method and sorts out eleven enabling strategy sets. The blockchain nodes cloudification plan is obtained by filtering the principles of blockchain cloudification and the characteristics of blockchain itself. Secondly, this thesis compares the similarities between smart contract and microservice in design principles, carries out the refactor of smart contract into microservice, and proposes the process of developing and operating microservices for smart contracts. Then, based on the cloud implementation plan of blockchain network nodes and the process of smart contract microservices, this thesis designs and implements the blockchain cloudification framework and its prototype platform for Hyperledger Fabric.

In this thesis, the framework and prototype tool are tested and evaluated in three dimensions: self-validation of basic functions, maturity measurement, and comparative analysis. In terms of self-certification of basic functions, the functional feasibility test of the prototype tool is carried out through a typical case analysis, and the quality attribute is verified by the SAAM architecture evaluation method. In terms of maturity measurement, this thesis conducts a qualitative evaluation based on the five maturity levels of Kubernetes Operator. In terms of comparative analysis, this thesis conducts a quantitative evaluation compared with the official BaaS platform Cello. The test and evaluation results show that the framework and prototype tool can make full use of the characteristics of the cloud to improve the portability, reliability, usability, scalability, and security of Hyperledger Fabric network. The prototype tool basically meets the requirements of five maturity levels and has better results than Cello in terms of deployment time and delivery efficiency.

keywords: Blockchain, Cloud Native, BaaS, Kubernetes Operator

目 录

目 录	v
插图清单	vii
附表清单	ix
第一章 绪论	1
1.1 研究背景及意义	1
1.2 研究现状	4
1.3 本文贡献与主要研究工作	6
1.4 本文组织结构	8
第二章 理论与技术支持	9
2.1 区块链技术	9
2.1.1 区块链基本概念	9
2.1.2 Hyperledger Fabric	11
2.1.3 Blockchain as a Service	13
2.2 云原生	14
2.2.1 云原生基本概念	14
2.2.2 Kubernetes	18
2.3 Helm	20
2.4 本章小结	21
第三章 区块链网络节点云化方案	23
3.1 快速评审	23
3.2 设计原则	25
3.3 策略集应用	26
3.4 本章小结	33
第四章 智能合约微服务化开发运维方法	35
4.1 研究目的	35
4.2 设计原则	36
4.3 智能合约微服务化改造	37

4.4 智能合约微服务化开发运维流程 ······	40
4.5 原型工具与方法评估 ······	42
4.6 本章小结 ······	45
第五章 面向 Hyperledger Fabric 的区块链云化框架 ······	47
5.1 面向 Hyperledger Fabric 的区块链云化框架 ······	47
5.1.1 Custom Resource Definition ······	48
5.1.2 Manager ······	50
5.1.3 Hyperledger Fabric 网络 ······	54
5.2 原型工具 ······	55
5.2.1 工具概述 ······	55
5.2.2 需求分析 ······	56
5.2.3 设计与实现 ······	60
5.3 本章小节 ······	64
第六章 测试与评估 ······	65
6.1 测试环境 ······	65
6.2 基本能力自证 ······	66
6.2.1 案例研究 ······	66
6.2.2 架构评估 ······	70
6.3 成熟度衡量 ······	73
6.4 对比分析 ······	75
6.5 本章小结 ······	78
第七章 总结与展望 ······	79
7.1 总结 ······	79
7.2 局限 ······	80
7.3 展望 ······	81
参考文献 ······	83

插图清单

1-1 研究内容	3
2-1 区块链示例图	9
2-2 Hyperledger Fabric 网络架构	11
2-3 Hyperledger Fabric 账本结构	12
2-4 BaaS 与 PaaS 以及 SaaS 的对比	13
2-5 BaaS 通用架构	14
2-6 云原生技术发展趋势	15
2-7 云原生技术范畴	16
2-8 国内外云原生技术现状	17
2-9 Kubernetes 架构图	18
2-10 Helm 架构图	21
3-1 策略集应用	32
4-1 镜像方式发布微服务流程	38
4-2 智能合约微服务化改造流程	38
4-3 智能合约微服务化开发运维流程	40
4-4 Mictract 架构图	43
5-1 面向 Hyperledger Fabric 的区块链云化框架	48
5-2 Manager 监听 CR	50
5-3 Controller 循环监听	51
5-4 原型工具存储策略	52
5-5 原型工具安全策略	53
5-6 智能合约微服务部署原理	54
5-7 原型工具总体功能	55
5-8 HF 网络管理用例图	56
5-9 通道管理用例图	58

5-10 创建 Ca Resource 时序图	60
5-11 Ca Controller Reconcile 逻辑时序图	61
5-12 Ca Enroll User 逻辑时序图	62
5-13 安装链码逻辑时序图	63
6-1 评估体系	65
6-2 测试网络	66
6-3 CRD 部署结果	67
6-4 创建 Org1 测试结果	67
6-5 创建 Orderer 测试结果	68
6-6 查看通道高度测试结果	69
6-7 查询链码	69
6-8 网络节点状态	70
6-9 网络存储状态	71
6-10 Ca CPU 监控图	72
6-11 成熟度模型	73
6-12 网络部署时间对比图	76

附表清单

1-1 主要公有云的 BaaS 平台	2
2-1 区块链类型	10
3-1 每个全文数据库的搜索字符串	23
3-2 快速评审入选文献列表	24
3-3 Kubernetes Operator 云化策略集	25
3-4 访问控制机制	30
4-1 智能合约与微服务设计原则	37
4-2 通信机制对比	39
4-3 操作链码实例	45
4-4 部署效率对比 (单位: 秒 (s))	45
5-1 CRD 描述	49
5-2 Ca 管理用例表	57
5-3 Peer 管理用例表	57
5-4 Orderer 管理用例表	58
5-5 通道管理用例表	59
5-6 链码管理用例表	59
6-1 配置详情	66
6-2 创建 Org1 测试用例	67
6-3 创建 Orderer 测试用例	67
6-4 注册、登记用户测试用例	68
6-5 创建通道测试用例	68
6-6 创建通道测试用例	69
6-7 场景分类结果	70
6-8 Hyperledger Fabric 版本信息	75

6-9 网络部署时间 (单位: 秒 (s))	76
6-10 链码交付时间 (单位: 秒 (s))	77

第一章 绪论

1.1 研究背景及意义

区块链 (Blockchain) 是一种按照时间顺序将数据区块以链条的方式组成的特定数据结构, 被视为分布式的共享账本和数据库。它能够使用户在无可信中介的情况下完成价值传输 [1]。作为区块链 2.0 的以太坊^①重新将智能合约描述为部署于区块链网络中的合同条款代码。这意味着传统的合同条款可以进入实体计算机中, 且在区块链网络去中心化、不可伪造、不可篡改的特性下严格执行。区块链推进了人与企业之间和线上与线下的全方面互联, 其被称为新型生产关系。随着区块链的快速发展, 智能合约极大地丰富和扩展了区块链应用场景, 已经快速渗入到人们生活的方方面面。目前, 去中心化应用纷纷落地, 逐步从概念验证阶段转化为工程化实践阶段^②。

与此同时, 云原生 (Cloud Native) 是基于云的软件架构思想和软件开发实践的一组方法论, 因其弹性和分布式的优势成为当今流行的软件服务模式。云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中构建和运行可弹性扩展的应用。用户可以借助平台的全面自动化能力, 持续交付部署业务生产系统。

区块链即服务 (Blockchain as a Service, 简称 BaaS) 则是基于云原生技术体系的一种构建、管理、托管和运维区块链网络及其应用的云服务平台 [2]。BaaS 支持将任何企业级区块链实施到云环境, 而无需任何 IT 专业知识。这大大降低了区块链技术的使用门槛, 是促使区块链技术更广泛、更深入地渗透到各个行业和企业的催化剂, 其市值预计从 2018 年的 6.23 亿美元猛增 2023 年的 150 亿美元^③。市场发展迅速, 具有极大的商业价值。其中, 云厂商提供了大多数的 BaaS 平台 [3], 如表 1-1 所示, 其底层区块链支撑技术大多数选择 IBM 开源的跨企业级联盟链 Hyperledger Fabric(简称 HF), 这也是本文选择 HF 的原因。

^①以太坊白皮书

^②自动化框架 (BAF) 简介

^③Global Blockchain-as-a-Service Market 2018-2022

表 1-1: 主要公有云的 BaaS 平台

BaaS 平台	Ethereum	Quorum	Corda	HF
AWS	Y	Y	Y	Y
Azure	Y	Y	Y	Y
IBM				Y
阿里云区块链服务	Y			Y
腾讯云区块链服务 TBaaS				Y
华为云区块链服务 BCS				Y

BaaS 基于云的自动化的功能屏蔽了底层区块链技术, 为上层去中心化应用提供了便捷地构建区块链网络等功能。然而, 软件工程中没有银弹, 当前区块链的发展依旧存在诸多挑战。

首先, 区块链基础商业化应用工具并不完善^①。BaaS 平台发展还处于初级阶段, 商业运行模式仍处在探索阶段。由于 BaaS 平台研发投入大, 目前市场上存在如 AWS、IBM、阿里云区块链服务等多种 BaaS 平台解决方案, 这些 BaaS 平台计费高昂^②并且与其他云服务捆绑销售。传统的企业业务或已存在稳定合作的云服务商, 使用不同云服务商的合作伙伴就需要跨云部署, 由于缺乏统一的顶层规划, 各云厂商的 BaaS 底层异构, 存在不可复用的跨解决方案的资源和工具, 这形成了多云的网络 [4]、技术信息孤岛。同时, BaaS 平台几乎都由互联网云服务巨头企业把控, 可用性限制通常会迫使企业为来自各种云提供商的基础设施即服务 (Infrastructure as a Service, 简称 IaaS) 付费, 最终形成马太效应 [3]。

其次, BaaS 平台利用云能力对区块链基础设施赋能乏力。作为一种广泛部署的技术, 云计算是实施区块链技术的合适目标。然而, 区块链与云原生两种技术都相对不成熟, 两者的集成可能会在技术方面产生新的复杂性 [2]。区块链服务比常规云服务更复杂, 问题仍然在于如何使区块链和云兼容 [5]。目前, 行业缺少既定的标准或最佳实践范例, 各云厂商的 BaaS 解决方案基本都采取已有的开源解决方案, 业务同质化严重。具体地, 他们仅提供一种基于开源区块链平台的自动部署管理方案, 未深入到区块链与云基础设施的底层。然而, 自动化脚本部署并不是有效的云化方式, 浪费了云原生技术的潜力。例如, 在数据存储备份及可扩展性方面, 随着交易数量的增加, 传统自动化部署方案无法利用云的弹性伸缩能力, 最终出现存储膨胀问题。这限制了区块链的数据可扩展性, 所以需要结合云的能力寻找数据备份、升级及可扩展性的方法。区块链基础设施如何与云

^① 区块链基础设施研究报告 (2021 年)

^② 阿里云区块链服务 BaaS 规格与定价

原生快速深度结合,利用伸缩性、可移植性和高可用性的特性来提供“高质量”的区块链服务,是当前仍然遗留的挑战。

最后,区块链领域缺少促进去中心化应用价值交付的有效手段。去中心化应用价值交付很大程度上就意味着利用智能合约编写合同条款代码,然而智能合约技术还处在初级阶段,其相较于传统应用而言缺少既定的标准开发运维流程。具体地,目前智能合约部署过程缺乏有效的自动化手段,耗费大量的时间成本;智能合约作为区块链的链上代码执行,安全问题产生会造成巨大的损失,合约代码在部署过程中缺乏有效的安全性验证,易导致代码漏洞;智能合约运行过程,缺乏有效的监控手段,对智能合约运行过程中的负载等情况一无所知。对于大部分的去中心化应用开发组织来说,如何将传统成熟工具和新型智能合约应用联系起来,仍是一个棘手的问题。

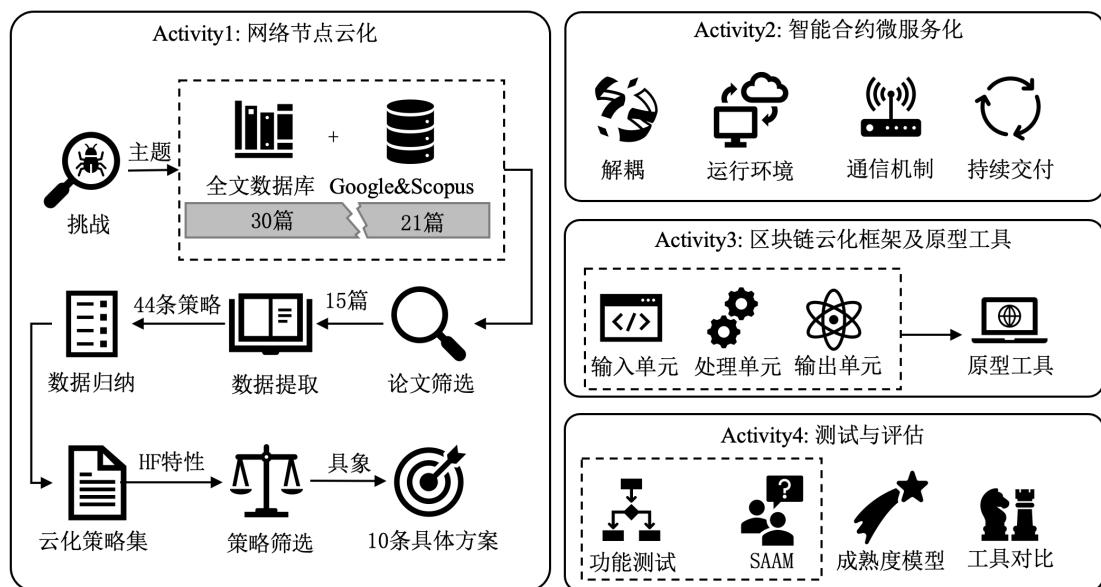


图 1-1: 研究内容

针对上述问题,本文选择面向联盟链场景的开源框架 **HF** 和 **Kubernetes** 提出了一种区块链云化框架,包含了对下层区块链网络基础设施节点的云化以及智能合约的开发运维流程两个方面,具体工作如图 1-1 所示。在区块链网络节点云化方面,本文对 **Kubernetes Operator** 进行了快速评审,得到 **Kubernetes Operator** 云化赋能传统领域的策略集。然后,本文结合 **HF** 网络的特性对策略集进行了筛选获得具体的节点云化方案。在智能合约的开发运维流程即价值交付方面,本文首先对比了智能合约与微服务在设计原则上的相似性,在此基础之上将智能合约进行微服务化改造,包括智能合约与网络节点的解耦、运行环境、通信机制

以及持续交付。智能合约经改造后能够有效利用传统成熟的工具支撑其开发运维工作。基于网络节点云化以及智能合约微服务化本文设计了面向 Hyperledger Fabric 的区块链云化框架。该框架对 HF 网络中的 Ca、Orderer、Peer 节点以及链码进行抽象适配并利用 Kubernetes Operator 进行完整生命周期管理, 利用智能合约微服务化流程将智能合约持续交付到云化节点。框架提升了 HF 网络的可移植性、可靠性、易用性、可扩展性以及安全性, 能够使 HF 网络获得类似云的自我管理, 使微服务的开发运维流程服务于智能合约领域。最终, 本文对框架及其原型工具从基本能力自证、成熟度衡量以及对比分析三个角度进行了测试与评估。在基本能力自证方面, 利用典型案例分析对原型工具进行了功能方面的可行性测试, 利用基于场景的 SAAM 架构方法对区块链基础设施的质量属性进行评估; 在成熟度衡量方面, 利用五层成熟度模型进行定性评估; 在对比分析方面, 与官方 Cello 对比进行定量评估。

1.2 研究现状

由于区块链智能合约的无法删除、修改、历史可追溯、去中心化、严格执行等特点, 越来越多的研究人员想挖掘区块链的潜能, 将区块链的能力应用到各种应用场景。McCorry 等人 [6] 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, 公开投票网络是一种自助协议, 每个选民都控制着自己投票的隐私, 只有在所有人都参与的情况下才会被破坏, 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。Chang 和 Chen[7] 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, 存在信任和性能问题, 利用区块链的潜力可以更好地扰乱供应链运作性能、分布式治理和过程自动化。Zhang 和 Wen[8] 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。Leka 等人 [9] 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 [10]、医疗、教育、政府各个领域都有涉及。

目前, 去中心化应用逐步从概念验证阶段转变为工程化、商业化阶段。由于区块链的复杂性给网络的构建以及智能合约的部署、运维工作带来了严重的时间成本。在去中心化应用价值交付过程中, 存在对于区块链底层技术的易用性、部署效率、安全性等多方面的挑战。研究人员在区块链基础设施与云原生结合

方面都开始了一定的探索,除了利用区块链的特性提升云的能力外 [11][12][13],研究人员也都期望利用云的特性自动化地构建出易于弹性扩展、高可用的区块链平台。

在学术研究方面,研究人员探究如何有效利用云平台来部署区块链平台。Gerrits 等人 [4] 在 **Kubernetes** 中部署了分布式账本 **Hyperledger Sawtooth**^①,并运行了一个用例。他们旨在探讨该用例在真实场景云部署中的可行性和可扩展性。Liang 等人 [14] 针对教育领域数据共享和信息欺骗的问题构建了一个高可用的教育联盟区块链平台,并实现了基于 **Kubernetes** 的 HF 部署,实现了将链码纳入 **Kubernetes** 环境管理的目标。然而, Wan 等人 [15] 指出当前主流的 **BaaS** 提供商通常采用 API 进行用户访问,或者简单地将区块链应用迁移到云,这会侵蚀不可信的机制并带来锁定风险。他们随后提出了一种新的服务范式来克服现有 **BaaS** 的局限性。基于 HF 的实施表明,该范式可以缓解当前 **BaaS** 对区块链特征的侵蚀。在云原生底层基础设施方面研究人员关注与区块链结合的 **Kubernetes** 调度问题。才丽 [16] 在 **Kubernetes** 上面对 PBFT 和区块链的本身特性提出了静态调度和自适应算法。Shi 等人 [17] 为了解决云端实现 PoS 区块链工作负载的高效调度问题,首次在云计算中设计和实现了基于 **Kubernetes** 的解决 PoS 区块链应用程序迁移成本的系统,最大限度地减少了使用的 **Kubernetes** 工作节点数量以降低总体费用,而且还提出了一种高性能的 **Kubernetes** 调度方案 HPKS 以最大限度地利用工作节点进行在线 Pod 管理。在智能合约开发运维层面, Porru 等人 [18] 指出需要为软件工程师提供特定于区块链软件开发的工具与技术,并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), 将开发 BOS 的软件实践称为面向区块链的软件工程 (Blockchain-Oriented Software Engineering, 简称 BOSE)。Tonelli 等人 [19] 在 2018 年就指出了微服务架构与智能合约结构之间的相似性, Hu 等人^② 提出一种建立智能合约的微服务化方法,以期望将智能合约在 **Kubernetes** 上运行。

相比于学术研究,工业领域的探索更加注重自动化实践。**Hyperledger Cello**^③ 支持在多种底层基础设施上从头快速构建 **BaaS** 平台, 提供管理区块链网络的生命周期、自定义区块链网络配置等功能帮助人们以更高效的方式使用和管理区块链。**Hyperledger Cello** 当前阶段重点关注在 Docker 安装,对于 **Kubernetes** 的支持方面仍处在相对初级阶段, 配置项简单灵活性不足且老旧。

^①Hyperledger Sawtooth github 地址

^②发明专利: 一种建立智能合约微服务化的方法

^③Hyperledger Cello

Blockchain Automation Framework^①提供了一个自动化框架,利用 Ansible^②以及 Helm^③快速地、一致地将生产就绪的分布式账本技术 (Distributed ledger technology, 简称 DLT) 平台部署到云基础设施。虽然, Blockchain Automation Framework 提供了一个自动化框架将区块链平台部署于 Kubernetes, 但本质上还是描述为一个需要希望远程主机执行命令的方案, 或者一组 IT 程序运行的命令集合。这大大提升了自动化程度, 但其远没有发挥 Kubernetes 的潜力。

尽管学术界以及工业界目前已有一些关于区块链云化的探索与研究, 但研究重点多为如何自动化地将区块链平台向云上迁移, 并且区块链平台的性能往往由于许多影响因素而不稳定, 尤其是当它们部署在动态云环境中的时候。综上, 当前研究缺少构建支持云和区块链一体化的有效服务模型 [17], 缺少在云中操作区块链的实证研究 [20]。

在云中操作应用程序就需要将应用程序部署进 Kubernetes, 这个过程需要部署多种资源。虽然这些资源有标准定义的模板, 但目前手工创造定义这些资源会导致大量重复的代码并引入人为错误。Kubernetes Operator 是将领域知识集成到 Kubernetes API 编排过程中的最新方法 [21]。一部分研究利用 Kubernetes Operator 方法通过自动化的方式将云底层的效率、灵活性等多方面优势拓展到多种领域。Huang 等人 [22] 提出了一个轻量级的遥感大数据处理云原生框架。该框架利用 Kubernetes Operator 融合 Spark, 自动化配置 Spark 参数, 提升并行遥感图像融合算法的效率。Zhou 等人 [23] 提出了 Torque Operator 对高性能负载 (High Performance Computing, 简称 HPC) 进行管理, 利用容器化来提升 HPC 的效率及灵活性。除此之外, 在 5G[24][25]、医疗 [26] 等领域, Kubernetes Operator 也发挥出其强大的自动化与编排能力, 利用云原生的可迁移性、可伸缩性、安全性等特性进行赋能, 但在区块链领域的 Kubernetes Operator 相关工作较少。

1.3 本文贡献与主要研究工作

本文的研究目标是在云原生技术体系下, 更简单、更原生地管理 HF。其核心价值体现在: (1) 快速部署。当前去中心化应用的价值交付过程有 10% 到 20% 的时间都花费在网络的启动和管理上, 尤其是还会出现更复杂的配置情况。(2) 区块链云化框架并不是建立新的壁垒, 而是避免云厂商的锁定。(3) 提供云与区

^①Blockchain Automation Framework

^②Ansible

^③Helm

块链结合的实践范例,包含区块链网络节点的云化方案以及智能合约微服务化的开发运维流程。(4)使网络根据业务可变。如联盟中新的成员的加入、随数据膨胀而增加账本资源等,这些改变通过规范化代码的角度而不是人工干预。

具体地,本文的主要研究工作分为以下四个方面:

1. 由于当前缺少云和区块链一体化有效服务模型,所以针对区块链的基础设施即区块链网络节点。本文首先以 **Kubernetes Operator** 及其衍生词汇为主题进行了快速评审,快速评审的范围涵盖了计算机与软件工程领域的权威全文数据库。经过快速评审,本文得到了 **Kubernetes Operator** 赋能传统领域提升软件质量属性的策略集。根据区块链云化设计原则并结合区块链网络的特性,最终形成了区块链网络节点云化的具体实施方案。

2. 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,本文将智能合约在保证运行无误的前提下进行了微服务化改造。最后本文提出了智能合约微服务化开发运维流程,其可以利用成熟的自动化工具对智能合约进行持续集成、持续部署与持续监控。

3. 基于上述区块链网络节点云化和智能合约微服务化开发运维方法,本文设计实现了面向 **Hyperledger Fabric** 的区块链云化框架及其原型平台。框架自定义三种网络节点资源类型 (**Custom Resource Definition**, 简称 **CRD**) 以及一种链码资源类型。这些 **CRD** 将区块链领域知识注入 **Kubernetes** 基础设施,通过这种方式可以自动化配置网络,消除人为错误,提升云化框架的易用性。其次, **Manager** 作为框架中枢处理单元对利用 **Helm** 对 **HF** 网络节点进行全生命周期管理,并根据节点云化实施方案,复用 **Kubernetes** 机制提升框架及 **HF** 网络节点的可移植性、可靠性、易用性、可扩展性以及安全性。最后, **HF** 网络作为框架的输出单元,包含了维持 **HF** 网络节点运行的各种配置以及合理的运行状态。成功启动稳定的区块链网络后,框架能够利用传统成熟的工具持续地将智能合约部署到云化的区块链网络上,持续地向用户交付智能合约的价值。

4. 对原型平台进行了测试与评估。本文从以下三个维度对框架及其原型工具进行评估:基础能力自证,本文以典型案例的方式对原型工具进行了功能性测试,利用 **SAAM** 架构评估方法验证网络节点的质量属性;成熟度衡量,本文结合五层成熟度模型对原型工具进行了定性评估;对比分析,本文与 **Hyperledger** 官方的 **BaaS** 工具 **Cello** 对比进行了定量评估。结果表明,本文的原型工具在兼容区

块链基本功能的前提下能够充分利用云的特性提升 HF 网络节点的可移植性、可靠性、易用性、可扩展性以及安全性,利用智能合约微服务化的开发运维方法促进智能合约的价值交付过程。本文的原型工具可以基本满足五层成熟度模型的能力并且在网络部署时间和链码交付效率方面,原型工具具有更优的表现。

1.4 本文组织结构

本文组织结构如下:

第一章 绪论。介绍了本文的研究背景及意义、国内外研究现状、工业界的主要探索以及本文主要的研究工作。

第二章 理论与技术支持。介绍了区块链尤其是 HF 的相关理论和概念; 同时介绍了云原生的基本概念发展历程, 并对云原生基础设施 Kubernetes 进行了详细介绍。

第三章 区块链网络节点云化方案。介绍了通过快速评审获得 Kubernetes Operator 赋能传统领域的策略集, 同时结合区块链云化设计原则以及区块链本身特性对策略集进行筛选得到具体云化实施方案。

第四章 智能合约微服务化开发运维方法。介绍了智能合约与微服务的设计原则, 同时基于这两者在设计原则上的相似性提出智能合约微服务化改造以及智能合约微服务化开发运维流程。

第五章 面向 Hyperledger Fabric 的区块链云化框架。介绍了本文提出的面向 Hyperledger Fabric 的区块链云化框架, 包含输入、处理以及输出单元; 介绍了原型工具的需求分析、设计与实现。

第六章 测试与评估。从基础能力自证、成熟度衡量以及对比分析三个维度对框架及原型工具进行测试与评估。

第七章 总结与展望。对全文进行总结; 阐述了区块链云化框架的局限性; 对未来工作进行了展望。

第二章 理论与技术支持

本章将介绍区块链及云原生相关概念和理论知识, 还将介绍本文区块链云化框架所涉及到的其他技术与工具。

2.1 区块链技术

2.1.1 区块链基本概念

区块链是以比特币等数字加密货币体系为核心支撑技术的一种全新的去中心化基础架构与分布式计算范式 [27]。区块链通常被当作分布式账本, 具有去中心化、持久性、匿名性、不可篡改性、可追溯性的特点。

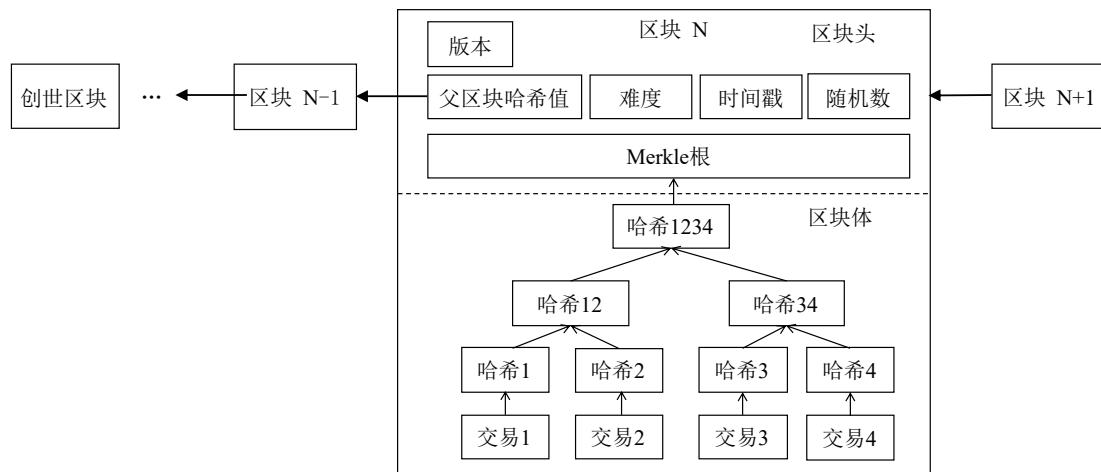


图 2-1: 区块链示意图

区块链典型示例如图 2-1所示, 可以看作是一种按照时间顺序将数据区块以顺序相连的方式组合成的链式数据结构, 并以密码学方式保证其不可篡改和不可伪造。每个数据区块包含区块头 (Block Header) 和区块体 (Block Body) 两部分, 区块头主要用来存储本区块的一些相关属性, 区块体则用来存储真实的交易数据记录。区块头主要由三组数据组成, 第一组是父区块的哈希值, 用来将该区块与它的前一区块相连接; 第二组数据和矿工竞争挖矿有关, 即难度、时间戳和随

机数 (Nonce); 第三组是由区块体中计算出来的根哈希值, 即默克尔 (Merkle) 根。区块体包括当前区块经过验证的、区块创建过程中生成的所有交易记录。这些记录通过默克尔树的哈希过程生成唯一的默克尔根并记入区块头。整个区块链的第一个区块称为创世区块 (Genesis Block)。如果网络中大多数节点就新区块中交易的有效性和区块本身的有效性达成共识, 则可以将新区块添加到链中。

表 2-1: 区块链类型

	公有区块链	私有区块链	联盟区块链
准入限制	无	有	有
读取者	任何人	仅限受邀用户	相关联用户
写入者	任何人	获批参与者	获批参与者
所属者	无	单一实体	多方实体
交易速度	慢	快	快

当前, 区块链分为公有区块链、联盟区块链和私有区块链。如表 2-1 所示, 公有链没有准入限制, 没有监管方参与, 任何人都可以参与共识, 常见的两种共识协议为工作量证明机制 (Proof of Work, 简称 PoW) 和权益证明机制 (Proof of Stake, 简称 PoS)。公有链允许任何人都可以自由加入, 具有高度分布式的拓扑结构。但是, 公有链在安全性和性能方面也进行了权衡。公有链上的许多服务器遇到了扩展瓶颈, 吞吐量相对较弱; 与公有区块链的无准入限制形成鲜明对比的是, 私有区块链建立了准入规则, 规定谁可以查看和写入区块链。因为在控制方面有明确的层次结构, 私有链也不是去中心化系统。在某些私有链中, 具备安全模型的背景下, 共识协议是多余的。因此在私有区块链中, 不使用 PoW 并不会造成很严重的威胁, 因为每个参与者的身份都是已知的, 是手动进行管理的; 联盟区块链是介于公有链和私有链之间的, 结合了两者的特征要素。在共识方面, 联盟链将少数同等权力的参与方视为验证者, 而不是像公有链那样开放的系统, 让任何人都可以验证区块, 也不是像私有链那样, 通过一个封闭的系统, 只允许某一个实体来任命区块的生产者。对于从事各类活动的个人和企业来说, 存在大量的区块链选择。即使在公有链、私有链和联盟链中, 根据复杂性的不同, 也会出现许多不同的用户体验。根据实际使用情况, 企业可以选择最适合的链实现目标产品。供应链、电商、医疗等需要彼此之间相互沟通的场景下, 联盟链可减轻私有链中交易对手的风险, 并且较少的节点数通常可使它们能够比公有链更有效率。

2.1.2 Hyperledger Fabric

Hyperledger Fabric^①是一种企业级联盟链解决方案，因其可插拔模块化、可伸缩、可扩展的架构、多编程语言的智能合约受到业界广泛追捧。

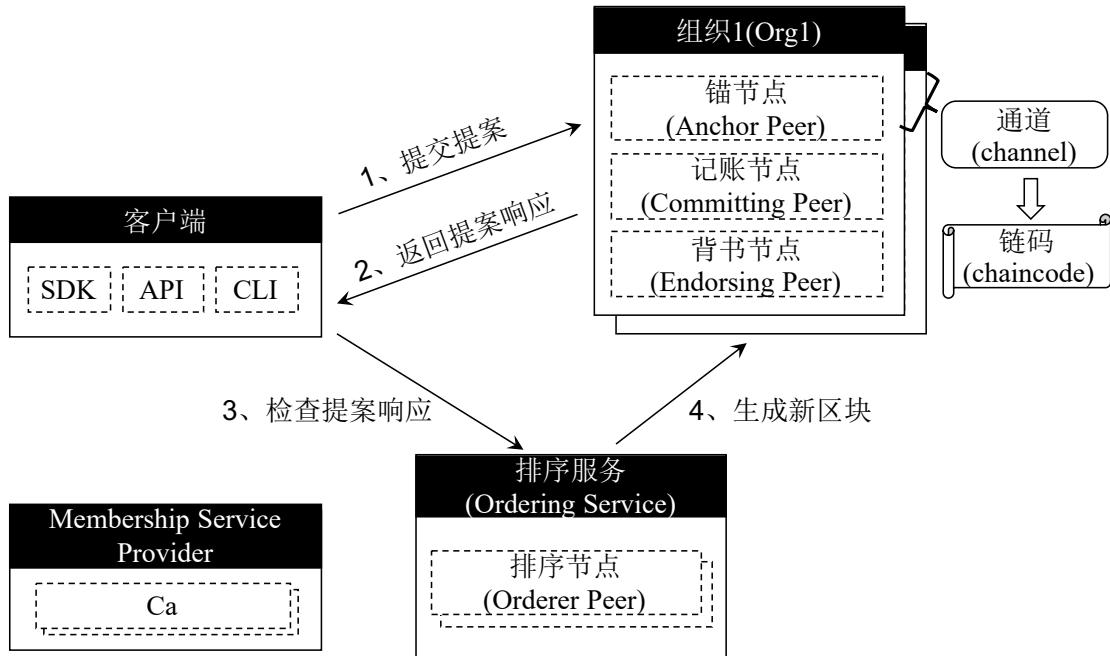


图 2-2: Hyperledger Fabric 网络架构

如图 2-2 所示，HF 网络通过组织划分，每个组织内包含多种不同角色的 Peer 节点，每个 Peer 节点又可以担任多种角色，所有的组织共用排序服务。HF 多种节点通过网络相互链接组成联盟链网络完成链上交易。

网络节点

(1) 客户端节点: 在 HF 网络外部用于主动与区块链交互、实现区块链操作的组件。常见的客户端节点有软件开发工具包 (Software Development Kit, 简称 SDK)^②、Fabric-CLI^③、REST API^④；

(2) Ca 节点: Fabric-Ca^⑤是一个官方可选的 Membership Service Provider(MSP) 组件，对 HF 网络中各实体 (Identity) 的数字身份数字证书进行管理。完成实体身份注册、数字证书的签发续签或吊销；

^①Hyperledger Fabric

^②Hyperledger Fabric Go 版本的 SDK

^③fabric cli

^④fabric api

^⑤fabric ca

(3) Peer 节点: HF 网络的每个组织都包含一个或多个 Peer 节点, 每个 Peer 节点可以通过配置文件担任一种或多种角色。

- 锚节点 (Anchor Peer): 负责与其他组织的锚节点进行通信;
- 记账/提交节点 (Committing Peer): 负责对区块及区块交易进行验证, 验证通过后将区块写入账本中, 同时提交节点会定期与其他节点通过 Gossip 协议进行信息交换;
- 背书节点 (Endorsing Peer): 负责对客户端发送的提案进行签名背书。背书节点与具体的链码 (Chaincode) 绑定, 其通过调用链码模拟执行交易并向生成提案的客户端返回提案响应。背书节点是动态的, 在客户端发起提案时才会根据背书策略 (Endorsement Policy) 成为背书节点, 其他时候为记账节点。

(4) Orderer 节点: 排序服务节点接收经过背书签名的交易并对未打包的交易进行排序生成新区块, 最终通过原子广播到记账节点。排序服务采取可插拔设计, 支持 Solo、Kafka 等分布式共识协议。

HF 区块链支持在通信节点之间启用传输层安全性协议 (Transport Layer Security, 简称 TLS) 保证两通信节点的数据保密性和完整性。TLS 采用 x509 证书进行身份验证并生成会话密钥, 不仅支持客户端节点对服务节点的身份验证, 同时也支持服务节点来验证客户端的双向验证。

账本结构

HF 中智能合约被称为链码, 通过通道 (Channel) 允许参与者同时履行不同的链码。HF 网络子链通常按照“1 个通道 +1 个账本 +N 个成员”组成。不同组织及其成员能够在通道中完成特定的交易, 限制信息传播范围。建立一个通道就相当于创建了一条子链, 这条子链上只拥有唯一的一份账本。

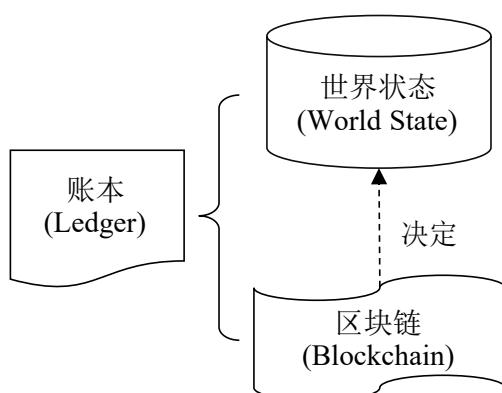


图 2-3: Hyperledger Fabric 账本结构

如图 2-3 所示, HF 账本由区块链以及世界状态 (World State) 组成, 其中世界状态由区块链决定。首先, 世界状态是一个可插拔的键值对 (Key-Value, 简称 K-V) 数据库, 提供简单、快速、丰富的账本状态检索和存储方式。通过世界状态, 客户端能够直接定位访问账本状态的某个值, 不需要遍历计算整个交易日志。为解决不同类型的问题, HF 提供了 LevelDB 和 CouchDB 来保障账本状态类型的灵活性。当账本状态是简单的键值对时, 使用 LevelDB 合适; 当账本状态结构为 JSON 时, 使用 CouchDB 合适。其次, 这里的区块链指的是交易日志, 不是指区块形成的链。区块记录了世界状态改变的历史, 并以文件的方式进行持久化。交易数据一旦写入区块链就无法篡改。

2.1.3 Blockchain as a Service

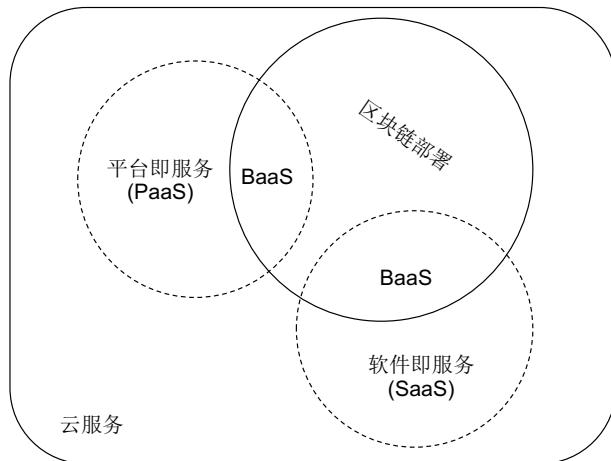


图 2-4: BaaS 与 PaaS 以及 SaaS 的对比

随着区块链以及云原生的发展, BaaS 也悄然兴起。云提供了一种按需的、虚拟的、资源可伸缩的 IT 环境, BaaS 则提供了一种基于云的区块链服务。BaaS 能够在云上构建、管理、托管和运维区块链技术, 能够快速部署区块链网络及其开发环境、编写智能合约、构建去中心化应用 (即区块链应用, Decentralized Application, 简称 Dapp)。基于云基础设施, BaaS 屏蔽了底层区块链与云原生的逻辑, 消除了用户构建开发去中心化应用的壁垒, 为用户提供便捷的、一体化的区块链的能力。如图 2-4 所示, 根据 BaaS 的实施方式, BaaS 在云环境中的位置会有所不同 [2]。BaaS 可以从平台即服务 (Platform as a Service, 简称 PaaS) 获得基础设施支持的同时也可以从软件即服务 (Software as a Service, SaaS) 获得软件服务。

微软推出由 Azure 云驱动的开放式区块链平台 Bletchley, 该项目保证服务对于所有平台、合作者和客户来讲都是开放的、灵活的 [28]。IBM 推出了名为 Bluemix 的云计算平台, 依托于 PaaS 云帮助开发者更快地进行应用开发和部署。随后 AWS、Google、阿里云等也相继推出自家的区块链即服务平台。以 HF 为例, BaaS 平台通用的架构如 2-5 所示, 本质上 BaaS 以计算、存储等资源为基础, 联合上层的区块链基础设施的相关能力, 如共识能力、记账能力、智能合约等转化为可编程接口, 使得区块链网络的部署以及去中心化应用开发过程简单而高效。同时, BaaS 通过底层标准化的云基础设施能力为上层的区块链及其去中心化应用提供安全可靠的支撑, 解决弹性、网络、安全性、性能等难题。本文重点关注于基础设施层中 Kubernetes 上的区块链云化框架的研究。

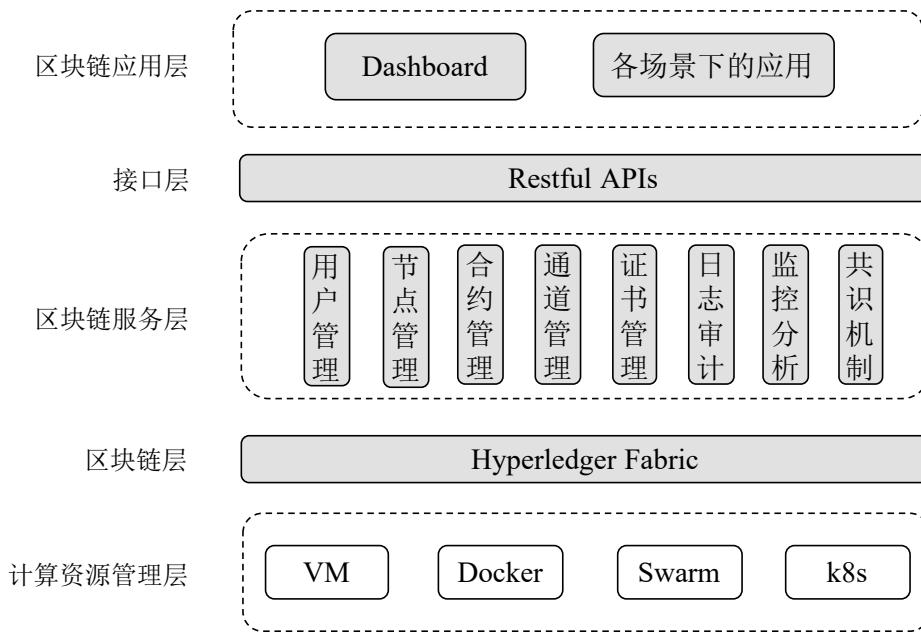


图 2-5: BaaS 通用架构

2.2 云原生

2.2.1 云原生基本概念

云原生, 即云原生计算。从发展历程来说, 云原生是云计算的升级。云计算最早由 Dell 公司在 1996 年提出 [29], 亚马逊公司在 2006 年率先推出的弹性计算云 (Elastic Compute Cloud, 简称 EC2) 服务对云计算产生了深刻影响, 越来越多的企业开始逐步接受云计算这一概念, 并将应用逐步迁移到云端, 享受这一新型

计算方式带来的技术红利。此后,软件系统规模、软件开发方式驱动着技术不断升级。在云计算的时代,云端只是用于计算的场所,应用无须重新编写,只需重新部署,应用的迁移从物理机到虚拟机,存储选用兼容的块存储或文件存储。但几乎所有分布式场景中都需应用自行解决稳定性、数据同步、容灾等方面的问题。要解决这些问题,只能从根本上寻求解决方案。即从迁移到云转变为诞生于云。2013年Docker开源,之后Pivotal公司提出了云原生的概念,这是对云计算概念的全面升级。Pivotal指出云原生由容器、微服务、DevOps以及持续交付等技术[30]组成,并充分利用云计算优势构建和运行应用。2014年,容器编排技术Kubernetes发布。容器技术日趋成熟,在业界开始广泛应用。在2015年,云原生计算基金会(Cloud Native Computing Foundation,简称CNCF)成立。而到了2021年,CNCF已经孵化了超过120个项目、740名成员以及142000的贡献者^①。除了工业界,云原生在学术界也引起了关注,《计算机学报》发起了以“云原生”为主题的专刊征文^②。

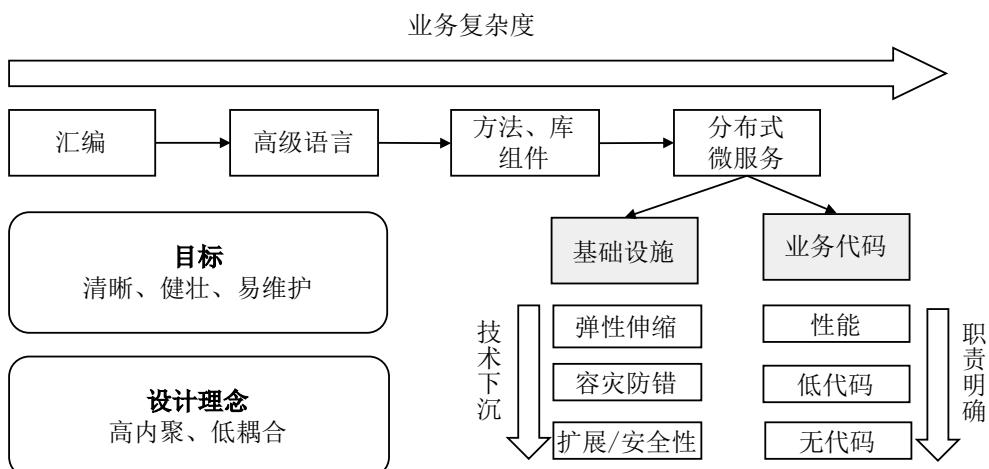


图 2-6: 云原生技术发展趋势

以软件工程的视角来看,如图2-6所示,随着软件规模、软件复杂程度在不断增大,软件上线速度不断加快,软件稳定性的要求在不断提高。围绕着“高内聚、低耦合”的设计理念,软件制品进一步深层次抽象,从单体架构演化为分布式微服务架构,从可复用的方法、库、组件进一步下沉到底层的基础设施。在这些客观需求的驱动下,敏捷进一步向运维端延伸,继瀑布开发、敏捷开发之后,开发运维一体化(Development and Operations,简称DevOps)成为又一新兴的软

^①CNCF2021 年年度报告

^②《计算机学报》云原生软件技术与工程实践专刊征文通知-CCF2021 中国软件大会

件发展理念和愿景。由于软件体量巨大,多个不同职责明确的团队负责整体软件项目的运行。DevOps 旨在通过一系列文化及技术手段(尤其是自动化 IT 工具链)打破开发和运维团队之间的壁垒,改善团队之间的协作关系,实现更加频繁快速、可靠的软件产品交付 [31]。DevOps 不仅向运维端扩展,其更涉及到软件全生命周期中的人、流程与平台。可以说,DevOps 是当下软件工程的第一生产力,其是一种普世的价值观。云原生作为一种基于云基础设施的技术体系涵盖了云应用定义、开发、构建与运行时的所涉及到的各工具或平台。云原生是 DevOps 生产力下的现阶段生产工具的体现,是 DevOps 的价值具象,“DevOps 时代下的云原生”就如“蒸汽时代的蒸汽机”。

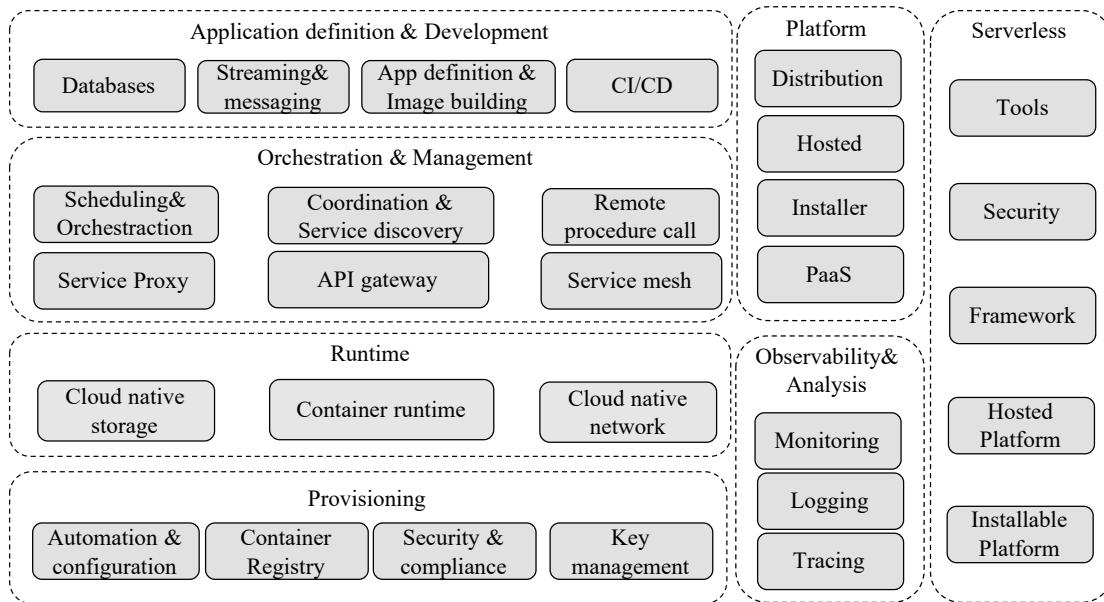


图 2-7: 云原生技术范畴

云原生技术囊括了 DevOps 的各环节。如图 2-7 所示,CNCF 定义了云原生的技术范畴,主要包括:

- 供应层 (Provisioning): 涉及云原生应用运行环境的自动化基础设施;
- 运行时 (Runtime): 指保障云原生应用程序正常运行所需的沙盒;
- 云应用编排与管理 (Orchestration and Management): 为云原生应用提供自动化编排和弹性伸缩能力,让云原生应用天然地具备可扩展性;
- 云应用定义与开发 (Application Definition and Development): 开发、构建、部署和运行应用程序的工具;
- 可观测性与分析 (Observability and Analysis): 全方位监控和分析云原生应用层的工具;

- 平台 (Platform): 主要指 Kubernetes, 将多类工具有机组合在一起解决庞大的工程问题;
- 无服务器 (Serverless): 提供函数级别更细粒度部署的一种新的云原生计算模型。

随着云架构的不断普及,“未来的软件一定生长于云上”的理念被越来越多的人所接受。云提供了一种面向企业应用按需进行资源分配的模型,以一种全新的高效的方式来部署应用。云原生是一系列基于云技术体系和企业管理方法的集合,既包含了实现应用云原生化的方法论,也包含了落地实践的关键技术。云原生应用利用容器、服务网格、微服务、不可变基础设施和声明式 API 等代表性技术,来构建容错性好、易于管理和便于观察的松耦合系统,结合可靠的自动化手段可对系统做出频繁、可预测的重大变更,让应用随时处于待发布状态。Gartner 指出,到 2022 年全球公共云服务市场预计将增长至约 3546 亿美元,60% 的组织将使用外部服务提供商的云管理服务 [32]。企业纷纷开始云化转型,希望将传统应用迁移到云端。

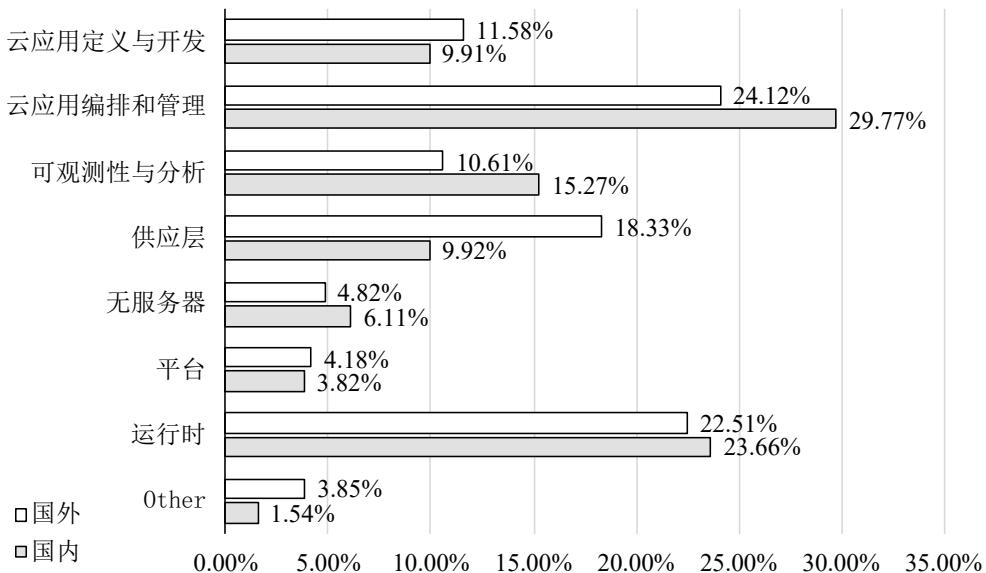


图 2-8: 国内外云原生技术现状

如图 2-8 所示,本文根据上述 7 类云原生技术范畴,对 2020-2021 年国内外重点技术会议的共 442 场分享进行了系统化分析,其中包括国内云原生社区 MeetUp 城市站以及 Cloud Native+Open Source Virtual Summit China 共 131 场分享,国际(欧洲、北美)KubeCon+CloudNativeCon Europe 以及 KubeCon+CloudNativeCon North America 共 311 篇场分享。从表中可知,目前国内外

研究重点的关注于云应用编排以及运行时, 国内对云应用编排与管理的讨论要大于国际并且云应用定义与开发流程、可观测性与分析、平台与无服务器方面国内外相关分享在比例上差距并不是很大, 云原生技术体系已经成为当代软件工程技术发展的主流技术体系。

2.2.2 Kubernetes

Kubernetes(简称 k8s) 是 Google 开源的容器集群管理平台。在容器化技术之上, Kubernetes 为容器化的云原生应用提供部署运行、资源调度、服务发现、弹性伸缩等一系列基础功能, 提升了大规模容器集群管理的便捷性。自开源来, Kubernetes 成为一种全新的基于容器技术的分布式架构解决方案, 在云原生领域具有举足轻重的地位。2019 年, 在我国 72% 的工程师已经在云原生生产环境中大规模使用 Kubernetes^①。

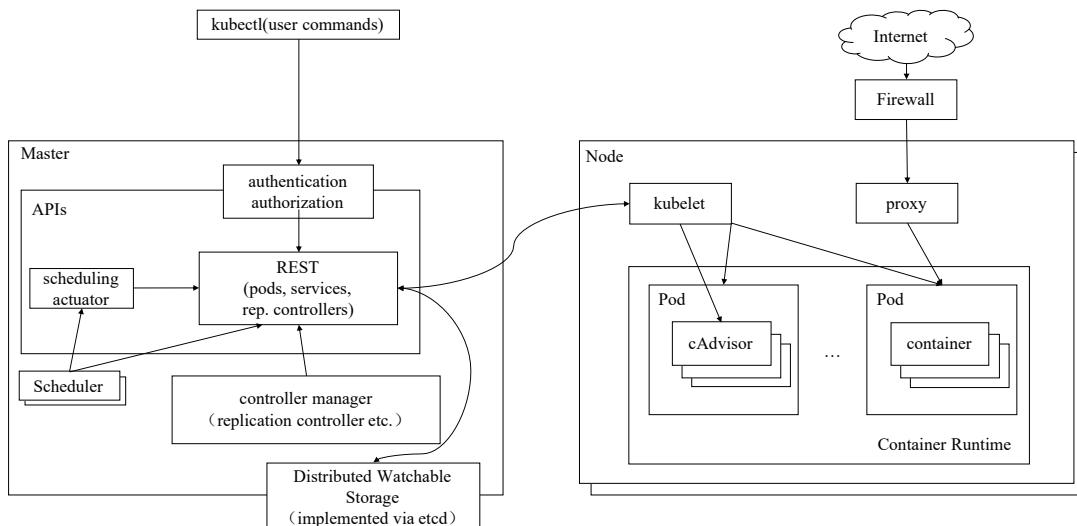


图 2-9: Kubernetes 架构图

Kubernetes 由节点代理 (kubelet) 和 Master 节点组成。如图 2-9 所示, Kubernetes 节点主要由以下核心组件构成:

- **Etcd:** 用于保存集群中一切网络配置和状态信息;
- **APIServer:** 是资源配置控制的入口, 具备完备的集群安全机制并且提供用于集群管理的 REST API 接口;
- **Controller Manager:** 集群内部的所有资源的管理控制中心, 负责集群内的

^①CNCF Cloud Native Survey China 2019

Node、Pod 副本、Endpoint、Namespace、ServiceAccount、ResourceQuota 的管理, 实现自动扩展、自动滚动更新、故障检测等功能;

- **Scheduler:** 根据特定的调度算法将 Pod 调度到指定的工作节点;
- **kubelet:** 定时检查获取节点上 Pod、容器的期望状态, 并调用对应的容器平台接口达到这个状态;
- **Container Runtime:** 负责镜像管理以及 Pod 和容器的真正运作 [33];
- **kube-proxy:** 为 Service 提供集群内部的负载均衡和服务发现。

Kubernetes 拥有独特的声明式 API 设计, 即声明式地告诉 Kubernetes 所需资源的状态, 而不是告诉它如何做。在 Kubernetes 中对象 (Object) 是持久化的实体, Kubernetes 使用这些实体去表示整个集群的状态, 同时这些对象可以在 Yaml[34] 文件中作为一种声明式 API 类型来灵活地创建并配置。典型的, Kubernetes 的对象主要有以下几种:

- **Namespace:** 能够隔离资源。Kubernetes 集群可以拥有多个命名空间, 这些命名空间在逻辑上彼此隔离, 实现了对多用户的资源隔离;
- **Pod:** Kubernetes 中最基本的操作单元, 包含一个或多个容器。Kubernetes 为每个 Pod 分配一个唯一的 IP 地址, Pod 内部的多个容器共享该 IP 地址。并且每个 Pod 都能设定自己的计算资源, 即 CPU 和 Memory;
- **Replication Controller(简称 RC):** 确保任意时间 Kubernetes 集群中运行指定数量的 Pod 副本;
- **Deployment:** 保证 Pod 的数量和健康, 绝大多数的功能与 RC 完全一样, 能够被当作全新一代的 RC;
- **Service:** 定义了一个 Pod 逻辑集合以及访问 Pod 的策略, 它提供一种桥梁会为访问者提供一个固定的 Pod 访问地址用于在访问时重定向到相应的后端;
- **Label:** 通过键值对的方式被附加到任何资源对象上, 用于配置资源;
- **Secret:** 不需要将敏感数据外露, 解决密码、Token、密钥等敏感数据的配置问题;
- **Role:** 一组权限的集合, 给某个 NameSpace 中的资源进行鉴权;
- **ConfigMap:** 为了让镜像和配置文件解耦, 应用程序会从配置文件、命令行参数或环境变量中读取配置信息, 以便实现镜像的可移植性和可复用性;
- **Volume:** Pod 中能够被多个容器访问的持久化共享目录;
- **Persistent Volume(简称 PV):** 类似于 Volume, Kubernetes 提供的存储资源

的抽象管理集群存储,其 API 内包含存储的细节实现;

- Persistent Volume Claim(简称 PVC): 对存储资源的请求声明,PVC 不关心底层存储实现的细节,只消耗 PV 资源。

随着 Kubernetes 生态的持续发展,上述常规的资源类型仅代表通用的对象,并不能适应多变的业务需求。为提升自身的扩展能力,Kubernetes 提供用户自定义 CRD 向 Kubernetes API 中增加定制化的资源类型。用户的自定义资源(Custom Resource,简称 CR)创建并注册到 Kubernetes API 后,其与常规通用资源对象都是原生的、存在于 etcd 中的同等资源,可以采用 Kubernetes 原生方式进行创建、查看。CRD 本质上用于声明用户自定义资源对象,开发人员还需要针对 CRD 提供关联的 Operator 对 CR 进行完整生命周期管控。

Operator 主要负责有状态应用(即 CR)及其组件的部署、更新、自动扩展、维护、数据备份等,保证它们的可用性。Operator 作为 Kubernetes 的一种扩展形式,基于 Kubernetes 的资源和控制器(Controller)概念构建,遵循 Kubernetes 原则,功能类似于 Controller Manager,但同时又注入了特定领域知识。

2.3 Helm

Helm 是一种基于 Kubernetes 云计算平台的打包和部署复杂软件应用程序的技术 [35]。随着云原生及微服务架构的发展,开发人员倾向于将大的单体应用分解成多个可以独立开发、部署、运维的微服务部署于 Kubernetes 之上。服务数量的增加为 Kubernetes 编排带来了复杂性,Helm 则通过软件打包的方式极大地简化了 Kubernetes 应用部署和管理的复杂性。

Helm 存在三个核心概念:

- chart: 即一系列包含创建 Kubernetes 应用实例必要信息的文件;
- config: 包含应用发布的相关配置信息;
- release: chart 及其配置的运行实例。

如图 2-10 所示, Helm 将 Kubernetes 资源(如 Deployment、Service)打包到 chart 中, chart 将会被保存到仓库中。Tiller 是部署于 Kubernetes 中的 Helm 的服务端,负责接受 Helm 请求并于 APIServer 进行交互,根据 chart 生成并管理 release。开发者使用 Helm 可以简化应用配置及版本管理,使得在 Kubernetes 上部署、升级、回滚、卸载应用程序更加方便。

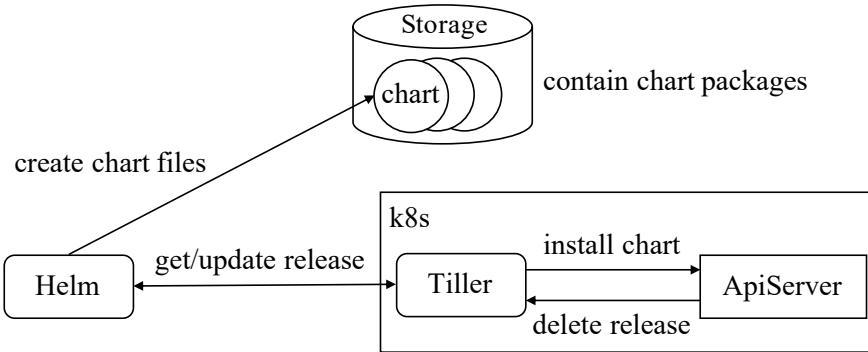


图 2-10: Helm 架构图

2.4 本章小结

本章第 2.1 节首先介绍了区块链基本概念、架构、分类，其次重点介绍联盟链解决方案 HF 的网络节点、账本结构，最后介绍区块链即服务的诞生和通用架构；第 2.2 节介绍云原生的发展历程、技术范畴以及 Kubernetes；第 2.3 节介绍本文区块链云化框架涉及到的其他相关技术。

第三章 区块链网络节点云化方案

本章首先通过快速评审获得 Kubernetes Operator 赋能质量属性的策略集, 随后阐述区块链云化框架的设计原则, 最后结合区块链去中心化等特性以及设计原则分析筛选策略集形成面向 Hyperledger Fabric 的区块链网络节点云化的核心具体实施方案。

3.1 快速评审

软件工程中, 快速评审 (Rapid Review) 是一种轻量级的二级研究, 以实践为导向专注于及时向研究人员提供证据 [36]。本文对已发表的学术文章进行快速评审, 快速评审的目的是为了在较短时间内了解目前学术界使用 Kubernetes Operator 进行云化的现状, 以及如何使用 Kubernetes 对现有系统进行赋能。随后, 本文对快速评审所得到的结果进行整理归纳得到 Kubernetes Operator 如何为质量属性赋能的策略集。

本文的快速评审过程分为以下步骤: 首先进行自动化的全文数据库检索, 再筛选出与 Kubernetes Operator 及架构改造强相关的文献, 接下来提取出质量属性及相关策略, 最后归纳整理出策略集。

表 3-1: 每个全文数据库的搜索字符串

全文数据库	搜索字符串	文献数
IEEE Xplore	(“kubernetes AND operator”) OR (“k8s AND operator”) OR (“custom resource definition”)	23
ACM	“kubernetes operator”OR “k8s operator”	7
Springer	((“kubernetes AND operator”) OR (“k8s AND operator”))) AND ((“custom resource definition”) OR ”CRD”)	0
ScienceDirect	(“kubernetes AND operator”) OR (“k8s AND operator”) OR (“custom resource definition”)	0
Scopus&Google	“kubernetes operator”	21(去重)

如表 3-1 所示, 为全面获取学术届对基于 Kubernetes Operator 云化的策略, 确定了本次检索的全文数据库以及搜索字符串。本次检索主要针对计算机与软件

工程领域的全文数据库[37](包含 ACM、IEEE Xplore、Springer、ScienceDirect),同时检索 Scopus 以及谷歌学术进行补充。最终,本文围绕“**kubernetes operator**”为检索主题共得到 51 篇论文。在文献筛选阶段,本文根据筛选标准筛选出 15 篇与 Kubernetes Operator 云化强相关的文献,入选文献如表 3-2 所示。文献筛选标准主要有:

- 论文的主要目的是利用 Kubernetes 对原有系统进行云化;
- 论文针对于 Kubernetes Operator 方法;
- 论文介绍了具体的改造策略及相关的质量属性。

表 3-2: 快速评审入选文献列表

文献编号	文献标题	文献引用
[P1]	Enhancement of observability using Kubernetes operator	[38]
[P2]	Designing a Kubernetes Operator for Machine Learning Applications	[39]
[P3]	Container orchestration on HPC systems through Kubernetes	[23]
[P4]	Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond	[40]
[P5]	On-the-fly fusion of remotely-sensed big data using an elastic computing paradigm with a containerized spark engine on kubernetes	[22]
[P6]	A Role-Based Orchestration Approach for Cloud Applications	[41]
[P7]	A Design of MANO System for Cloud Native Infrastructure	[42]
[P8]	Dynamic Updates of Virtual PLCs Deployed as Kubernetes Microservices	[43]
[P9]	Suture: Stitching safety onto kubernetes operators	[44]
[P10]	Automation of virtualized 5G infrastructure using mosaic 5G operator over kubernetes supporting network slicing	[25]
[P11]	5G Cloud-Native: Network Management & Automation	[24]
[P12]	Proposed model for distributed storage automation system using kubernetes operators	[45]
[P13]	Monitoring Resilience in a Rook-managed Containerized Cloud Storage System	[46]
[P14]	Reproducible Benchmarking of Cloud-Native Applications With the Kubernetes Operator Pattern	[21]
[P15]	Pivotal Greenplum®for Kubernetes: Demonstration of Managing Greenplum Database on Kubernetes	[47]

在数据提取阶段,本文对 5 个提取项(文献标题、文献发表年份、质量属性、策略、所属领域)进行抽取,共得到 44 条针对不同质量属性的相关策略。在数据归纳阶段,由于不同的文献中对语义相同的质量属性用词存在明显差异,

本文根据国际软件质量评价标准 ISO/IEC 25010:2011^①所定义的质量模型对收集到的文献中表述的质量属性进行映射, 同时对相同或相似的策略进行整理合并, 得到策略集如表 3-3 所示。

表 3-3: Kubernetes Operator 云化策略集

文献中质量属性	ISO 质量属性	编号	策略	参考文献
可迁移性	可移植性	T1	沙盒	[P1-P3, P7-P8, P10-P12]
可用性	可靠性	T2	升级重启	[P2, P15]
		T3	主备切换	[P15]
		T4	监控	[P1, P5, P12, P13]
生产效率	易用性	T5	支持用户初始化	[P1, P2, P12-P15]
		T6	支持系统初始化	[P1, P4, P13]
可扩展性	无	T7	基于监控指标的伸缩处理	[P2, P6, P13]
		T8	数据分发	[P15]
安全性	安全性	T9	限制访问	[P2, P9]
		T10	认证授权	[P3, P13]
		T11	分离实体	[P9, P13]

3.2 设计原则

区块链云化框架致力于在 BaaS 一站式构建、管理、托管和运维区块链网络及其应用的基础上更深入云原生底层基础设施 Kubernetes 的底层, 有效利用云能力对 HF 基础设施节点赋能。BaaS 在设计上以简单易用、成熟可扩展、安全可靠、可视运维等为主要方向^②。区块链云化框架需要在上述设计原则的基础上进行适配与拓展:

- 简单易用: BaaS 平台需要具备帮助企业实现自动配置, 快速启动区块链的能力。区块链云化框架需要在 Kubernetes 上完成 HF 的全生命周期管理。在 Kubernetes 上启动管理 HF 网络并部署链码需要专业的领域知识, HF 各节点的配置项繁多且与 Kubernetes 适配极易出错。区块链云化框架需屏蔽底层 HF 配置与 Kubernetes 逻辑, 帮助用户采用命令行配置的方式提供完整的 HF 各组件的全生命周期管理;
- 灵活扩展: BaaS 平台设计应采用抽象架构以及可插拔的模块。在此原则

^①ISO/IEC 25010:2011 System and software quality models

^②2019 区块链即服务平台 BaaS 白皮书

上, 区块链云化框架需按照模块化配置, 将 Kubernetes 底层的计算资源、存储资源、网络资源等供给 HF, HF 的 Ca、Peer、Orderer 基本节点的证书认证、共识、TLS 加密、存储等功能模块作为可配置项进行命令式配置; 区块链云化框架需要能够确保系统核心底层逻辑稳定运行的同时, 对外提供小而精的扩展边界, 实现系统的高内聚与低耦合;

- 安全可靠: BaaS 平台应具备有效的安全机制保证区块链服务的安全性。区块链云化框架为满足上述原则, 需要具备有效的认证、鉴权、准入机制来确保区块链系统的安全性; 具备可插拔的共识算法确保区块链的去中心化、可追溯等特点, 支持完善的用户密钥授权、保存、隔离处理以及提供可靠的故障恢复能力;
- 可视运维: BaaS 平台要提供必要的运维接口和运维授权的能力。当前区块链系统缺乏一套涵盖不同层面的标准方法来监控区块链及其智能合约的运行。区块链云化框架需提供基本运维能力, 有效复用云上监控方案为区块链系统提供 7*24 小时可视化资源监控能力;
- 云链结合: BaaS 平台需要结合云平台提供各种区块链所需的丰富资源。在此原则之上, 区块链云化框架需要具备高可迁移性, 具备基础架构的云独立性。本框架依托 Kubernetes, 需要具备迁移到支持 Kubernetes 任何云的能力;
- 合作开放: BaaS 平台应该和各个行业伙伴共同打造行业可信的区块链生态。为此, 区块链云化框架应该具备反馈机制的透明性。区块链云化框架需保证区块链上所有的交易记录是可追溯的、不可篡改的, 区块链交易过程以及获取交易产生的记录需要专业人员才能获取操作, 非专业人员难以理解, 框架需通过简单透明的方式获取交易记录。

3.3 策略集应用

通过快速评审的方式形成的基于 Kubernetes Operator 云化策略集作为指导方案, 需要贴合区块链领域自身约束才能有效纳入区块链云化框架中落地。本节将围绕“所选策略为何能提升质量属性”以及“所选策略是否适用于 HF”进行阐述。

T1: 沙盒

原理: 沙盒将系统实例与现实世界隔离开来, 使系统运作能够消除现实世界的影响, 虚拟机与容器化都提供了沙盒的能力。相较于虚拟机而言, 容器化在

云原生应用程序中的拥有更高的部署效率 [23]、可迁移性。容器化可以使每个软件应用程序在隔离的环境中运行, 其将应用程序及库、配置文件和其他依赖项封装在一起, 确保了环境兼容性, 从而使用户能够轻松地在不同的环境中移动和部署程序。因此, 容器化使得应用程序具备极强的可移植性。Docker^①是容器化的典型代表, 其可以将运行态的容器打包成镜像 (image) 并储存在在线存储仓库中。Docker 容器不是虚拟机, 这意味着它可以使用宿主机现有的网络接口 [48]。一旦创建了容器, 就可以为容器提供一个专用回环接口, 用于内部通信, 完成部署。

适配性: HF 官方提供了 Binaries 和 Docker 镜像两种安装方式^②, 其天然满足容器化构建。

T2: 升级重启

原理: 当系统某部分发生故障时, 需要能够快速的从故障中重新引入新的可用组件。Kubernetes 作为容器管理调度平台, 其具备自恢复能力。Kubernetes 会自动打包应用程序, 并根据预设需求运行 Pod 并管理容器。Pod 内置标准的自重启策略的字段 (restartPolicy), 首先 Kubernetes 会对 Pod 进行健康检查, 若存在故障则根据 Pod 的自重启策略对其进行重启。利用这种机制, 可以自动重启在执行过程中失败的容器, 并杀死那些没有响应用户定义的健康检查的容器。但如果节点本身死亡, 那么它会替换并重新安排发生故障的容器到其他可用节点上。

适配性: 根据 T1, HF 网络采用容器化构建, 自然就可以通过 Kubernetes 托管。然而, 托管于 Kubernetes 需要从零开始为 HF 不同类型的网络节点提供编写多种的资源对象, 包含但不限于 Deployment、Service 等。一旦 HF 网络节点托管于 Kubernetes, 即可原生地利用 Kubernetes 进行高可用的管理。

T3: 主备切换

原理: 在具备严格高可用的系统中, 往往需要处理主备切换。主节点就是提供服务的节点, 备份节点就是不提供服务的控制节点。所谓的主备切换就是当主节点故障时, 系统自动切换备份节点接管服务。要保证主备切换往往涉及到多种复杂的机制, 如数据同步、探活、网络切换等。所有冗余组件并行响应事件, 所有组件都处于相同状态。但响应只使用了一个组件, 其余组件被丢弃。当发生故障时, 停机时间通常不存在, 因为备份是当前的, 唯一的时间是恢复时间。

Kubernetes Operator 可以协助实现目标系统的主备切换任务, 并不丢失数据。

适配性: 区块链网络是去中心化的网络, 所有的节点共同参与记账, 只需要

^①Docker 官网

^②fabric_bootstrap.sh

网络中的大多数(超过 51%)节点的账本状态一致即可,并不需要所有网络中的所有的 Peer 节点时刻保持高可用状态。针对于 Peer 的主备切换意义并不大,针对 Orderer 的主备切换目前仍然是研究的空白。

T4: 监控

原理: Prometheus[49] 作为云原生领域的监控事实标准,其是一个时序数据库又是一个监控系统,有着强大的功能和良好的生态。Grafana 可以与各种其他类似于 Prometheus 的数据源进行交互并进行可视化。基于 Prometheus 的监控体系可以收集开放的监控指标并进行多维度数据模型的灵活查询,极大增强了被监测系统的可监控性。

适配性: Mahajan 等人 [44] 指出,只有 58% 的人认为他们有工具来理解 Operator 在异常情况下的行为,并且他们强调了对 Operator 事件日志与监控的不满。HF 网络各节点的运行终态是 Pod,利用 Prometheus 监控体系可以无侵入式的收集 HF 网络各节点的监控指标,并将指标数据以 Grafana 图表的方式展示,做到 7*24 小时可视运维。

T5: 支持用户初始化

原理: 支持用户初始化,即支持用户纠正错误或提高效率。框架需要帮助用户自动化配置 HF 领域知识,提升用户启动 HF 网络的效率。Kubernetes Operator 提供的 CRD 方式可以插拔式地将领域知识注入进云基础设施内部,CRD 就是领域专家自定义的资源范式,其规定了领域资源应该具备何种属性。特定领域专家编写的 CRD 能够注册进入集群,注册之后领域的开发人员即可以根据 CRD 定义的属性编写特定于自身业务的 CR。CRD 降低了集群管理人员对 HF 网络的二次学习成本,集群管理人员无需了解特定领域的领域知识,仅需要通过 kubectl 原生的方式即可管理陌生领域的应用程序。

适配性: 可以将 HF 网络节点通过 CRD 注册进入 Kubernetes,然而这存在一定的复杂性。编写 HF 领域 CRD 需要根据各网络节点的配置^{①②③}以及网络预期状态设计注入的属性。以 Ca 为例,其存在 22 种一级配置项,在编写 Ca CRD 时就需要对这些配置项进行甄别与筛选,选取出适合插拔配置以及使用频率较高的配置并利用编程语言对其进行编写转换。

T6: 支持系统初始化

原理: 本策略需要做到支持用于预测自身行为或用户意图的模型,具体表现

^①Ca Config

^②Orderer Config

^③Peer Config

为：框架可以维护一个任务模型，这能够使得框架了解用户正在尝试什么并实时提供反馈。本框架可以利用 Operator 与 Helm 相结合的方式，通过 Controller 协调机制时刻监听用户 CR 的状态并将状态更新到 Helm 最终反馈到集群中。由于 Helm 能够自动化地在 Kubernetes 内构建部署应用程序，但这是一个一次性部署动作，仅仅完成了一次性自动化构建、部署 HF 网络的任务。本质上与现有 Blockchain Automation Framework 采用脚本方式部署无异，这虽然能在一定程度上提升 HF 网络的部署效率，但这无法将 HF 复杂的领域知识以插拔化的方式配置进入云基础设施内部。通过 Operator 与 HF Helm 的结合，不仅能够提升 HF 网络的部署效率，也可以时刻监听 Helm 状态实现对整个网络进行完整生命周期管理。

适配性：根据 T2，将 HF 网络托管于 Kubernetes 就需要为其配置资源对象。一种是分别配置所需资源对象，其次就是利用 Helm 为 HF 网络节点配置资源对象模板。若采用 Helm，需要利用模版语言配置 HF 网络节点，并在 Operator 中对模板进行填充。

T7: 基于监控指标的伸缩处理

原理：自动弹性伸缩能够基于资源使用情况对工作负载进行伸缩处理。Kubernetes 具备强大的伸缩能力，其可以自动打包应用程序，并根据预设需求运行 Pod 并管理容器。利用自身机制，Kubernetes 可以轻松根据 CPU、内存对工作负载进行横向扩容 (Horizontal Pod Autoscaler，简称 HPA) 或纵向扩容 (Vertical Pod Autoscaler，简称 VPA)。但是，在多种场景中需要按照业务监控指标进行扩容操作，本策略能够利用定时的方式，在 Operator 中收集业务监控指标对目标工作负载进行多副本的伸缩处理。

适配性：无论是基于 Kubernetes 的多副本伸缩还是基于监控指标进行伸缩仅保证的是无状态应用的高可用性，并不解决数据一致性问题，所以为某个 Pod 节点增加多副本进行可伸缩意义不大，利用 Kubernetes 的自恢复能力保证单 Pod 节点可用即可。

T8: 数据分发

原理：虽然针对 Peer 节点的可伸缩性意义不大，但从海量数据存储的角度来看，使用云存储是加强区块链网络存储一种替代方法，它减少了因区块存储空间有限而造成的限制 [5]。云可以为区块链网络提供存储即服务 (Storage as a Service，简称 SaaS) 作为一种可扩展的链外解决方案。Kubernetes 的 SaaS 通过服务的形式精细化使用存储资源，实现了对存储定义 (PVC) 和存储申

请 (StorageClass) 的灵活分离。这种链外存储机制可以使得 HF 网络能够通过 Kubernetes 的 SaaS 机制挂载外部存储单元而不必关心存储资源的实际物理位置, 能够做到存储资源的即插即用。

适配性: HF 网络的世界状态是一种脱链的链外附加缓存机制, 其最终被保存在链外的 LevelDB 或 CouchDB 中。同时, 世界状态的丢失并不会对区块链中的数据产生影响。除世界状态的存储, 针对网络节点以及链码的存储都可以通过定制“PVC+StorageClass”机制对外进行挂载。

T9: 限制访问

原理: 访问控制是提供云数据安全和隐私的一种基本方法, 可以防止未经授权的用户侵入云数据。不可靠的访问控制方法也会影响其他功能, 如身份验证、授权和数据审核。除自定义访问控制机制外, 云中的传统访问控制方法主要基于成熟的访问控制策略, 现有的传统访问控制策略分为四种: 自主访问控制 (Discretionary Access Control, 简称 DAC)、强制访问控制 (Mandatory Access Control, 简称 MAC)、基于角色的访问控制 (Role-Based Access Control, 简称 RBAC) 和基于属性的访问控制 (Attribute-Based Access Control, 简称 ABAC), 其对比如表 3-4 所示。

表 3-4: 访问控制机制

名称	机制	优点	缺点
DAC	根据被操作对象的权限控制列表或者权限控制矩阵决定用户的是能否对其进行操作	操作简单	权限控制分散, 不利于管理
MAC	每个对象含有权限标识, 每个用户也含有权限标识, 用户能够操作对象取决于双方权限标识的关系	适合机密或等级严格的场景	缺乏灵活性
RBAC	一个用户关联一个或多个角色, 一个角色关联一个或多个权限, 根据权限需要创建角色并与用户绑定	灵活易扩展、职责分离	未提供操作顺序控制机制
ABAC	根据规则动态计算一个或一组属性是否满足某种规则并授权判断	集中管理、按需实现不同粒度权限控制	不直观、规则复杂过多会出现性能问题

云化框架需要在 Kubernetes 上部署 HF 节点和管理 HF 网络, 需要限制 Kubernetes 上其他用户对 HF 网络节点运行时的权限。在 DAC 中, 合法用户 (如云服务提供商) 根据自主访问控制策略来允许其他用户 (如云用户) 访问操作对象 [50], 同时阻止非授权的用户访问。DAC 操作简单, 云用户能够进行灵活的访问控制, 然而 DAC 不需要严格的规则造成了权限管理的过于分散, 不利于综合管理; MAC 则解决了 DAC 权限控制过于分散的问题, MAC 是通过预定义的信

任策略实现且该策略不能动态更改, 目标是限制访问的用户对目标对象的操作能力, MAC 过于强调保密性, 管理缺乏灵活性; ABAC 将访问规则写在配置文件里以实现多粒度的权限控制, 但是由于定义权限时不易看出用户和对象间的关系, 很容易给集群管理者带来运维上的麻烦; RBAC 基于角色的权限控制体系, 能够实现“以岗定人”的目标。

适配性: Mahajan 等人 [44] 指出, 47% 的 Operator 缺乏安全性的配置。利用 RBAC 可以从 Kubernetes 层面对 HF 网络节点添加安全权限机制。虽然 RBAC 没有提供顺序操作的控制机制, 但能兼容 Kubernetes, 有效的为操作 HF 网络各类型节点提供安全保证机制, 防止集群中其他用户恶意增删 HF 节点。若现有的访问控制机制不能满足需求, 可以创建自定义的访问控制机制实现更加符合业务实践的安全权限管理。

T10: 认证授权

原理: 在设计软件系统时, 需要考虑如何让应用在各种复杂的环境中保证安全的访问。这中间涉及到针对用户的认证、授权。认证意味着系统识别合法用户, 如通过账号密码进行登录认证; 授权则是认证通过之后合法用户在系统中所能处理的问题。常见的认证包括信道 SSL/TLS 上的认证、协议 HTTP 上的认证以及内容 Web 上的认证, 常见的授权机制包括 T9 提到的访问控制机制授权以及 OAuth2 授权等。

适配性: 由于 HF 网络是基于 MSP 的认证性网络, 是基于 SSL/TLS 的认证机制, 这种机制可以保护 HF 网络内部交易流程的安全性。这个过程包含了加密算法、生成密钥、公钥的分发、Ca 认证、签名验证等各项复杂的过程, 所有的 HF 网络的参与者都需要提供可认证的身份信息, 即 HF 网络利用 SSL/TLS 作为本身的多用户认证授权的机制。云化框架需要自动化的兼容基于 TLS 的认证机制, 除此之外, 传统手动生成的 HF 网络密钥以文件的形式保存, 增加了密钥泄漏的风险, 云化框架需要结合 HF 的 SSL/TLS 机制提供安全的证书管理策略。

T11: 分离实体

原理: 使用 Namespace 来隔离实体工作负载是一项基本的安全最佳实践。若不通过 Namespace 进行项目级别的资源隔离, Kubernetes 中所有的工作负载是可以相互访问的。Kubernetes 可以将内部的资源划分到不同的 Namespace 中以形成逻辑上的隔离。同一 Namespace 下的进程可以感受彼此的存在, 而对外界一无所知, 这可以实现与其他工作负载的资源与安全性隔离。

适配性: 尽管 Namespace 是官方的简易强制执行资源隔离的方式。Mahajan

等人 [44] 指出, 48% 的受访者报告称在生产中使用了多个 Operator, 但只有三分之一的受访者表示他们使用了标准的资源隔离技术。可以通过结合 T9 的 RBAC 将 HF 网络放在单独的 Namespace 下交给不同的角色进行管理, 这有利于增强 HF 网络的资源安全性以及访问控制安全性。

除上述策略外, HF 致力于构建透明、公开、去中心化的企业级分布式应用, 区块链云化框架需要在以下方面满足透明性原则。首先, 区块链云化框架构建 HF 网络过程的透明性, 区块链云化框架需要具备完备的日志体系, 日志能够提供记录区块链云化框架操作 HF 网络的行为以及网络状态, 并能够规范的表达出来; 其次, 对于链上交易而言, HF 网络本就是公开透明的。区块链云化框架不能屏蔽这一特性, 需要对 HF 网络账本具备更加简易、公开、透明的命令式查询渠道, 如命令式查询交易记录。

综上, 如图 3-1 所示, 在区块链 HF 网络场景下结合云化框架设计原则, 剔除了 Kubernetes Operator 云化策略集中不适用于 HF 网络的策略, 最终形成了面向 Hyperledger Fabric 的区块链网络节点云化框架的核心策略实施方案。



图 3-1: 策略集应用

3.4 本章小结

首先详细介绍了快速评审的过程并得到基于 Kubernetes Operator 云化策略集;其次,本章重点介绍了面向 Hyperledger Fabric 的区块链云化框架所应满足的设计原则;最后结合 HF 网络特性以及区块链云化设计原则对策略集进行筛选,形成了面向 Hyperledger Fabric 的区块链网络节点云化的核心策略实施方案。

第四章 智能合约微服务化开发运维方法

本章首先详细分析智能合约开发运维过程中遇到的挑战，并对比智能合约与微服务架构在设计原则上的相似性。基于这两者的相似性，本章阐述了智能合约微服务化改造过程以及智能合约微服务化开发运维流程。最后，本章介绍该方法的原型工具并对所提方法进行评估。

4.1 研究目的

区块链具有分布式、不可篡改、去中心化、历史可追溯等特点。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、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。智能合约的引入有效的将上述特点落地到应用中。然而当前业界缺少促进去中心化应用价值交付的有效手段。具体地：

(1) 研究的重点是利用区块链技术特性应用在不同领域 [9]，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 [51]。目前去中心化应用纷纷从概念验证阶段转化为工程化实践阶段。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 [52]，使用门槛高。

(2) 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 [53]。智能合约的部署通常采用的纯手工或脚本方式 [54][55]，消耗大量的时间成本。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，HF 工程师调试智能合约困难，他们开发无漏洞智能合约具有挑战性。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 [56]。

(3) 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 [57],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,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。随着 DevOps 理念的兴起,DevOps 与微服务架构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。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,支撑微服务开发运维的自动化工具不断增多,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。为此,本节提出一种智能合约微服务化开发运维方法。在该方法中,首先进行智能合约微服务化改造,经改造后的智能合约能够利用各种成熟的自动化工具支撑其开发运维工作。该方法扩充了智能合约开发运维工具库,提升了智能合约开发运维的自动化水平,最终能够提高去中心化应用的价值交付能力。本章的智能合约微服务化开发运维方法已经公开发表于国内高水平期刊《软件学报》。

4.2 设计原则

在智能合约微服务化开发运维方法之前,需要考虑两者在设计原则上的相似性,只有这两者在设计原则上相似才有可能进行智能合约微服务化改造。

微服务架构提倡将单一应用程序根据业务垂直划分成一组相互独立的小服务,服务之间相互协调、互相配合,为用户提供最终价值。每个服务运行在其独立的进程中,服务和服务之间采用轻量级的通信机制相互沟通。每个服务都围绕着具体的业务进行构建,并且能够被独立的部署到生产环境、类生产环境等,这些服务可以使用不同的编程语言进行开发,可以自由根据业务需求进行横向扩展。与此同时,这些微服务分配给不同的团队进行开发维护,做到对整个应用程序以及团队的解耦,由于微服务的生命周期由小而精的团队全权负责,它们更容易维护且容错能力更强,单独微服务的宕机不会影响破坏整个系统。

另一方面,智能合约作为运行在区块链上的计算机程序,极大丰富了区块链的功能,使得区块链不仅仅是分布式账本,并且能够完成一定程度的业务能力。在区块链出现之前,信任的建立是基于诚信的中心化组织和个人。在区块链网络中没有中心化的记账机构,其以密码学为基础,所有的分布式节点执行共识机制来保障分布式账本不可伪造与不可篡改的特性,所以用户可以信任区块链系统的数据。智能合约可以执行任何类型的计算,由区块链驱动将会要求双方遵守他们的承诺并且智能合约永久储存且公开化,这使得区块链从数据可信达到了业务可信。如表 4-1 所示,微服务与智能合约在设计原则上有许多相似性 [58]。

智能合约能够根据业务需求定义良好的边界。在边界内,每个智能合约要汢单一职责,专注于自己的需求目标并且要完美的完成目标。但是,目前智能合约在自动化、灵活性等方面还有很多欠缺,因此如果将智能合约进行微服务化,就能够借助微服务领域成熟的流程与技术规范智能合约的开发运维工作,促进去中心化应用的价值交付流程。

表 4-1: 智能合约与微服务设计原则 [58]

微服务设计原则	是否适用智能合约	理由
小而精,专注于功能	适用	每个智能合约都应该关注做很少的事情,但要把这些做好
职责分离	适用	智能合约通常范围有限并负有全部责任
全栈	部分适用	根据上下文,全栈或拆分智能合约可能是更好的选择
独立更新,无需停机	部分适用	更新可以是独立的,但依赖于治理,这是对智能合约的信任来源之一
无状态	很少适用	典型的合同带有状态

4.3 智能合约微服务化改造

本节探寻如何利用微服务思想改造智能合约,使改造后的智能合约能够有效利用传统自动化工具支撑其开发运维工作。由于微服务的设计原则在很大程度上可以适用于智能合约,这是本节能进行智能合约微服务化改造的基础。SpringBoot^①是开发微服务的常用框架,当前部署 SpringBoot 制品主要有以下两种方式:

^①SpringBoot

- Jar 包: 开发人员直接将编写好的 Jar 包以命令的方式运行在环境中的某端口上, 并对外暴露服务;
- 镜像发布: 开发人员将编写好的软件制品打包成 Docker 镜像, 打包好的镜像不仅可以直接运行并暴露端口对外服务, 而且可以托管于 Kubernetes 发布服务。托管于 Kubernetes 中的流程如图 4-1 所示。

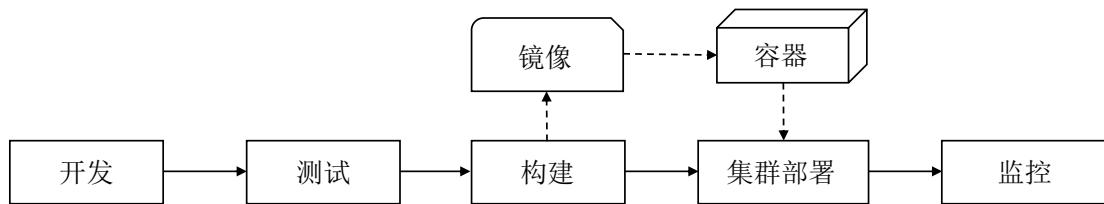


图 4-1: 镜像方式发布微服务流程

由于当前智能合约与区块链节点之间是强耦合的关系, 这使得我们很难利用现有工具单独对智能合约进行独立开发、测试、部署、运维。因此, 本节进行的智能合约微服务化改造流程如图 4-2 所示。

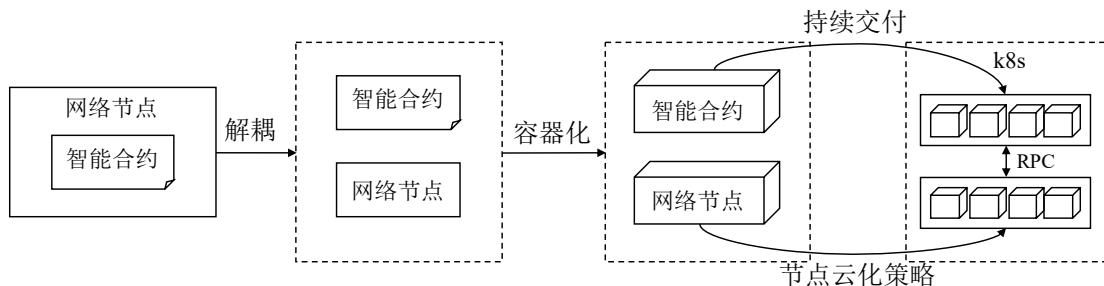


图 4-2: 智能合约微服务化改造流程

(1) 运行环境

智能合约微服务化改造的目的是解耦, 让智能合约更容易拓展、更富有弹性。在传统的链码安装与实例化的方式中, 链码的构建和运行是 HF 节点的一部分, 链码和节点之间的耦合度较高, 不具有可复用性。解耦后的智能合约更加专注于自身功能, 不需要关心账本数据的一致性、网络的安全性等问题。解耦后, 需要给智能合约提供容器环境使其可以运行。可以说, 容器化是智能合约与共识节点解耦后平稳运行的基础, 是智能合约微服务化的有效手段。本节将智能合约与网络节点解耦后具备以下优势:

- 单一职责: 职责愈加明确, 每个智能合约容器只需要完成自己特定的功能。独立的智能合约能够有效利用现有自动化工具完成持续交付;

- 高融入性^①: 智能合约将服务封装成 API, 用户可直接将智能合约安插在自己的业务逻辑中;
- 高可用性: 利用 Kubernetes 本身对于高可用性的保障, 只需要在 Kubernetes 层面进行简单的配置(例如 Replica、QoS、HPA 等), 不必针对区块链层面专门进行高可用策略配置;
- 高可扩展性: 智能合约进行容器化编排后, 只需要将自己的服务暴露给其他节点及应用。当智能合约需求发生变更时, 只需向集群内部增加新变更的智能合约容器即可。

(2) 通信机制

智能合约解耦后需要为智能合约与网络节点提供良好的通信机制, 只有合理的通信才能保证智能合约正确的运行。

如表 4-2 所示, 服务间通信的方式主要有两种, 分别是远程过程调用 (Remote Procedure Call, 简称 RPC) 和表述性状态转移 (Representational State Transfer, 简称 REST)。这两种方式各有利弊, 存在不同的使用场景。本节的智能合约微服务化改造为保证区块链网络的性能选择 RPC 的通信方式。

表 4-2: 通信机制对比

	RPC	REST
场景	适用于内部的函数方法调用	适用于外部资源接口
优点	性能表现出色	便于开发人员理解、调试
缺点	与代码耦合度强, 不方便理解	性能表现较差

(3) 持续交付

持续集成 (Continuous Integration, 简称 CI) 是一种软件开发实践, 团队成员经常集成他们的工作, 通常每个人至少每天都进行集成。每个集成都由自动化的构建(包括测试)进行验证, 以尽可能快地检测集成错误^②。持续交付 (Continuous Delivery, 简称 CD) 的关注点不在于源码而在于可交付物。

借鉴微服务持续交付的思想, 智能合约从编写完成后就要不断的集成交付。只有持续不断的交付才能持续验证智能合约与下层的区块链云化节点结合的功能完整性。当前, 可交付物从代码制品变为智能合约镜像, 借鉴现有的 CI/CD 工具(例如 Jenkins、Travis CI 等)帮助智能合约微服务化开发快速构建持续交付流水线, 可以实现:

^①发明专利: 一种建立智能合约微服务化的方法

^②Martin Fowler. Continuous Integration.

- 基础设施自动化,为智能合约微服务化开发降低了构建、部署、交付操作难度,减少集成交付过程中的时间开销;
- 频繁的构建升级智能合约,更早的识别错误,有效控制风险,保证代码质量;
- 有效的帮助团队成员之间的协作,促进代码共享,团队中每个人都能看到构建交付结果。

4.4 智能合约微服务化开发运维流程

本节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 4-3 展示了智能合约微服务化开发运维流程以促进智能合约价值交付过程。该图展示的是在已经搭建好的区块链云化网络中,使用现有自动化工具无侵入的支撑智能合约完整生命周期。

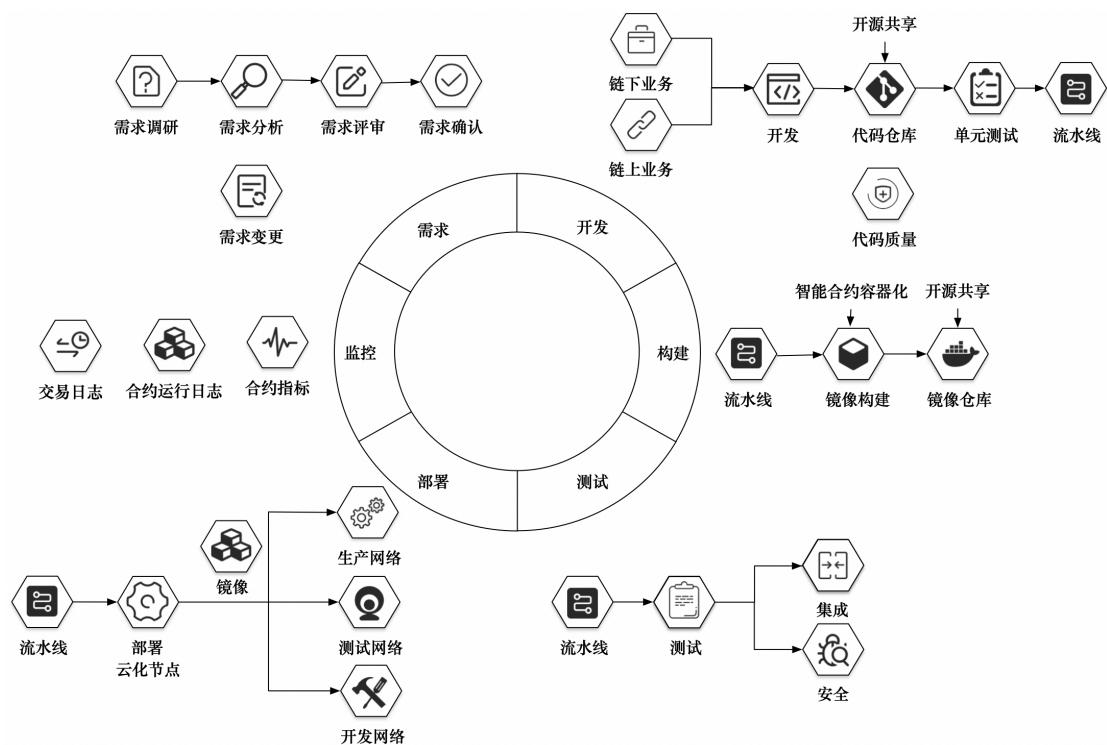


图 4-3: 智能合约微服务化开发运维流程

(1) 需求

区块链网络是一个典型的分布式网络,在理论计算机科学中, Eric Brewer 提

出 CAP 定理^①, 分布式数据存储系统中只能同时满足以下三个中的两个:

- 一致性 (Consistency): 所有节点数据一致性更新;
- 可用性 (Availability): 系统是否还能在正常时间内响应用户的请求;
- 分区可用性 (Partition Tolerance): 尽管节点之间的网络丢弃 (或延迟) 任意数量的消息, 系统仍将继续运行.

分布式系统必须满足分区可用性, 即某个节点宕机不会影响其他节点的正常工作, 达到高可靠性。在区块链系统中, 要求所有节点共同维护相同的账本, 所以必须满足一致性原则。在满足上述两个原则的条件下就要牺牲可用性, 系统的响应时间则会变慢。

将一个大的复杂问题, 变成多个小问题解决, 体现了微服务架构的分而治之的思想。合理的微服务划分是微服务架构成功与否的关键, 但是微服务架构是手段而不是最终的目的。智能合约微服务化改造的最终目的是解耦, 以业务为根本出发点进行抽象划分。为了使去中心化应用具备良好的响应性能, 所以在需求分析阶段智能合约需要额外考虑微服务架构在数据处理响应时间方面的区别。区块链的新数据要经过多个节点消耗大量的计算资源才会产生且数据会有许多处冗余的备份, 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智能合约代码处理, 仅将需要达成共识的数据上链即可。

(2) 开发

当链上业务代码编写完成后托管于代码仓库, 由于现如今智能合约很少开源, 本文提倡以代码库或镜像库的形式分享有用且经过市场检验的智能合约, 促进入界交流, 减少重复开发。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。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 CI/CD 工具, 做到合并代码后自动提交流水线交付。

(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/CD 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

^①CAP Theorem

常见的智能合约漏洞包括重入攻击、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 [59], 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

在智能合约微服务运行时采用已有监控工具全面采集智能合约容器以及下层区块链云化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、报警、智能决策分析以及可视化。常见的监控指标有:

- 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。

4.5 原型工具与方法评估

由于本章的智能合约微服务化开发运维方法已经公开发表, 故本节将简要描述基于智能合约微服务化开发运维方法的原型工具 Mictract 核心工作原理以及智能合约微服务化开发运维方法的评估过程。

如图 4-4 所示, Mictract^{①②}作为传统 BaaS 平台能够为构建区块链应用程序的人或组织提供第三方区块链服务, 允许用户构建、托管、运行、监控自己的区块链应用。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。值得注意的是,

^①Java for Mictract

^②Go for Mictract

Mictract 搭建的下层区块链网络与传统 BaaS 平台类似, 只是简单的将区块链网络迁移上云。因此, Mictract 在下层网络节点云化方面存在一定的不足, 需要整合区块链节点云化方案形成完整的区块链云化框架。

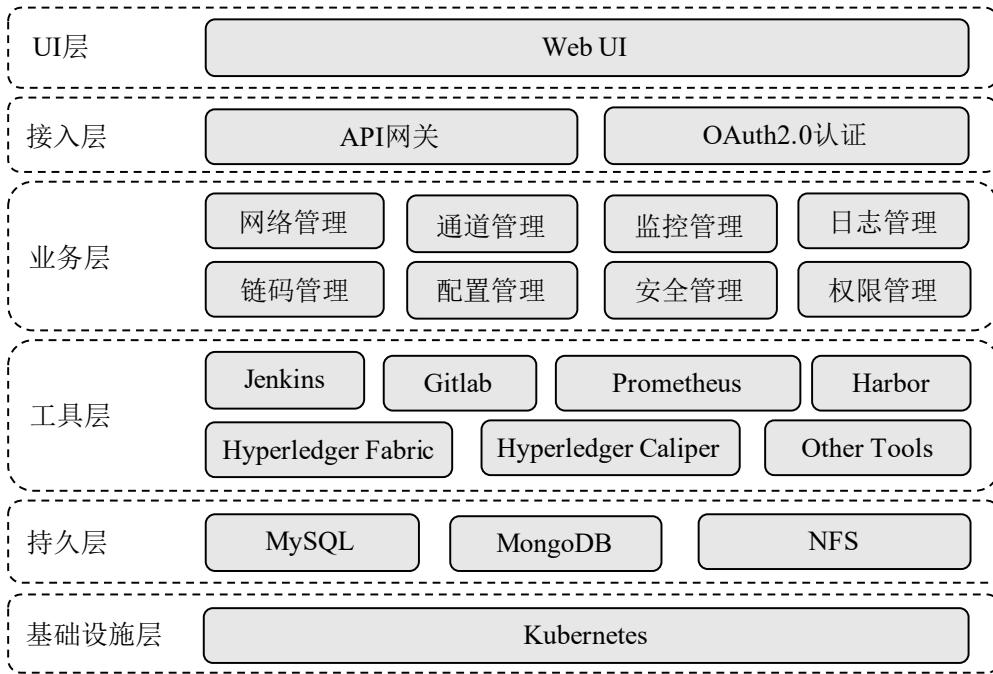


图 4-4: Mictract 架构图

将官方 HF 的链码打包成 Docker 镜像并将这些镜像部署于 Kubernetes 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。HF 2.0 引入了链码新生命周期, 工程师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 packageID。原型工具利用伪代码 4.1 将链码打包成 Docker 镜像。当需要部署链码时, 利用伪代码 4.2 配置链码的部署文件, 拉取链码镜像并为其配置环境变量, 最终生成相应的 Kubernetes Pod 容器作为独立的运行单元。除了对上述镜像及 Deployment 的改造外, Mictract 在核心部署环节还存在三个难点:

- 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;
- Mictract 将 packageID 等环境变量和共享的配置文件以 Deployment 容器变量的形式传入;
- 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。

伪代码 4.1 智能合约 Dockerfile 伪代码

Input: 智能合约代码
Output: 暴露端口提供服务

```
FROM <base-image:tag>
COPY <src> <dest>
WORKDIR <dest>
RUN <build-command>
EXPOSE <port>
CMD <run-command>
```

伪代码 4.2 智能合约 Deployment 伪代码

Input: 链码镜像地址,命名空间,链码包 packageID
Output: 暴露端口提供服务

```
namespace <namespace>
containers: image <image-uri>
image: imagePullPolicy <IFNotPresent>
env: CHAINCODE-PACKAGEID <packageID>
env: CHAINCODE-ADDRESS <address>
container: port <port>
```

在测试方面, Mictract 提供对 Go 语言编写链码的测试,其借鉴 HF 的 MockStub 类完成账本初始化工作,采用 MockStub 类单元测试以及性能测试,利用 pprof^①完成性能剖析工作;在智能合约监控层面, Mictract 提供智能合约资源消耗以及交易过程监控两个方面的监控。

为了验证本节提出的智能合约微服务化开发运维方法的可行性与部署效率,本节选取了官方链码 asset 进行案例研究,使用智能合约微服务化开发运维方法对 asset 进行托管、运行、监控。可行性是本文首要关心的问题,智能合约微服务化开发运维方法需保证智能合约运行无误。为此本文将官方 asset 链码进行微服务化改造,验证其是否满足预期功能。操作过程如下表 4-3 所示,该表为表明智能合约微服务化开发运维方法确保智能合约正确运行,并不是为了展示 asset 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。

目前,缺乏智能合约微服务化的有效评估指标。由于智能合约微服务化开发运维方法的目的是利用已有的成熟工具快速部署、操作、监控智能合约,以此来提升智能合约开发运维自动化水平,故本文选取部署效率作为分析指标。表 4-4 展示了采用官方脚本方式以及智能合约微服务化开发运维方法搭建、部署不同网络规模所消耗的时间。由于 Peer 节点的数量、组织的数量以及通信效

^①go pprof github 地址

率的影响,同一规格的网络部署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智能合约微服务化开发运维方法平均多使用 225.6 秒。该方法整合进入面向 Hyperledger Fabric 的区块链云化框架后与官方 BaaS 平台 Cello 针对链码部署效率的对比情况详见第 6.4 节。

表 4-3: 操作链码实例

步骤	操作	预期结果	实际
step1	InitLedger()	txid=<TXID>	通过
step2	GetAllAssets()	[{"Key": "asset1", "Record": {"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko", "appraisedValue": 300}, {"Key": "asset2", "Record": {"ID": "asset2", "color": "red", "size": 5, "owner": "Brad", "appraisedValue": 400}}]	通过
step3	ReadAsset("asset1")	{"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko", "appraisedValue": 300} txid=<TXID>	通过
step4	TransferAsset("asset1", "Brad")	Tomoko txid=<TXID>	通过
step5	ReadAsset("asset1")	{"ID": "asset1", "color": "blue", "size": 5, "owner": "Brad", "appraisedValue": 300} txid=<TXID>	通过

表 4-4: 部署效率对比 (单位: 秒 (s))

总 Peer 数	总 Orderer 数	组织数量	脚本方式	智能合约微服务化方法
2	2	2	342	136
4	3	2	440	185
2	3	2	358	146
3	4	3	483	202
1	2	1	306	132

4.6 本章小结

本章首先分析了智能合约开发运维过程中的挑战,并分析了智能合约与微服务架构在设计原则上的相似性,结果表明智能合约与微服务都具备相似的特性。基于这种相似性,本章随后阐述了智能合约微服务化改造过程,包含核心的解耦思想、运行环境以及通信机制,然后提出了智能合约微服务化开发运维流程。最后,本章介绍了方法的原型工具并对所提方法进行了可行性与部署效率的评估。

第五章 面向 Hyperledger Fabric 的区块链云化框架

本章首先阐述面向 Hyperledger Fabric 的区块链云化框架的原理、流程以及输入、处理单元和输出,随后给出了基于该框架的原型工具的需求分析、设计以及相关实现。

5.1 面向 Hyperledger Fabric 的区块链云化框架

区块链的成熟以及 BaaS 的市场规模不断扩大促进了去中心应用的落地,同时对云原生服务市场也促进明显。云的开放、动态可伸缩的能力成为了去中心化应用落地的最佳载体。结合第3.3节所得的策略具体实施方案和第4.4节的智能合约微服务化开发运维流程,本文提出了面向 Hyperledger Fabric 的区块链云化框架,利用 Kubernetes Operator 方法将领域知识集成到 Kubernetes API 编排过程中 [21]。Kubernetes API 是云原生容器管理系统的大脑,它是一个复杂的 API,具有多个层与各种资源 [60]。由于已经介绍了 Mictract 对于智能合约微服务化开发运维流程的支持,故本章侧重介绍区块链网络节点利用 BaaS 计算资源层 Kubernetes 的计算资源、存储资源等资源实现节点云化。该框架能够提供完备的去中心化应用开发能力同时提供隐式管理密码文件、按需配置调度 Kubernetes 资源,解决 HF 生产部署效率、安全性、数据弹性等运营难题,节约开发人员以及运维人员时间成本,使得其更加专注于去中心化应用的逻辑。

Operator 应该管理单一类型的应用程序,遵循 UNIX 原则:只做一件事,并把它做好 [61]。本框架必须解决的首要任务是屏蔽 HF 及 Kubernetes 底层细节,简化 HF 网络各节点的部署,以及有效利用 Kubernetes 代码化、云化的管理基础设施,所以本框架需要分别对 Ca、Orderer、Peer 三种不同的网络节点进行完整生命周期管理。同时,本框架需要承接来自持续交付工具的链码安装部署任务,所以也需要能够对链码进行监控管理。

如图 5-1 所示,本框架的整体工作流程将分为几个步骤。首先,将 HF 领域知

识注入 CRD, 这些属性包含 HF 网络中 Ca、Orderer、Peer 各节点以及链码所具备的功能、性能、监控等可插拔的基础属性。将 CRD 作为本框架的输入, 通过自定义命令完成静态 CRD 相关属性的配置。其次, Manager 是本框架的处理单元, 其被设计成一个黑盒 [62], 用户无需关心 Manager 的内部逻辑设计。Manager 自动生成部署的配置文件, 使用生成的配置文件并结合 Helm 可以轻松的将 HF 网络各节点部署进目标 Kubernetes 集群。部署成功后, Manager 持续监控这些节点及其存储资源的状态。除此网络节点外, Manager 还会持续监听来自持续交付工具所产生的部署链码任务以启动对应的链码容器。Manager 根据持续监测结果调用 Kubernetes 及 HF 相关 API 将各节点及链码调整到期望状态以维持 HF 网络的稳定。最后, HF 网络是本框架的输出, 除 HF 网络各节点基本 Deployment、Service 外, 本框架利用 Istio 基础设施层完成高性能、适应性和可用性 [63][64] 的 TLS 通信负载均衡; 采用原生 Role、Secret、PVC 等方式管理 HF 网络的权限、密码及存储。

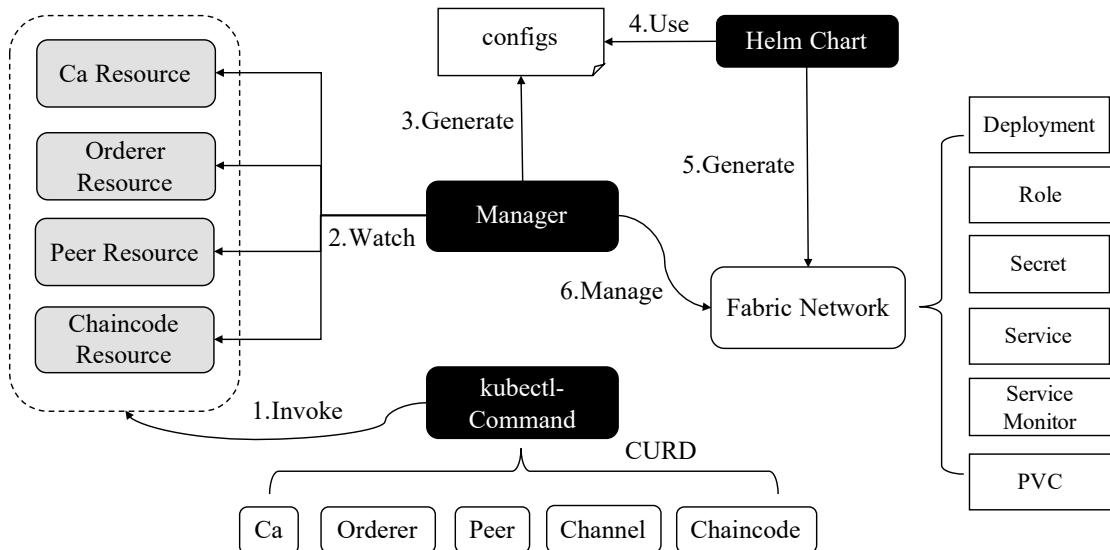


图 5-1: 面向 Hyperledger Fabric 的区块链云化框架

5.1.1 Custom Resource Definition

虽然 HF 和 Kubernetes 形成了理想的匹配, 但许多 HF 网络管理员缺乏必要的 Kubernetes 专业知识, 集群管理员的区块链领域知识也相对匮乏。CRD 则通过扩展 Kubernetes API 将具备领域知识的资源类型注入进 Kubernetes 集群中。Kubernetes 提供了标准资源 ConfigMap, 也可用于使配置数据项供应用程

序使用,但这两种针对不同的情况。ConfigMap 擅长为 Pod 中运行的程序提供配置,应用程序通常希望从 Pod 中读取此类配置,例如文件或环境变量的值,而不是从 Kubernetes API 中读取。CR 由标准的 Kubernetes 客户端创建和访问,遵守 Kubernetes 规范。通过 Controller 可以监控 CR 运行,这些代码可以反过来创建、更新或删除其他集群对象,甚至集群外的任意资源。

表 5-1: CRD 描述

CRD 名称	属性	描述
Ca Resource Definition	CRLSizeLimit	可接受证书撤销列表 (Certificate Revocation List, 简称 CRL) 的大小限制
	TLS	服务器侦听 TLS 端口以及证书等信息
	CA	包含与证书颁发机构相关的信息
	Database	用作数据存储
	CFG	配置身份允许的错误密码尝试次数
	CSR	控制根 CA 证书的创建,如根 CA 证书的过期时间配置
	Registry	部分控制 fabric-ca 服务器执行验证包含用户名和密码的注册和检索标识的属性名称、值的方式
	BCCSP	用于选择要使用的加密库实现
Orderer Resource Definition	Genesis	初始区块相关配置
	BootstrapMethod	指定了获取引导块系统通道的方法
	ChannelParticipation	通道管理对系统链码的依赖
	Secret	包含 Orderer 数字签名以及与 Ca 通信所需的基本信息
Peer Resource Definition	Gossip	确保 Peer 间通过 Gossip 协议来达到所有账本的最终一致性
	LevelDB/CouchDB	HF 提供 LevelDB 与 CouchDB 用以保存 HF 账本信息,用以灵活适应 Peer 不同数据库之间的转换
	CouchDBExporter	采集 CouchDB 的监控数据
	ExternalChaincodeBuilder	提供外部链码构建的能力
	Secret	包含 Peer 的数字签名以及与 Ca 通信所需的基本信息
	MSP	所属的组织信息
Chaincode Resource Definition	PackageID	链码唯一标识符
	Secret	包含用户的数字签名以及与 Ca 通信所需的基本信息
	Env	包含链码所需要的环境变量信息

HF 作为一种去中心化的多方信息对接的网络,具有一套标准化的数据结构与接口。本框架基于 HF 网络各节点自身功能及配置^{①②③}设计三种 HF 静态资源类型以及一种链码动态资源类型作为输入 (S2),包括但不限于如表 5-1 所示的属

^①Ca Config^②Orderer Config^③Peer Config

性,篇幅原因仅展示部分内容。额外的,除上述针对不同网络节点的特殊属性外,本框架为每个节点及链码提供如副本数、镜像、Hosts、日志、ServiceMonitor等基本属性维持 HF 网络的基本运行状态。

5.1.2 Manager

CR 本身仅为特定应用程序提供声明式 API 的数据项的集合,Controller 负责对 CR 的不同事件做出反馈,管理 CR 的完整生命周期。

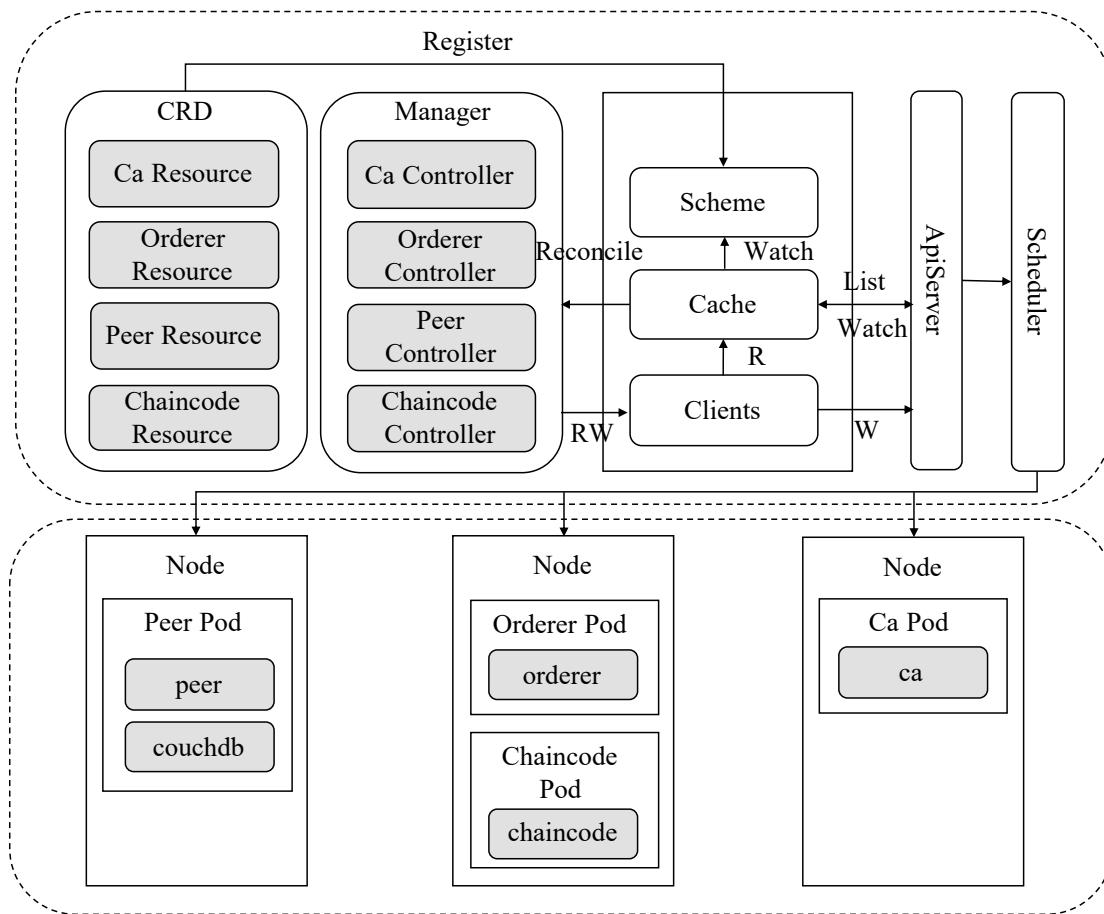


图 5-2: Manager 监听 CR

在 Manager 内部管理了多个 CR,就需要使用多个 Controller 进行协调循环。这有助于关注点分离和代码的可读性。图 5-2 展示的是 Manager 监听 CR 的全过程。首先,需要在 CR 中指定 HF 网络各节点所期望的状态,CRD 会提前注册进入 Scheme,其提供了 APIServer 中 GVK(Group Version Kind) 与 CR 资源类型的映射,通过资源类型 Controller 就能获取 CR 所定义的期望状态;其次,Cache 通过 List-Watch 机制与 APIServer 进行通信用以同步监听 HF 网络各节点

在 Kubernetes 集群中的创建、删除、更新等操作, Cache 可以获取 HF 网络各节点的实际状态; 最后, Controller 循环监听期望状态与实际状态, 若期望状态与实际状态不一致, 则通过调用 Clients 更新、缩放、扩展、备份等操作进行协调一致。

区块链云化框架遵循标准化原则为 HF 网络中的各节点提供了标准化的 Helm chart 模板。在 Helm 中复用 HF 网络节点镜像(S1)并利用 Kubernetes 进行编排和管理底层的物理资源。这可以使用户能够轻松地在集群之间移动和部署云化框架及 HF 网络, 确保区块链系统基础架构的云独立性, 取消对云提供商的强依赖性, 提升本框架的可移植性与通用性。

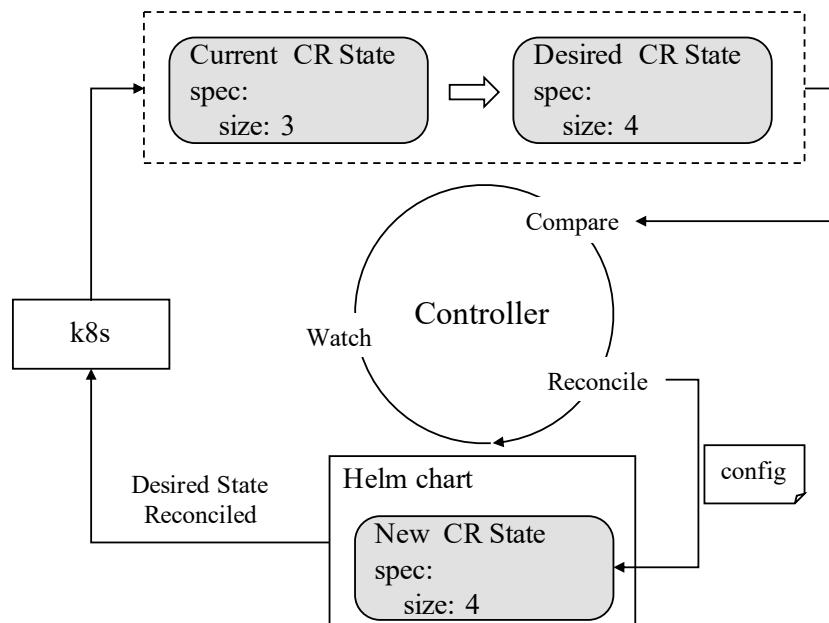


图 5-3: Controller 循环监听

为提升区块链云化框架的生产效率, 本文设计了一套针对 HF Helm 的 Controller(S3), 直接将提前配置好的 Helm 随 CRD 以及 Controller 一起部署进 Kubernetes。Helm 通过调用 Kubernetes 的 APIServer 逐个将 Helm chart 中的 yaml 推送给 Kubernetes。但 Helm 的弊端是缺乏对资源的全生命期监控, 只有 CRD 才能持续监听 Kubernetes 资源对象的变化事件, 进行全生命期监控响应。一旦创建新的 CR, Controller 根据对应的资源对象更新 Helm 的模板参数并重新部署入 Kubernetes 集群。图 5-3 以 spec.size 为例展示了 Controller 更新 Helm 的流程。本框架不仅通过 Helm 简化部署流程, 并且还能实现带全生命周期管理的 Helm 效果。根据这个特性, 每个 CR 中对应的属性一旦经过管理员更改即可反

馈到 Kubernetes 以及 HF 网络中, 这能够在不介入网络的情况下修改、配置 HF 网络。

除第 5.1.1 节所提到的利用 CRD 对 HF 网络配置进行模块化设计外, 本框架在 Helm 中为每个运行中的 HF 网络节点选择 PVC(S4) 作为链外存储, 并为每个 PVC 中预留出一定的额外存储资源。这相较于对所有持久存储的服务使用一个 PVC 而言, 虽然增加额外的存储资源冗余, 但拥有更多的 PVC 能够保障每个网络节点拥有足够的存储空间, 以便在不缺乏存储资源的情况下正确运行节点。拥有更多的 PVC 增加了首次部署难度及过度调配的风险, 但多 PVC 能够灵活针对不同节点运行情况利用 Kubernetes 进行有针对性的存储扩容, 增强框架对于存储的扩展性。尽管多 PVC 在管理方面存在一定的复杂性, 但在选择多 PVC 更加符合最佳实践, 并且效率更高 [61]。具体地, 框架首先部署基础 StorageClass 并针对于 HF 网络中的所有 Ca、Orderer、Peer 节点都会设置专属 PVC 存储单元。这些存储单元会根据 CRD 的定义挂载到具体的网络节点, 网络节点生成的交易数据将会高效的写入可插拔式的持续久化介质里面。

如图 5-4 所示, 当创建 Peer 节点并输入需要多少容量的 CouchDB 时, 框架会为管理员定制生成对应的 Peer CR, 并按照前面所述流程更新 Helm。此时, Kubernetes 会根据 PVC 进行适配寻找合适的 PV 进行存储。PVC 只是针对于存储的声明并不会进行真正的存储, 其服务于 Peer Pod。框架在定义 StorageClass 时会将“allowVolumeExpansion”字段设置为“true”, 这会允许 PVC 的扩容操作。当初始设定的初始存储值不够时, 可以通过修改 CR 中的 PVC 的容量进行动态的灵活扩容。链外存储使得账本数据与操作独立实现, “账本数据生于链却独立于链”, 能够更好地支持数据治理。

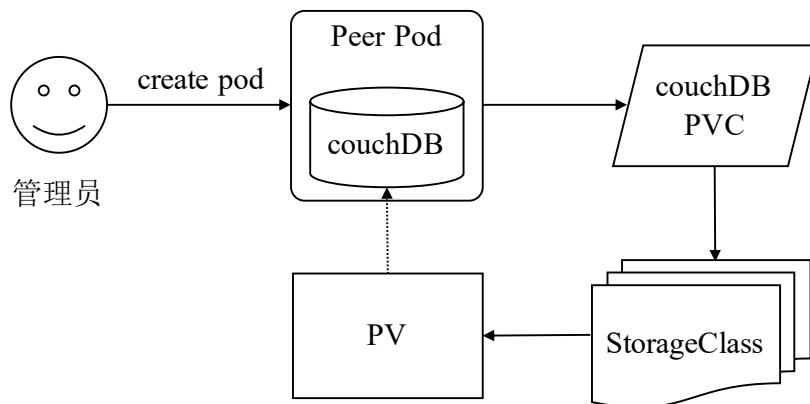


图 5-4: 原型工具存储策略

在安全性方面区块链云化框架复用 Kubernetes 的原生安全保障体系, 主要

涉及到两方面。一方面是 Kubernetes 集群中用户对框架操作的访问控制限制。区块链云化框架会生成很多清单文件向 Kubernetes 集群中部署 HF 网络, 此时框架将所有资源放入同一命名空间下 (S7)。由于 Kubernetes 没有以用户身份进行身份验证, 所以框架采用 RBAC(S5) 将对同一命名空间下资源操作的最小权限映射到框架中的 Manager 及 HF 网络节点。对于自定义生成的资源, 设置了两种类型的角色 (ClusterRole) 分别是 editor 以及 viewer, 如图 5-5-I 展示了 viewer 角色的权限, editor 相较于 viewer 则增加了 create、delete、update、patch 的权限, 使用 ClusterRoleBinding 将角色与用户 (ServiceAccount) 进行捆绑。值得注意的是, 区块链云化框架无需以 root 身份运行, 在确保 HF 网络正常工作的同时, 应尽可能限制访问。上述安全策略只是保护 HF 网络的第一道防线, 另一方面是防止非法用户参与 HF 网络内部交易流程以及数据篡改。在经过 Ca 节点生成用户密码后, 框架避免使用直接向节点镜像中注入环境变量的方式管理密码信息。如图 5-5-II 所示, 本框架采用 Secret(S6) 配合 x509[65] 存储管理导出的敏感密钥数据, 这种方式不但能提高灵活性而且增加了密码的传输、存储、访问安全, 增强隐私保护。

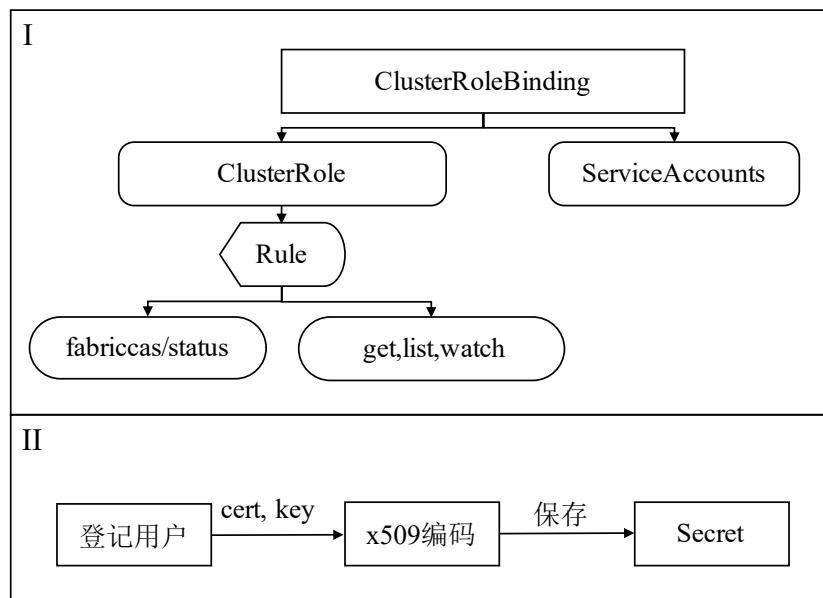


图 5-5: 原型工具安全策略

本框架采用非介入式的云上监控方案 Prometheus 以及 Grafana 进行可视运维 (S9), 在 CRD 中预留 Exporter、ServiceMonitor 等属性, 对应的在 Helm chart 中定制抓取周期等相关配置对 HF 网络中的 Ca、Orderer、Peer、CouchDB、链码等进行可视化监控。同时, Grafana 开源特性能够创建自定义插件, 以及图形化的

方式运维区块链网络。通过 Prometheus 监控体系能够实时监控区块链网络的运行状态,帮助 HF 网络管理人员及时发现并解决问题,提升本框架的可靠性。

HF 2.0 之前链码打包安装于 Peer 节点内部,这是一种强耦合的关系。HF 2.0 之后引入了外部链码的新生命周期,即可以支持链码在 Peer 节点外部构建与启动。如图 5-6 展示了智能合约部署原理。值得注意的是,该流程仅展示流水线的核心部署环节。本框架为链码定制 Dockerfile 模板以及 Kubernetes 部署文件,使单个链码进行容器化部署托管于集群之上。然后,将该过程封装配置作为持续交付流水线中核心的智能合约部署环境。同时,每次智能合约进行升级时,只需修改合约代码即可完成自动升级。通过智能合约微服务化流程可以极大的提高链码部署、升级过程的灵活性。本框相较于 Mictract 在部署中的做了如下优化:(1) 取消 NFS 外部文件服务器的方式,利用 Secret 保存密钥文件;(2) 将环境变量注入链码 CRD 作为输入,而不是直接硬编码 Deployment 配置文件。

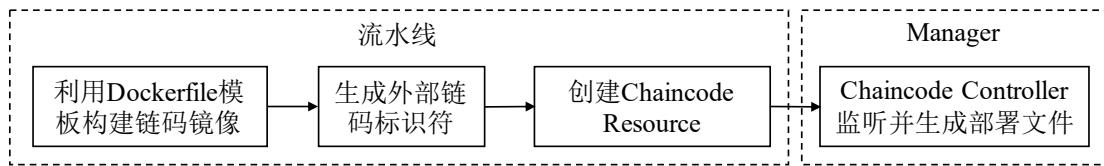


图 5-6: 智能合约微服务部署原理

最后,本框架对外采用动态命令式查询(S10),对内封装创建、更新、删除 HF 网络各节点以及通道、链码相关功能模块的各项操作。通过命令查询的方式,可以大大降低区块链云化框架的使用门槛,无需关心内部复杂的证书、网络等创建逻辑。

5.1.3 Hyperledger Fabric 网络

HF 网络是一个复杂的分布式系统,需要权衡速度、性能等条件对不同网络节点的部署状态进行合理设计。静态节点 Ca、Orderer、Peer 在首次启动网络时就需要部署在 Kubernetes 集群中并以 Pod 形式运行。Pod 是 Kubernetes 中可以创建和部署的最小单位。在 Kubernetes 集群中,Pod 内的容器有两种运行状态:

- 单容器: Pod 当作单容器进行封装,Kubernetes 管理的是 Pod 而非容器;
- 多容器: 当容器间需要紧密协作时可以在同一 Pod 中运行多容器。

Ca 是 HF 网络中的证书授权中心,Orderer 负责交易的排序,这两个节点在配置上需要满足可插拔式设计。但在网络运行时,需要各自在一个 Pod 中运行即

可,同时在一个 Pod 中运行可以有效地利用 Kubernetes 的自动缩放功能进行弹性伸缩。

Peer 是 HF 网络中被使用最多的模块,是 HF 网络的基石,其负责区块链数据的存储以及运行链码。由于 Peer 节点需要频繁地与账本存储单元如 CouchDB 进行交互,所以 Peer 容器应当与 CouchDB 容器存在于同一 Pod 中。在同一个 Pod 中,Peer 容器与 CouchDB 紧密协作,拥有相同的存活周期,更优的,相同 Pod 中的不同容器共享进程、IP 地址和数据卷,可以进行频繁的文件和数据交换。Peer 支持外部链码部署,即链码拥有自己独立的 Pod 运行环境,这能够纳入智能合约微服务化流程进行快速响应与监控。

5.2 原型工具

5.2.1 工具概述

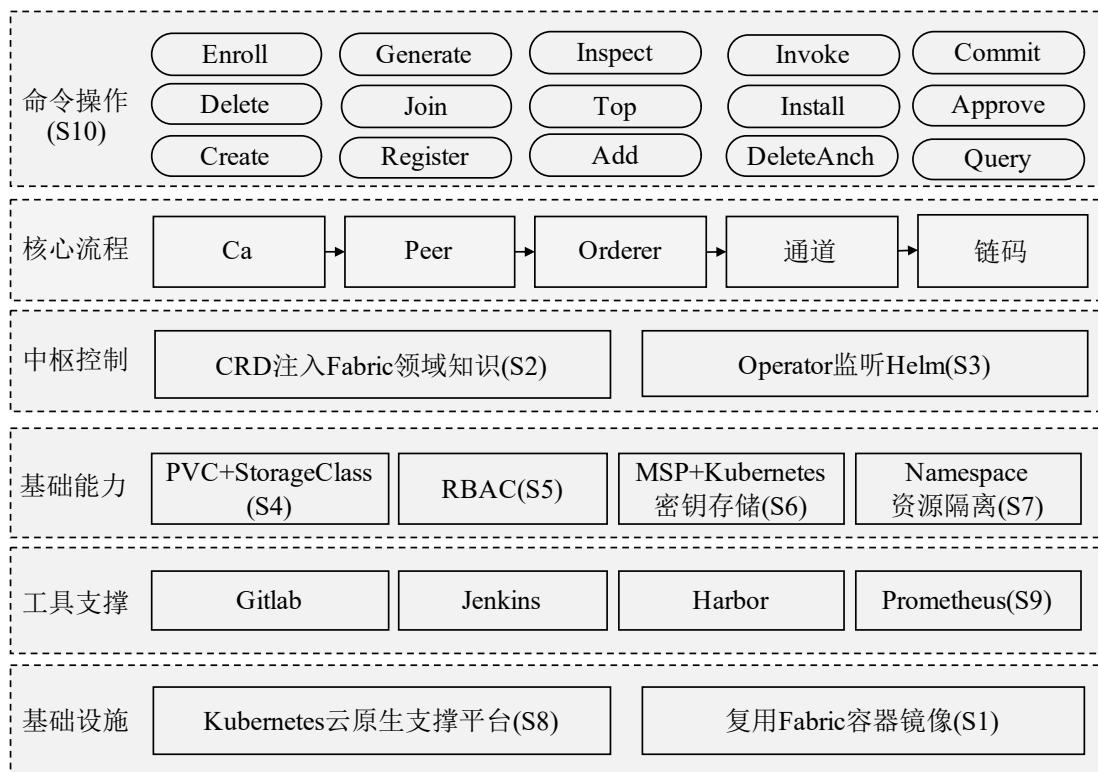


图 5-7: 原型工具总体功能

云原生具有资源按需配置,动态伸缩的特性。当前,BaaS 虽然能够基于云平台构建区块链系统,但仅提供脚本化的方式部署区块链网络及智能合约,仍未深

入云基础设施平台的底层有效利用云的特性管理区块链平台,这导致了 BaaS 平台对云特性的严重浪费。因此,一个支持区块链有效云化的工具十分重要。如图 5-7 所示,本文基于提出区块链云化框架提供配套的面向 Hyperledger Fabric 的区块链云化原型工具。原型工具建立在 HF 核心流程之上,对外以命令的方式动态管理整个 HF 网络,以持续交付的流程部署智能合约,对内声明式配置 HF 网络中的 Ca、Orderer、Peer 实体,利用可移植性、可靠性、易用性、可扩展性以及安全性等策略为 HF 网络赋能。

5.2.2 需求分析

本节重点介绍原型工具的功能性需求,如图 5-8 所示,HF 网络管理人员期望能够通过原型工具对 HF 网络各节点分别进行命令式启停,降低 HF 网络的启动时间成本。

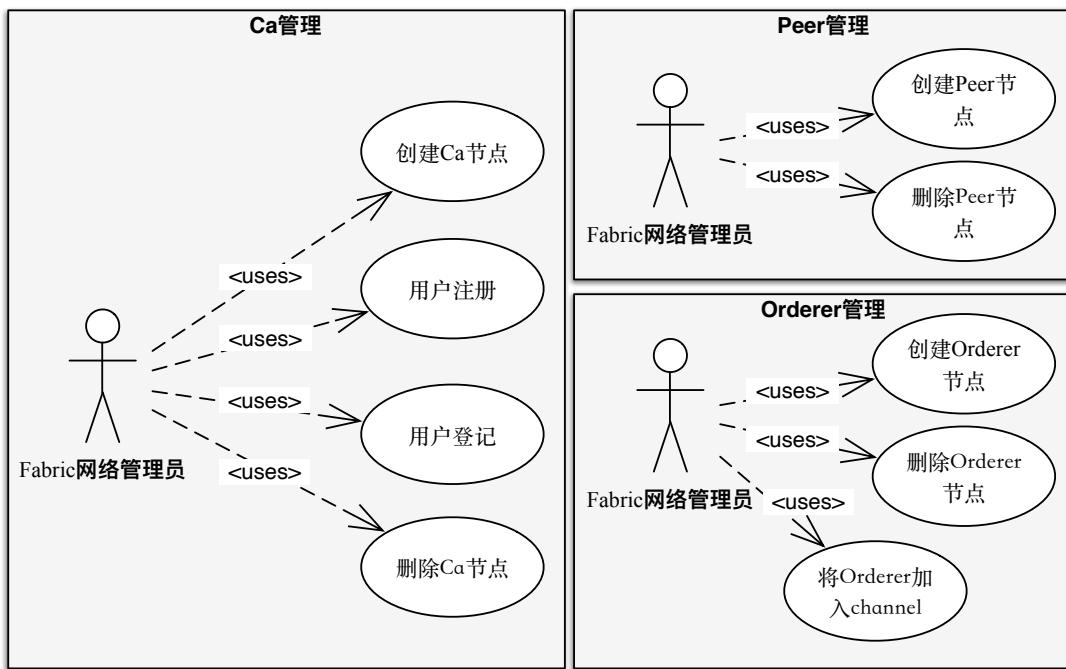


图 5-8: HF 网络管理用例图

表 5-2 展示了 Ca 管理的用例描述,Ca 管理需为 HF 网络管理人员提供灵活方便的启停 Ca 节点的功能,同时提供注册、签发证书的功能。在启动过程中应以命令行参数配置的方式设置 Ca 启动所可选的配置项,如 Database、Host、CLRSIZELIMIT 等。同样,Ca 管理需要覆盖 Ca 节点原有的用户注册、用户登记等基本功能。

表 5-2: Ca 管理用例表

ID	UC-Ca
名称	可视化建模用例
描述	HF 网络管理员可以通过命令的方式创建、删除 Ca 节点，并能够以命令行参数的形式将 Ca 启动参数注入。同时，HF 网络管理员能够无障碍进行用户注册、登记。
触发条件	HF 网络管理员输入对应 Ca 命令
前置条件	Kubernetes 集群环境
后置条件	无
正常流程	(1)HF 网络管理员输入“kubectl hf ca create”并以可选参数的形式输入其他如 Ca 名称、容量等配置项 (2)HF 网络管理员输入“kubectl hf ca register”并以可选参数的形式输入其他如用户名、密码等配置项 (3)HF 网络管理员输入“kubectl hf ca enroll”并输入已经注册过的用户名、密码等信息将注册的用户进行登记并导出证书信息 (4)可选的, HF 网络管理员输入“kubectl hf ca delete”删除 ca 节点
异常流程	无

表 5-3 展示了 Peer 管理的用例描述, Peer 管理需为 HF 网络管理人员提供灵活方便的启停 Peer 节点的功能, 在启动过程中应以内置命令行参数配置的方式设置 Peer 启动所可选的配置项, 如 Peer 对应的 Ca 名称账本存储类型、Gossip 协议、组织信息等。

表 5-3: Peer 管理用例表

ID	UC-Peer
名称	可视化建模用例
描述	HF 网络管理员可以通过命令方式创建、删除 Peer 节点, 并能够以命令行参数的形式将 Peer 启动参数注入。
触发条件	HF 网络管理员输入对应 Peer 命令
前置条件	已经启动对应组织级的 Ca
后置条件	无
正常流程	(1)HF 网络管理员输入“kubectl hf peer create”并以可选参数的形式输入其他如 Peer 名称、所属组织、账本存储类型等配置项 (2)可选的, HF 网络管理员输入“kubectl hf peer delete”删除 Peer 节点
异常流程	HF 网络管理员输入的对应 Ca 名称匹配不上, 报错提示

表 5-4 展示了 Orderer 管理的用例描述, Orderer 管理需为 HF 网络管理人员提供灵活方便的启停 Orderer 节点的功能, 同时能够将 Orderer 加入到通道中, 在启动过程中应以内置命令行参数配置的方式设置 Orderer 启动所可选的配置项,

如 Orderer 对应的 Ca 名称、自己的名称、组织、容量等信息。

表 5-4: Orderer 管理用例表

ID	UC-Orderer
名称	可视化建模用例
描述	HF 网络管理员可以通过命令的方式创建、删除 Orderer 节点，并能够以命令行参数的形式将 Orderer 启动参数注入。
触发条件	HF 网络管理员输入对应 Orderer 命令
前置条件	(1) 已经启动对应组织级的 Ca (2) Orderer 加入通道前确保通道建立
后置条件	无
正常流程	(1) HF 网络管理员输入“kubectl hf orderer create”并以可选参数的形式输入其他如 Orderer 名称、所属组织、所属组织的 Ca 名称等配置项 (2) HF 网络管理员输入“kubectl hf orderer join”并以参数的形式输入名称、命名空间、输入创世区块信息、自己的证书文件 (3) HF 网络管理员输入“kubectl hf orderer delete”以删除 Orderer
异常流程	Orderer 节点证书文件验证不通过，禁止加入

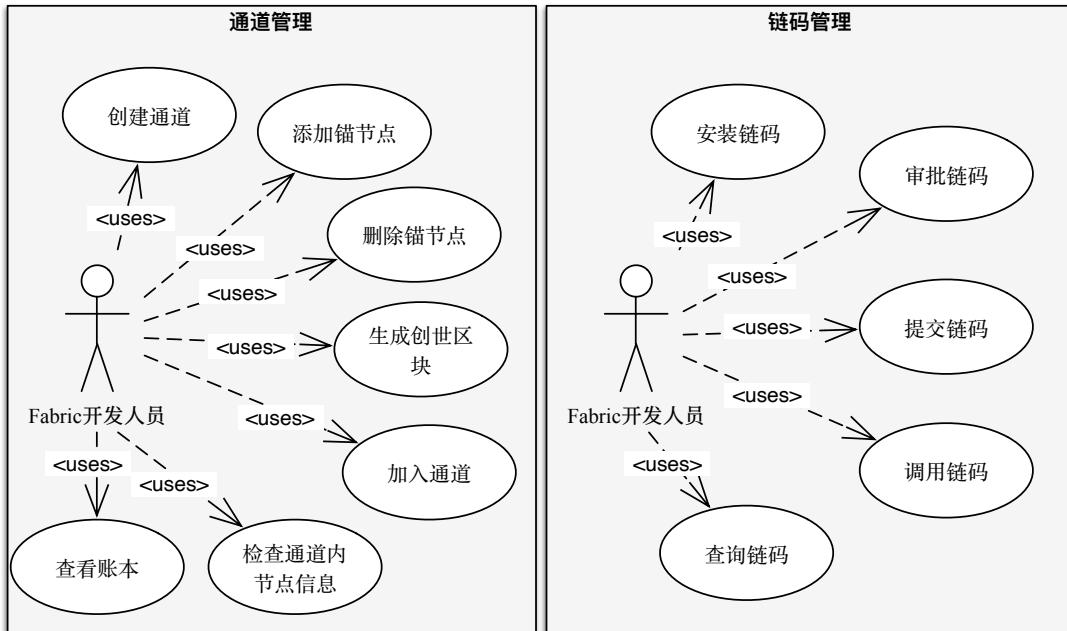


图 5-9: 通道管理用例图

除管理 HF 网络节点外，如图 5-9 所示，本文展示了通道管理以及链码管理的用例图，HF 开发人员期望能够通过原型工具完成管理通道和操作链码的各项操作。表 5-5 展示了通道管理的用例描述，通道管理需为 HF 开发人员屏蔽大量证书文件提供灵活方便地开启通道的功能，应以内置命令行参数配置的方式配置

通道,包含对通道内部锚节点、创世区块、账本的管理。表 5-6展示了链码管理的用例描述,链码管理需为 HF 网络开发人员提供灵活方便的安装、提交链码等功能。

表 5-5: 通道管理用例表

ID	UC-Chan
名称	可视化建模用例
描述	HF 开发人员可以通过命令的方式创建、管理通道
触发条件	HF 开发人员输入对应通道命令
前置条件	已经启动好基础 HF 网络,包含 Ca、Orderer、Peer
后置条件	无
正常流程	(1)HF 开发人员输入“kubectl hf channel generate”并以可选参数的形式输入所包含的组织名称以及输出的创世区块保存文件 (2)HF 开发人员输入“kubectl hf channel join”并以参数的形式输入应当加入该通道的节点名称、组织信息和用户 (3)HF 开发人员输入“kubectl hf channel addanchorpeer”并以参数的形式输入通道的名称用以向通道中加入锚节点 (4)HF 开发人员输入“kubectl hf channel top”并以参数的形式输入通道的名称用以向查询账本的高度
异常流程	无

表 5-6: 链码管理用例表

ID	UC-CC
名称	可视化建模用例
描述	HF 开发人员可以通过命令的方式安装、提交、查询链码;
触发条件	HF 开发人员输入对应链码命令
前置条件	已经部署好通道
后置条件	无
正常流程	(1)HF 开发人员输入“kubectl hf chaincode intall”并以可选参数的形式输入链码地址、链码语言、标签等参数用以安装链码 (2)HF 开发人员输入“kubectl hf chaincode approveformyorg”并以参数的形式传入链码 package-id、链码名称、组织信息、策略等参数用以所在组织审批链码 (3)HF 开发人员输入“kubectl hf chaincode commit”并以参数的形式输入链码名称、策略等信息用以提交链码 (4)HF 开发人员输入“kubectl hf chaincode invoke”并以参数的形式输入链码名称、调用方法等信息用以调用链码 (5)HF 开发人员输入“kubectl hf chaincode query”并以参数的形式输入链码名称、调用方法等信息用以查询链码
异常流程	(1) 因网络原因安装链码时间过长而导致,提示并报错。

5.2.3 设计与实现

面向 Hyperledger Fabric 的区块链云化工具是一个基于 Kubernetes Operator 的应用, 其整合 Helm 用以完成 HF 网络的快速部署和节点的全生命周期管理。为完成命令式查询原型工具采用 Cobra^①完成命令的封装。由于原型工具在管理 Ca、Orderer、Peer 各节点时, 处理逻辑存在重复性, 故本节将以 Ca 节点作为案例介绍详细介绍网络节点针对需求分析的设计与实现。

当 HF 网络管理员需要创建 Ca 并输入对应的命令及参数时, 如图 5-10 所示, 原型工具首先解析输入参数的合法性, 然后将对应的输入参数填充到已经定义好的 FabricCA 模版中, 最后调用 Kubernetes Client 将 Fabric Ca Resource 依据 CRD 的规则部署进入集群中。伪代码 5.1 展示了创建 Ca Resource 的伪代码。

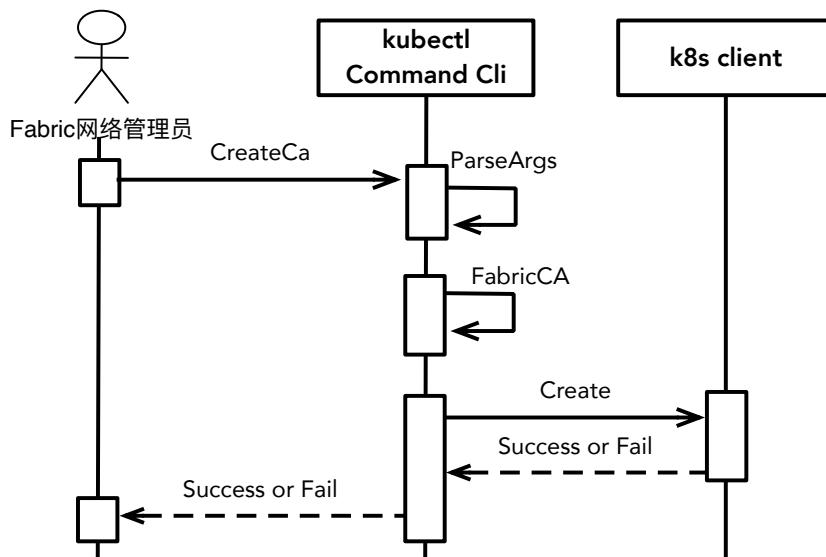


图 5-10: 创建 Ca Resource 时序图

伪代码 5.1 创建 Ca Resource 伪代码

```

Input: "kubectl ca create" with args
Output: Success or Fail
err := ParseArgs()
client, err := GetKubeOperatorClient()
fabricCa := &v1alpha1.FabricCA{Initialization according to parameters}
if args.output then
    out, err := yaml.Marshal(&fabricCa)
else
    err := client.FabricCA(namespaces).Create(fabricCa)
end if
return success

```

^①cobra github 地址

当 Fabric Ca Resource 一旦被部署到集群中就会被处理单元 Manager 中的 Ca Controller 探查到其存活。如图 5-11 所示, Ca Controller 通过 Reconcile 循环探听 Ca Resource。由于资源被删除后再也无法获取到被删除资源的信息, 所以利用 Finalizer 字段进行标识, GetDeletetionTimeStamp() 用于获取 CR 被删除时的时间戳。一旦探听到 Ca Resource 的存在就会先处理 Finalizer 字段。DeletionTimestamp 不为空时, Controller 会轮询该 CR 的更新请求执行处理所有的 Finalizer。随后, Ca Controller 会检查当前是否存在 Ca Helm release, 若不存在则将当前状态 Ca Resource 所定义的 TLS、CFG 等信息生成 Helm chart 并将其部署进入集群中; 若存在则先获取当前 Ca Resource 的状态并对其进行更新之后再生成 Helm chart 部署。Helm chart 中定义了 Ca 启动所需要的全部如 Deployment、Service、Istio、ServiceMonitor 等 Yaml, 通过启动 Helm 就可以一次性将其全部启动部署。伪代码 5.2 展示了 Ca Controller Reconcile 的相关代码。

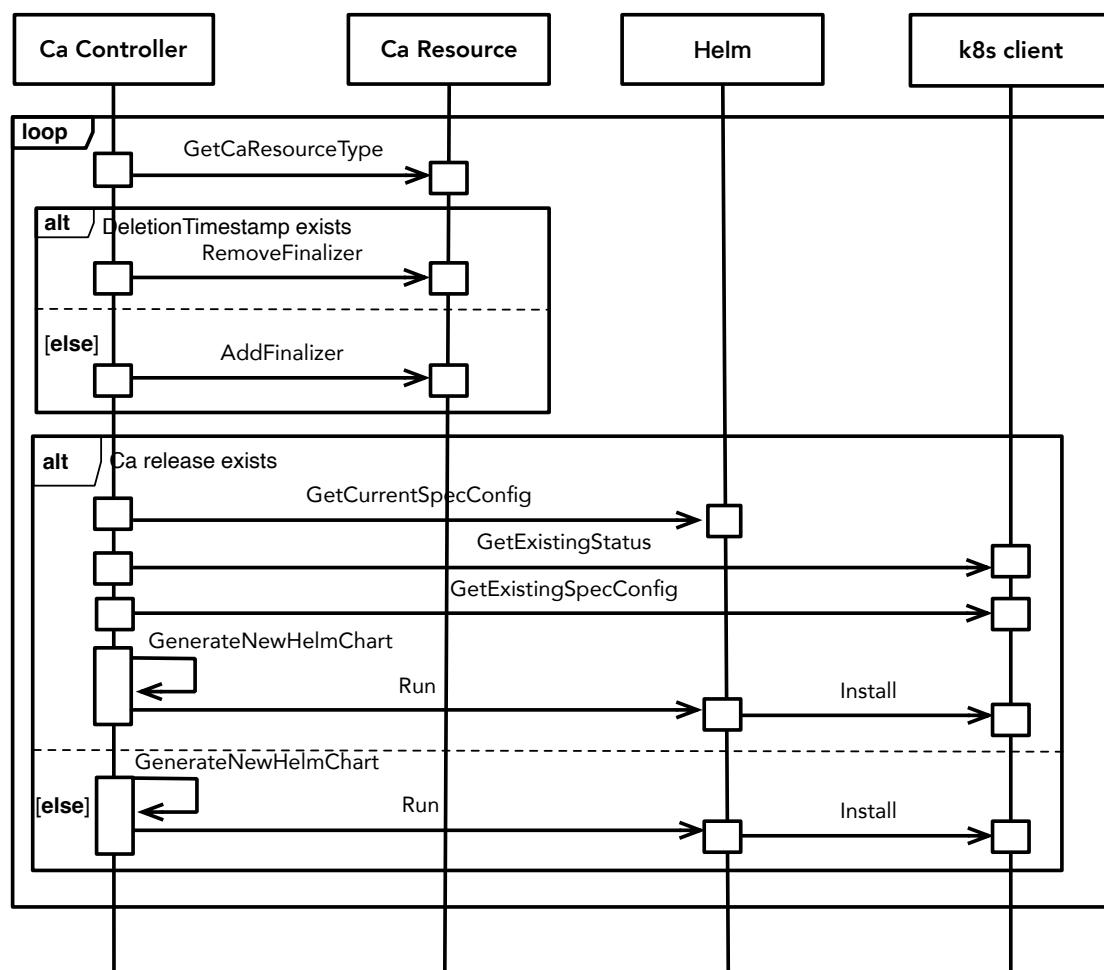


图 5-11: Ca Controller Reconcile 逻辑时序图

伪代码 5.2 Ca Controller Reconcile 伪代码

```

Input: nil
Output: nil
if fabricCa.GetDeletionTimestamp() != nil then
    if fabricCa.GetFinalizers().contains(caFinalizer) then
        RemoveFinalizer(fabricCa, caFinalizer)
    end if
end if
if !fabricCa.GetFinalizers().contains(caFinalizer) then
    AddFinalizer(fabricCa, caFinalizer)
end if
exits := status.Run(caReleaseName)
if exits then
    c := GetCurrentSpecConfig(caReleaseName)
    s := GetExistingStatus(caReleaseName)
    newCa := fabricCa.DeepCopy()
    newCa.Status = s.Status
    release := cmd.Run(caReleaseName, c)
    if !reflect.DeepEqual(newCa.Status, release.Status) then
        err := status().Update(newCa)
    end if
else
    c, err := GetCurrentSpecConfig(caReleaseName)
    release := cmd.Run(caReleaseName, c)
end if

```

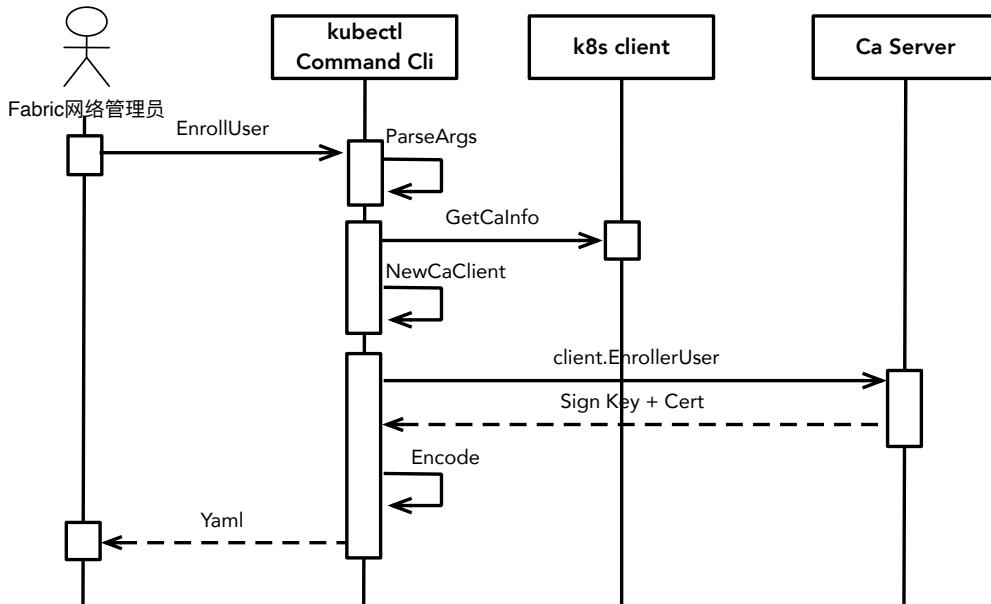


图 5-12: Ca Enroll User 逻辑时序图

成功在 Kubernetes 中启动后, HF 网络管理员可以通过命令形式完成用户注册、用户登记的功能。如图 5-12 所示, HF 网络管理员输入 Enroll 命令以及相关参数, 原型工具会对其进行参数解析。当解析成功后, 原型工具会获取集群中 Ca 对外接口的相关信息, 包含 URL、端口等。随后, 原型工具会生成 Ca Client, 并

将管理员输入的用户信息通过接口传递给 Ca Server。Ca Server 就是 Fabric Ca 在集群中的服务状态所以其具备完整的 Ca 的功能,当 Enroll 完成后会返回 key 以及 cert,原型工具会利用 x509 对其编码并保存到 yaml 中返回给管理员。伪代码 5.3 展示了 Ca Enroll User 的相关代码。

伪代码 5.3 Ca Enroll User 伪代码

Input: “kubectl ca enroll”with args

Output: User key&cert yaml

```

ParseArgs()
client := GetKubeOperatorClient()
url := GetURLForCa()
crt, pk := client.EnrollUser(args.Name, args.Secret, url)
crtPem := EncodeX509Certificate(crt)
pkPem := EncodePrivateKey(pk)
userYaml := yaml.Marshal({
    "key": pkPem,
    "cert": crtPem
})
io.writeFileSync(args.output, userYaml)
return nil

```

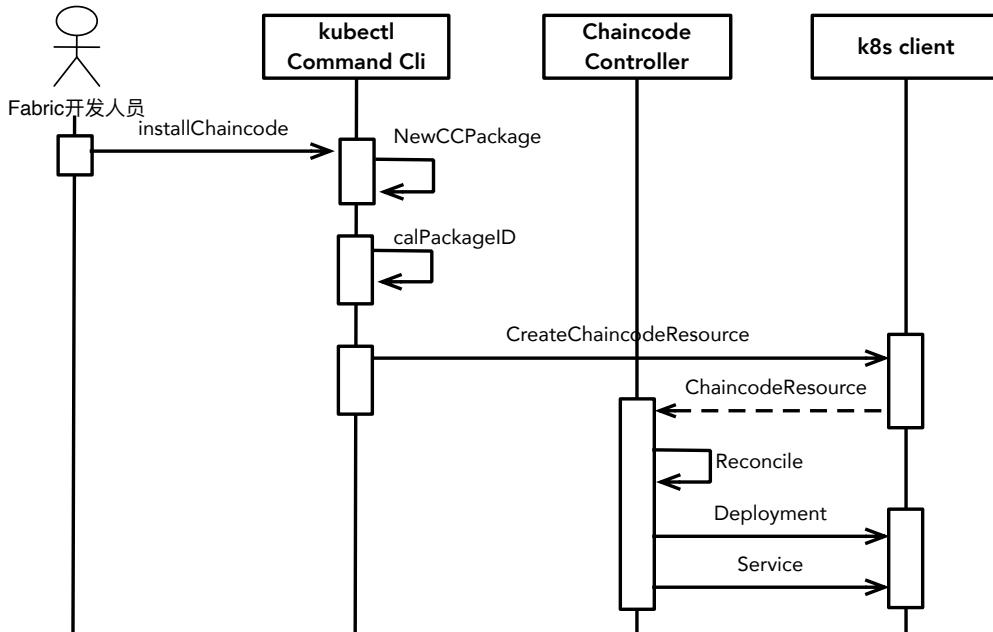


图 5-13: 安装链码逻辑时序图

当成功启动网络后, HF 开发人员能够以持续交付流水线的方式或命令的方式完成链码的部署与升级。如图 5-13 展示了采用命令行方式安装的过程, 安装命令可以配置进行流水线。HF 开发人员输入 install 命令以及链码地址、用户信息等相关参数, 原型工具首先会预先根据输入的参数信息预安装链码包,

并根据预安装返回的 pkgTarGzBytes 生成 packageID 作为链码的唯一标识。然后，原型工具会调用 Kuebrnetes 接口生成 Chaincode Resource。一旦 Chaincode Resource 被部署进入集群就能够被 Manager 中的 Chaincode Resource 监听到，并根据 Chaincode Resource 中所带的 packageID、链码地址、数据卷信息等创建链码的 Deployment 以及 Service。

5.3 本章小节

本章介绍了面向 Hyperledger Fabric 的区块链云化框架及其原型工具。首先，给出了云化框架的工作流程、输入单元 CRD、处理单元 Manager 以及输出单元 HF 网络节点运行状态。然后结合云化框架介绍了其原型工具的需求分析、设计与实现。

第六章 测试与评估

本章将介绍面向 Hyperledger Fabric 的区块链云化框架的测试与评估工作。首先, 本章以典型案例以及 SAAM 方法对原型工具基本能力自证; 其次, 采用五层成熟度模型进行成熟度衡量; 最后, 与官方工具 Cello^①进行对比评估。

6.1 测试环境

如图 6-1 所示, 本文从三个方面对本文提出的区块链云化框架与工具进行综合评估。首先是基本能力自证, 本文以典型案例研究的方式证明原型工具的功能完备性以及智能合约微服务改造的可行性, 结合软件体系结构评估方法 Software Architecture Analysis Method(简称 SAAM) 验证区块链云化节点的质量属性; 其次是成熟度衡量, 本文以定性研究 [66] 的方式结合 Operator 五层成熟度模型对原型工具进行成熟度能力的评估; 最后是工具的对比分析, 本文将原型工具与 Cello 进行定量对比, 评估原型工具的网络节点部署时间以及链码交付效率。

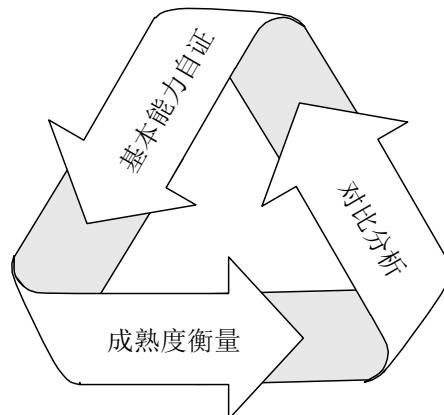


图 6-1: 评估体系

由于在本机环境下搭建 Cello 会出现诸多问题, 如文件挂载权限、编译等, 所以本文准备了两套测试环境。本文首先在本地环境进行功能方面的测试, 本地机器为 MacBook Pro, 并采用 Docker for Desktop 搭建的单节点 Kubernetes 充当集

^①Hyperledger Cello

群环境;其次,在第6.4节中,为顺利运行对比工具Cello,本文利用云主机Ubuntu 18.04搭建Minikube充当集群环境。上述两种环境具体配置如表6-1所示。

表6-1: 配置详情

环境	配置项目	配置详情
本机环境	Docker for Desktop	Version 4.7.0
	Kubernetes	v1.22.5
云主机	Docker	Community 20.10.13
	minikube	v1.25.2
	Kubernetes	v1.23.3

6.2 基本能力自证

6.2.1 案例研究

本小节主要以案例研究的方式对原型工具进行功能性测试。如图6-2所示,本节将以搭建最经典、简单的HF网络案例的方式针对5.2.2节的用例进行完整的功能测试。测试重点针对于Ca、Peer、Orderer启动、创建通道、部署链码、调用的全部过程,验证HF网络启动及链码部署的正确性。

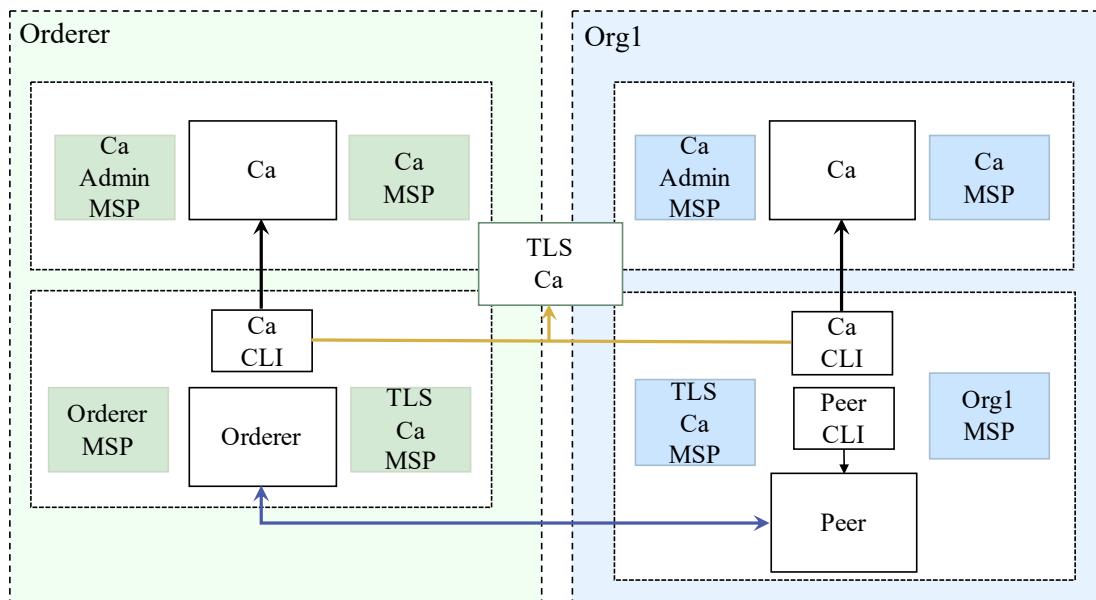


图6-2: 测试网络

前置条件: 原型工具首先将编写好的HF网络各节点的CRD注入Kuber-

netes。注入的 CRD 可以通过原生 kubectl 命令进行管理。如图 6-3 所示为部署之后的结果, 已经将 Ca、Peer、Orderer 和链码的 CRD 部署入集群。

```
[~ $ kc get crd | grep zhangfuli
fabriccas.hlf.zhangfuli.com          2022-04-20T03:17:27Z
fabricchaincodes.hlf.zhangfuli.com    2022-04-20T03:17:27Z
fabricorderernodes.hlf.zhangfuli.com  2022-04-20T03:17:27Z
fabricpeers.hlf.zhangfuli.com         2022-04-20T03:17:27Z
```

图 6-3: CRD 部署结果

步骤一: 如表 6-2 所示, HF 网络管理员首先为 org1 创建了名为“org1-ca”的 Ca 节点; 其次, 在 org1 中创建了名为“org1-peer0”的 Peer 节点, 并利用链外存储 CouchDB 作为账本存储单元。如图 6-4 所示为测试结果。

表 6-2: Org1 测试用例

用例编号	测试编号	用户输入	预期结果	实际
UC-Ca	TC1.1	创建名为 org1-ca 的 Ca 节点	成功启动 Ca	通过
UC-Peer	TC1.2	创建名为 org1-peer0 的 Peer 节点, 其拥有外部 存储 CouchDB, 所属于 Org1MSP	成功启动 Peer 节点	通过

```
[~ $ kubectl get deployment | grep org1
org1-ca                           1/1      1        1      82m
org1-peer0                         1/1      1        1      81m
] [~ $ kubectl get svc | grep org1
org1-ca                           NodePort   10.111.160.49  <none>
                                    7054:31436/TCP,9443:31273/TCP
org1-peer0                         NodePort   10.99.151.250 <none>
                                    7051:31336/TCP,7052:31738/TCP,7053:30941/TCP,9443:32743/TCP 81m
org1-peer0-fs                      ClusterIP  10.106.62.81  <none>
                                    8080/TCP
                                    81m
```

图 6-4: 创建 Org1 测试结果

步骤二: 如表 6-3 所示, HF 网络管理员需要首先为 OrdererMSP 创建名为“ord-ca”的 Ca 节点; 其次, 在 OrdererMSP 中创建了名为“ord-node1”的 Peer 节点。如图 6-5 所示为测试结果, 原型工具可以为 HF 网络管理员创建 Orderer 组织的 Ca 与 Orderer 节点, 其中包含但不限于 Deployment、Service、Pod、Secret 等。

表 6-3: 创建 Orderer 测试用例

用例编号	测试编号	用户输入	预期结果	实际
UC-Ca	TC2.1	创建名为 ord-ca 的 Ca 节点	成功启动 Ca	通过
UC-Orderer	TC2.2	创建名为 ord-node1 的 Orderer 节点, 所属于 OrdererMSP	成功启动 Orderer 节点	通过

```

[~ $ kubectl get deployment | grep ord
ord-ca                               1/1      1          1          122m
ord-node1                            1/1      1          1          121m
[~ $ kubectl get svc | grep ord
ord-ca                               NodePort   10.109.237.105 <none>
                                         7054:30299/TCP,9443:30814/TCP           122m
ord-node1                            NodePort   10.102.86.183 <none>
                                         7050:30395/TCP,7053:31496/TCP,9443:31045/TCP       121m
[~ $ helm list | grep ord
ord-ca                               17          2022-03-18 15:56:54.839698 +08
00 CST deployed          hlf-ca-1.3.0          1.4.3
ord-node1                            15          2022-03-18 15:57:23.871473 +08
00 CST deployed          hlf-ordnode-1.4.0          1.4.3
~ $ ]

```

图 6-5: 创建 Orderer 测试结果

步骤三：如表 6-4 所示, HF 网络管理员需要为 OrdererMSP 以及 Org1 注册并登记若干用户, 并将用户的证书密钥信息导到 Yaml 文件中。表中仅展示在 Orderer 中注册用户, Org1 中步骤相似, 注册名为 peeruser 的用户。测试结果表明, 可以成功将用户及 OrdererMSP 信息导出。

表 6-4: 注册、登记用户测试用例

用例编号	测试编号	用户输入	预期结果	实际
UC-Ca	TC3.1	为 OrdererMSP 注册名为 ordereruser 的用户, 其密码为 ordererpw	成功注册	通过
UC-Ca	TC3.2	输入用户名密码, 为 ordereruser 用户登记	成功登记, 输出证书文件	通过

步骤四：如表 6-5 所示, HF 开发人员需要在新创建的 HF 网络上创建通道, 并初始化该通道内的创世区块。当通道被创建完成之后, 需要将在该通道内进行交易的 Orderer、Peer 加入该通道。最后, HF 开发人员可以指定锚节点并查看通道的高度。如图 6-6 所示为测试结果, 展示了新创建的通道的高度。

表 6-5: 创建通道测试用例

用例编号	测试编号	用户输入	预期结果	实际
UC-Chan	TC4.1	选择 OrdererMSP 为排序组织在 Org1MSP 上创建通道	成功创建通道	通过
UC-Chan	TC4.2	在通道内创建创世节点	成功创建创世节点	通过
UC-Orderer	TC4.2	将 ord-node1 加入通道	加入成功	通过
UC-Peer	TC4.3	将 org1-peer0 加入通道	加入成功	通过
UC-Chan	TC4.4	指定锚节点	指定成功	通过
UC-Chan	TC4.5	查看通道高度	查看成功	通过

URL	MSP ID	HEIGHT
192.168.65.4:31336	Org1MSP	L=1

图 6-6: 查看通道高度测试结果

步骤五：如表 6-6 所示, 为测试通道内所有的参与节点按照链码的合同规则执行, HF 开发人员需要在新创建的通道内安装官方提供的 asset^①链码, 安装完链码之后, 需要对其进行审批、提交。最后, HF 开发人员可以调用链码接口对其进行初始化、查询等一系列操作。如图 6-7 所示仅为调用测试结果, 第 4.5 节已经证明了智能合约微服务化的可行性。

表 6-6: 创建通道测试用例

用例编号	测试编号	用户输入	预期结果	实际
UC-CC	TC5.1	安装链码并指定语言、label	安装成功	通过
UC-CC	TC5.2	查询已经安装的链码	查询成功	通过
UC-CC	TC5.3	审批链码, 并提供链码 ID 以及策略	审批成功	通过
UC-CC	TC5.4	提交链码, 并提供链码 ID 以及策略	指定成功	通过
UC-CC	TC5.5	调用链码, 并提供链码调用函数	调用成功	通过

```
[{"Key": "asset1", "Record": {"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko", "appraisedValue": 300}, {"Key": "asset2", "Record": {"ID": "asset2", "color": "red", "size": 5, "owner": "Brad", "appraisedValue": 400}, {"Key": "asset3", "Record": {"ID": "asset3", "color": "green", "size": 10, "owner": "Jin Soo", "appraisedValue": 500}, {"Key": "asset4", "Record": {"ID": "asset4", "color": "yellow", "size": 10, "owner": "Max", "appraisedValue": 600}, {"Key": "asset5", "Record": {"ID": "asset5", "color": "black", "size": 15, "owner": "Adriana", "appraisedValue": 700}, {"Key": "asset6", "Record": {"ID": "asset6", "color": "white", "size": 15, "owner": "Michel", "appraisedValue": 800}}
```

图 6-7: 查询链码

经过上述步骤 1-5, 最终的网络节点状态如图 6-8 所示, 本文搭建了一个单 Orderer 以及单 Org、单 Peer 的经典 HF 网络, 并分别为 Orderer、以及 Peer 组织搭配 Ca 证书认证。在搭建了基本的网络之后, 分别为 Orderer、Peer 注册并登记用户, 最后在组织 Org 中搭建了一个通道并成功安装调用并链码。由案例分析的结果可知, 本文的区块链云化框架及原型工具能够成功搭建 HF 网络节点并能够正确地部署、运行链码。

^①asset 链码

NAME	READY	STATUS	RESTARTS	AGE
asset-6fc4b8f4c7-px56d	1/1	Running	0	3m37s
ord-ca-574bc7574c-5qbgv	1/1	Running	0	8m30s
ord-node1-5c6c9d9dc8-gbw2s	1/1	Running	0	8m2s
org1-ca-58b7748447-7nkh8	1/1	Running	0	12m
org1-peer0-5b6884c4-1c5nm	2/2	Running	0	9m57s

图 6-8: 网络节点状态

6.2.2 架构评估

在软件工程中, 常见的评估软件体系结构的方法有 Software Architecture Analysis Method(SAAM)、Architecture Trade-off Analysis Method(ATAM), 以上都是基于场景的质量属性评估方法 [67]。SAAM 相较于 ATAM 而言其结构以及评估过程相对简单, 需要准备的工作较少并且 SAAM 适用于对软件最终版本的评估 [68]。本文首先对第 3.2 节所涉及到的设计原则的描述映射到对应的质量属性, 然后采取 SAAM 方法对其进行评估。

在 SAAM 评估方法开始之前, 本文邀请了 4 位熟悉 HF 基本概念及网络搭建流程的软件开发工程师分别两两扮演 HF 网络管理员、HF 开发人员, 1 位熟悉 Kubernetes 操作的集群运维工程师扮演集群管理员, 以上 5 位分别为本次 SAAM 评估方法的风险承担者。本节实施 SAAM 评估方法涉及以下步骤:

1. 场景开发: 所有的风险承担者通过头脑风暴的方式, 提出反应自己需求的场景, 产出结果见步骤 3。

2. 软件架构描述: 本文详细的为各位风险承担者介绍面向 Hyperledger Fabric 的区块链云化框架及其原型工具, 包括框架设计理念与原理、原型工具功能和工具实现细节。

3. 场景分类: 在这个过程中, 如表 6-7 所示, 本文对步骤 1 所产出的场景进行分类并设定优先级。同时, 在分析时根据场景是否需要修改特定的体系结构把场景分为直接场景与间接场景。

表 6-7: 场景分类结果

编号	风险承担者	场景描述	设计原则	质量属性	场景分类	优先级
T1	HF 网络管理员	命令式启停 HF 网络中的任意节点	简单易用	功能需求	直接需求	高
T2	HF 网络管理员	命令创建 HF 网络用户	简单易用	功能需求	直接需求	高
T3	HF 开发人员	命令式创建通道	简单易用	功能需求	直接需求	高
T4	HF 开发人员	持续交付部署链码	简单易用	功能需求	直接需求	高

T5	HF 开发人员	命令式操作链码	简单易用	功能需求	直接需求	高
T6	HF 网络管理员 HF 开发人员	能在 30min 内启动完整 网络	简单易用	易用性	间接需求	高
T7	HF 网络管理员	扩容 HF 节点资源	灵活扩展	可扩展性	间接需求	中
T8	HF 网络管理员	扩容账本存储单元	灵活扩展	可扩展性	间接需求	高
T9	HF 开发人员	保存用户密钥	安全可靠	安全性	间接需求	高
T10	HF 开发人员	操作网络时需要身份 认证	安全可靠	安全性	间接需求	高
T11	k8s 管理员	HF 网络与其他集群程 序进行隔离	安全可靠	安全性	间接需求	高
T12	k8s 管理员	Operator 限定管理 HF 网络节点	安全可靠	安全性	间接需求	高
T13	HF 网络管理员 HF 开发人员	良好异常反馈机制	安全可靠	可靠性	间接需求	高
T14	HF 网络管理员 HF 开发人员	HF 网络节点的全面监 控能力	可视运维	可靠性	间接需求	高
T15	HF 网络管理员 HF 开发人员	Operator 在不同云平台 之间迁移的便捷性	云链结合	可移植性	间接需求	中

4. 场景评估: 这个过程重点针对间接场景, 并对其进行评估。针对直接场景, 第 6.2 节已经对其进行了较为全面的功能性测试。

简单易用, 本文同时邀请了这 5 位风险承担者对本文的原型工具进行易用性测试。为了排除网络的影响, 本文提前下载好原型工具搭建 HF 网络所需要的 Docker 镜像。在介绍了本工具的基本原理以及命令行的基本使用方法后, 5 位风险承担者分别独立自行搭建不同规格的 HF 网络, 最终 5 位风险承担者都能够在 30 分钟内学习并掌握原型工具的使用方法。

```
~ $ kc get pvc
NAME          STATUS    VOLUME                                     CAPACITY   ACCESS MODES  STORAGECLASS  AGE
ord-ca        Bound    pvc-577daf9e-fa1e-4bfc-8e2c-c05b7aaf4071  214748364800m  RWO          standard      4h5m
ord-node1     Bound    pvc-813d842b-73f6-4575-b8ab-6abf714bf481  214748364800m  RWO          standard      4h4m
org1-ca       Bound    pvc-7cf56d16-ae9a-41a7-88cd-babef3aff6d  214748364800m  RWO          standard      3h54m
org1-peer0    Bound    pvc-1187ba09-7690-4de2-b905-844203815673  1Gi          RWO          hostpath     3h53m
org1-peer0--couchdb Bound   pvc-bb8c934e-3e1b-46b4-a7f1-d192c4c78e27  1Gi          RWO          hostpath     3h53m
```

图 6-9: 网络存储状态

灵活扩展, 在 HF 网络节点资源方面, 由于 HF 网络节点的 CR 中配置了 HF 网络节点运行态时所需的资源 (CPU、内存) 大小, 所以当对节点 CR 进行修改时, Controller 会监听到 CR 的变化并更新运行中节点 Deployment 的资源大小。在数据存储的可扩展性方面, 如图 6-9 所示, 原型工具为每个 HF 网络中的工作节点设置了专属存储单元。每个存储单元配置专属的 PVC 管理, HF 网络管理员

可以不仅可以通过修改节点的 CR 进行扩容, 可以直接在 Kubernetes 中对 PVC 进行动态扩容。值得注意的是, 由于存储单元 PVC 扩容依赖于 StorageClass, 当前只有 AWS-EBS、GCE-PD、Azure 磁盘、Azure 文件、Glusterfs、Cinder、Portworx 和 Ceph RBD 数据卷插件才能支撑数据扩容操作。

安全可靠, 原型工具通过可以以文件形式导出 HF 网络登记用户产生的密钥信息, 并可以通过 Secret 进行保存。当这些用户操作 HF 网络时, 原型工具会要求提供对应的密钥文件作为身份认证的方式。同时, 原型工具将所有的 HF 资源放在同一命名空间下, 并通过 RBAC 为该命名空间提供了两种类型的角色。当集群其他用户对已经部署的 HF 网络节点拥有超越权限的操作时, 原型工具会提示并不进行相关操作。在可靠性方面, 本文在使用原型工具搭建 HF 网络过程中有意输入错误的命令行参数, 输入不正确的密钥文件等, 原型工具都能在 1s 内给出参数的异常情况; 同时, 原型工具能够有效利用 Prometheus 监控体系对工具本身以及 HF 网络的所有节点、链码以及 DB 进行监控, 如图 6-10 展示了利用 Prometheus 与 Grafana 对 Ca Resource 进行监控的画面, 图中展示了 Ca CPU 的使用情况。在搭建完 HF 网络之后, 原型工具在两周内仅崩溃 1 次, 且能通过日志以及 Prometheus 的告警设置在 5 分钟内进行恢复。

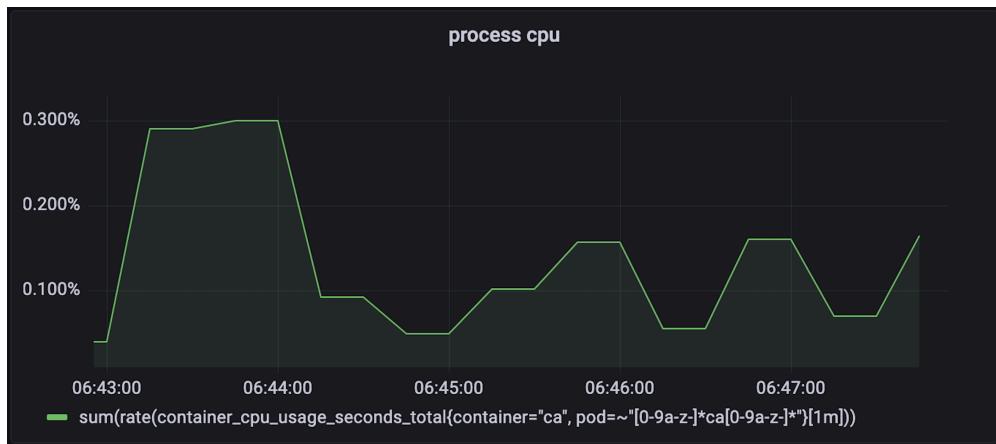


图 6-10: Ca CPU 监控图

云链结合, 重点是在可移植性下的场景。本文通过 Dockerfile 将本工具打包成 Docker 镜像^①, 使用预先打包并经过检验的镜像, 并为该原型工具配备专用的 Helm chart, 其中包含有关原型工具构建的所有必要信息, 这样可以以最少的时间部署。结果表明, 原型工具可以通过 Helm 在支持 Kubernetes 1.18 版本以上的云平台上运行, 原型平台具备良好的可移植性。

^①HFOperator 镜像

5. 总体评估: 本文的原型工具可以有效利用 Kubernetes Operator 云化策略来提升 HF 网络节点的易用性、可扩展性、安全性和可靠性。

6.3 成熟度衡量

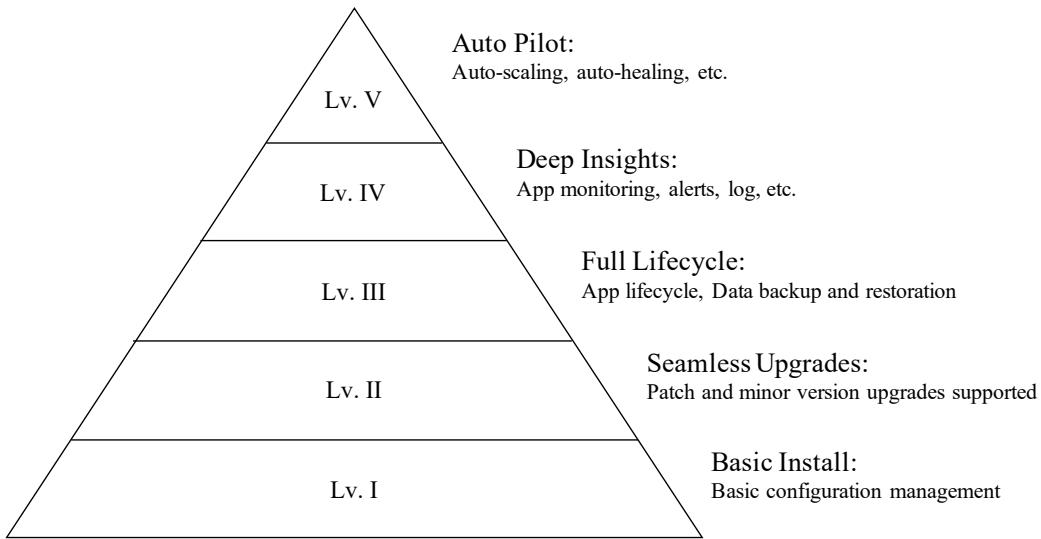


图 6-11: 成熟度模型

如图所示 6-11, Kubernetes Operator 拥有 5 个成熟度级别的定义 [69], 其通过定性的方式分析某个 Operator 应用是否达到了某个级别, 这 5 个成熟度级别定义如下:

- 第一级: Basic Install, 该级是 Operator 成熟度模型中最基本的级别。在该级中, 用户能够使用 CRD 对目标程序进行配置和安装;
- 第二级: Seamless Upgrades, 在该级别上的 Operator 能够不丢失数据的升级所管理的工作负载;
- 第三级: Full Lifecycle, 是否能达到该级别取决于 Operator 是否具备生命周期管理和的数据备份、恢复能力。在该级别上的 Operator 能够备份数据, 并在发生任何数据灾难时从备份数据中恢复数据;
- 第四级: Deep Insights, Operator 提供监控和报警等功能。在该级别, Operator 能够包含所有组件的运行状况指标, 并根据指标配备报警功能;
- 第五级: Auto Pilot, 最终级别拥有许多高级功能, 如自动伸缩。Operator 可以根据收集到的指标来扩展工作负载。

Basic Install

借助于原型工具, 用户能够通过命令行方式一键启停 HF 网络中任意节点, 而且不需要进行复杂的证书管理。一旦原型工具被部署在 **Kubernetes** 网络中, 原型工具就可以对 CR 以及 Helm 进行管理, HF 网络管理人员可以借助命令行参数的方式进行创建、更新 CR 以此来创建特定规格的 HF 网络。一旦 CR 进行更新, 原型工具就会将当前状态调整为与指定状态一致。

Seamless Upgrades

在原型工具中, HF 网络管理员可以通过修改 CR 的内容对包括 HF 网络节点镜像、端口、Host、版本等进行无缝修改。此外, 由于原型工具依赖于链外存储的 CouchDB 以及外部的 Prometheus 监控体系, 这两者的升级并不会对原型工具产生严重的负面影响。

Full Lifecycle

虽然原型工具能够在 Manager 中对结合 Helm 对 HF 网络进行全生命周期管理, 但目前原型工具目前仍缺少数据备份和恢复的能力。要达到这一能力需要在 CR 中指定远程备份的数据存储平台, 并在 CR 中指定数据备份的凭证以及远程备份的链接。同时, 原型工具需要在一定的时间周期内将数据备份到远程的数据存储平台上。

Deep Insights

原型工具利用 Prometheus 监控体系实现监控与报警的功能, 与此同时, 除了利用 Prometheus 抓取 Pod 的基本指标外, CRD 中还设定了 PodMonitor、ServiceMonitor、CouchDBExporter 接口, 可以让 Prometheus 更全面的抓取 HF 网络的监控指标。HF 网络管理员可以通过 Grafana 可视化图表查看整体 HF 网络运行状态, 并且能够根据监控指标创建自定义的告警规则。

Auto Pilot

在 Kubernetes 中存在两种自动伸缩的插件, 即 HPA、VPA。当负载超过一定的阈值时, 就会对其进行伸缩或配置更多的资源。然而在 HF 网络中, 每个 Peer 都有记录全部账本的职责, 并且只需要超过 51% 的 Peer 节点保持一致即可, 所以针对于 Peer 并不需要根据监控进行自我伸缩的能力。在数据存储方面, 随着账本的膨胀, 原型工具可以针对链外存储进行扩容。

综上, 通过定性分析, 本文原型工具利用 Operator 管理 HF 网络能够完美的具备第一级、第二级的能力, 在第三级上能够支持全生命周期管理但是对于数据的备份能力依旧是存在欠缺, 借助 Prometheus 实现了第四级的监控, 对于自动伸缩方面支持资源及数据层面的扩展, 所以本文原型工具基本满足 5 层成熟度模

型的功能。

6.4 对比分析

除上述通过定性分析的手段对原型工具进行评估外,本文选取了 Hyperledger 官方推出的 BaaS 平台 Cello 进行定量的对比分析。本文重点关注的是对 HF 网络节点的云化问题,所以需要在网络部署时间与链码交付时间上对 Cello 以及原型工具进行对比。

本文在云主机环境下拉取 Cello 的 release-0.9.0-h3c 并进行打包构建以及运行。Cello 通过图形化的 Cello Operator 进行主机绑定、组织管理、网络管理以及用户管理。操作 Cello Operator 的就是 HF 网络管理员,管理员首先在主机管理中添加主机,主机就是 Cello 将要部署的目标 Docker 或者 Kubernetes 环境,然后再依次创建组织并启动网络,整个过程全都通过图形化界面的方式完成。

表 6-8: Hyperledger Fabric 版本信息

镜像	Cello 版本	原型工具版本
Fabric-Ca	1.4.2	1.4.9
Fabric-Peer	1.4.2	2.4.1
Fabric-Orderer	1.4.2	2.4.1

由于现在在 Docker 环境中部署 HF 网络仍旧是主流,所以本文利用 Cello 在 Docker 中部署作为测试基准。由于 Cello 的局限性,仅支持在部署 HF 网络的 1.4 版本,而原型工具能更优地支持 HF 网络 2.X 以上版本。如表 6-8 展示了本次工具对比所部署的 HF 网络各节点的版本信息,测试基准为基于单组织单 Peer 的网络部署时间,共识算法选择 Solo, 数据存储选择 LevelDB。

为了避免人为的手工干扰,获得更加准确的网络部署时间。本文提前在 Cello Operator 中创建好 Orderer 以及 org1 组织并为每个组织配置一个对应的节点。当点击提交网络时开始计时,刚开始创建时,网络节点的状态是“故障”,当网络节点的状态从“故障”变成“正常”时停止计时,随后删除该网络。重复 10 次上述操作,且为避免后端镜像遗留干扰,每次操作间隔 3min~5min。在原型工具中,为避免手工输入命令而带来的误差,本文预先编写好创建网络的脚本,脚本中创建 10 次网络,创建完成后删除该网络并在删除后休眠 30s,如此循环 10 次共得到 10 次网络启动时间如表 6-9 所示。

表 6-9: 网络部署时间 (单位: 秒 (s))

工具类型	节点类型	序号										平均
		1	2	3	4	5	6	7	8	9	10	
Cello	整体网络	73	119	115	90	73	89	137	85	103	127	101.1
原型工具	ca(peer)	13	12	13	13	13	13	13	12	12	12	12.6
	peer	20	21	29	16	23	27	20	20	16	22	21.4
	ca(ord)	13	13	13	13	13	13	14	13	15	15	13.5
	orderer	27	36	34	24	33	24	27	25	32	25	28.7
	整体网络	73	82	89	66	82	77	74	70	75	74	76.2

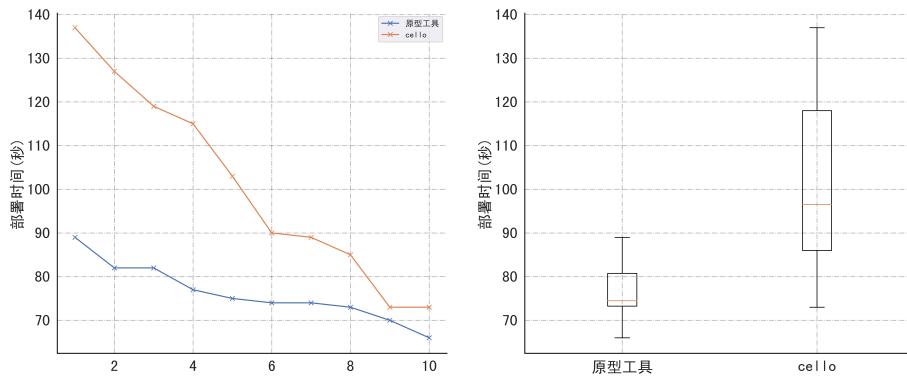


图 6-12: 网络部署时间对比图

如图 6-12展示了原型工具和 Cello 分别部署 HF 网络的对比图。值得注意的是,由于 Cello 采用图形化界面无法配置单个 Ca 的启停,所以采用 Cello 部署的单 Orderer 单组织的 HF 网络仅在单个组织内部启动了一个 Ca 节点,共计 3 个网络节点。而采用原型工具部署的单 Orderer 单组织的 HF 网络分别为 Orderer 以及单组织部署了一个 Ca 节点,共计 4 个网络节点。图 6-12左侧展示了 10 次部署排序后的时间曲线图,图中可以较为直观的看到原型工具比 Cello 部署的时间要少,原型工具部署整体网络的平均时间为 76.2 秒,而 Cello 部署整体网络的平均时间为 101.1 秒;经计算,原型工具的总体标准偏差约为 6.29,Cello 的总体标准偏差约为 21.41。图 6-12右侧分别展示了两者的箱线图,结合标准偏差可知原型工具部署时间相对而言更为稳定,尤其在部署 Ca 节点时,稳定在 13 秒左右。

通过分析源码^①可知,当在 Cello Operator 前端中提交网络之后,Cello 对应的 Docker Agent 会循环解析传入的 HF 网络节点信息及数量的数据结构,解析完成后串行构建 HF 网络中的各类节点。构建过程中,生成 Docker Compose 文件

^①cello create network

并启动对应的网络节点。同样,当 Cello 选择在 Kubernetes 部署时根据 Template 模板在代码中硬编码生成 Yaml 文件。由此可得, Cello 网络部署时间会随着 HF 网络中节点数量的增加而线性递增。本文的原型工具,因需要手动以命令方式部署对应的网络节点,其部署时间也会随着节点数量增加而递增。因此,只对单 Orderer 单 Peer 进行对比测试即可预估出不同网络规模下的网络部署时间。

在链码部署方面,Cello 与原型工具都需要经过创建通道、将账本节点加入至通道内、安装链码等过程。在 Cello 中上述过程均在图形化界面中配置完成,原型工具通过命令行方式构建。这两者在安装链码这个环节均在 1s 内完成,这对于 HF 开发人员而言都是能够接受的,所以这两者仅在安装链码这个环节不需要进行对比。智能合约微服务化的优势在于将智能合约进行解耦,将智能合约单独作为可交付物。同时开发人员可以利用传统成熟的工具链扩充智能合约开发运维领域的工具空白,利用已有成熟工具对智能合约进行持续测试、持续交付与持续监控。

表 6-10: 链码交付时间(单位: 秒(s))

工具类型	序号										平均
	1	2	3	4	5	6	7	8	9	10	
Cello	141	128	185	153	166	105	137	160	186	117	147.7
原型工具	119	89	100	76	104	140	136	125	83	164	113.6

本文对比链码部署过程中所有消耗的时间,即 HF 开发人员编写完链码后到将链码成功部署于 HF 网络节点上所用的时间。如表 6-10 展示了原型工具和 Cello 分别部署 asset 链码所用的时间对比表。Cello 部署链码依次为: 创建通道、加入节点、代码仓库下载链码、压缩链码、计算压缩包 MD5、上传链码、安装链码; 原型工具在持续交付流水线中的流程为: 创建通道、加入节点、代码仓库下载链码、打包链码镜像、安装链码。值得注意的是,本次由于人工操作以及网络波动等情况,数据可能会存在偏差。最终,Cello 的部署链码的平均时间为 147.7 秒,原型工具部署链码的平均时间为 113.6 秒。

综上,本节对 Cello 与原型工具进行了定量的对比分析。在网络节点部署方面,本文原型工具相较于 Cello 在具有更优的节点部署时间,并且每次节点部署时间更加稳定。虽然在链码的部署时间方面原型工具表现的较为良好,但智能合约微服务化的开发运维流程相较于 Cello 固定式的部署流程能够在流水线中纳入更多的环节如智能合约代码质量、安全等,以促进智能合约更全面的发展。

6.5 本章小结

本章介绍了对原型工具的测试与评估。首先介绍了本文涉及到的两个测试环境，并进行了三个维度的评估工作：(1) 基本能力自证，利用典型案例研究的方式进行功能可行性测试，利用 SAAM 架构评估方式进行质量属性验证；(2) 成熟度衡量，利用五层成熟度模型的定性评估；(3) 对比分析，与 Cello 对比的定量分析。

第七章 总结与展望

7.1 总结

区块链网络具有去中心化、不可伪造、不可篡改的特性,其被称为新型生产关系。随着区块链概念的普及,其为供应链、金融等传统领域带来了巨大变革。当前,去中心化应用逐步从概念验证阶段转变为工程化实践阶段。在去中心化应用落地的过程中,构建区块链网络消耗了大量的时间成本。**BaaS** 基于云原生技术体系可以帮助区块链开发人员构建、管理、托管区块链网络,这在一定程度上降低了去中心化应用的落地门槛。

然而,当前区块链的发展依旧存在着诸多挑战。首先是区块链商业化应用工具不完善,当前 **BaaS** 平台由云提供商巨头把控,行业马太效应明显。不同云厂商拥有其独立的 **BaaS** 平台,缺乏顶层规划导致多云网络及区块链数据孤岛。其次,虽然各个云厂商拥有独立的 **BaaS** 平台及其相关生态,但本质上还是将云原生平台作为部署环境,仅提供一种类似于脚本化部署区块链网络的功能,远未挖掘云原生底层特性。最后,缺少促进去中心化应用价值交付的有效手段,智能合约缺少标准的开发流程,工程师很难将传统成熟工具应用到智能合约开发运维领域。

因此,本文针对上述存在的挑战,本文结合 **HF** 以及 **Kubernetes** 提出了一种面向 **Hyperledger Fabric** 的区块链云化框架,综合考虑区块链网络节点和智能合约开发运维两个维度。

在区块链网络节点方面,本文首先对 **Kubernetes Operator** 进行快速评审,快速评审的范围涵盖了计算机与软件工程领域的 4 个权威全文数据库并将 **Scopus** 和谷歌学术作为补充。将得到的 51 篇论文,经过筛选得到 15 篇。对这 15 篇论文进行数据抽取获得了基于 **Kubernetes Operator** 云化的策略集。针对策略集中的策略在区块链背景中进行适配获得了区块链网络节点云化的具体实施方案。在智能合约开发运维方面,本文对比了智能合约与微服务在设计原则上的相似性,并进行了智能合约微服务化改造。本文在智能合约微服务化改造的基础上提出了智能合约微服务化开发运维流程来促进智能合约的价值交付过程。

根据区块链网络节点云化实施方案以及智能合约微服务化改造,本文设计实现了面向 Hyperledger Fabric 的区块链云化框架及其原型平台。云化框架及其原型平台利用 Operator 分别管理 HF 网络中的 Ca、Orderer、Orderer 三类节点和链码。CRD 作为框架的输入,根据官方配置对节点的属性进行可插拔设计; Manager 作为中枢处理单元,通过四个 Controller 对应上述三个节点以及智能合约进行协调循环,时刻保持区块链网络在期望状态。本文结合节点云化实施方案来提升 HF 网络节点的可移植性、可靠性、易用性、可扩展性以及安全性,本文结合智能合约微服务开发运维流程提升链码的部署、升级效率; HF 网络作为输出,包含并不限于维持 HF 网络节点稳定运行的 Deployment、Service 等 Kubernetes 配置。

本文对原型工具进行了包含基础能力自证、成熟度衡量、对比分析三个维度的测试与评估。在基础能力自证方面,本文以典型案例的方式进行功能可行性测试并以 SAAM 架构评估方法对区块链网络节点进行质量属性验证,结果表明原型工具能够利用云原生的特性满足区块链网络的易用性、可扩展性、安全性、可靠性以及可移植性等质量属性要求。本文采用定性分析的方式结合五层成熟度模型对原型工具进行成熟度衡量,采用定量分析的方式对比 HF 官方 BaaS 平台 Cello 在网络部署时间以及链码交付时间上进行了全面对比分析。结果表明,原型工具基本满足五层成熟度模型的功能且在部署时间和链码交付时间上存在较优的表现。

本文提出的面向 Hyperledger Fabric 的区块链云化框架及其原型工具依托于 Kubernetes 基础设施,可以方便地迁移到支持 Kubernetes 的任何云,包括公有云、专有云以及混合云,打破云厂商的垄断格局。本文利用输入本应由领域专家执行的参数与操作,复用 Kubernetes API 的公共功能,如 PVC 存储、资源隔离、内置认证等方式更好地发挥云原生的潜能,提升 HF 网络的易用性、可扩展性、安全性、可靠性以及可移植性,这减轻了 HF 网络管理员的负担。本文能够利用智能合约微服务化开发运维流程帮助 HF 开发人员利用传统成熟工具促进智能合约价值交付流程。

7.2 局限

本文搭建了原型平台并与 Cello 进行对比分析,虽然原型平台在网络部署时间与链码交付效率上优于 Cello,但本文提出的面向 Hyperledger Fabric 的区块链

云化框架及其原型工具依旧还有很多局限性:

- 为轻量级、快速地将已有知识转移到实践中,在区块链云化框架调研过程中采用了快速评审的方式,其相对于系统化文献综述 (Systematic Literature Reviews, 简称 SLRs) 而言缺乏更加深层次的知识掌握。同时 Kubernetes Operator 以及云原生技术体系在工业界应用广泛而本文的快速评审面对的是学术工作,未对灰色文献进行调研。
- 本文的 Manager 中枢控制系统利用 Controller 对 Helm 进行生命周期监控,并通过 CRD 的配置对 Helm value 进行参数传递。这种方式虽然有利于快速启动 HF 网络节点,但增加了 Helm 作为中间传递的过程,未能直接对 HF 网络节点本身的 Deployment 等配置进行关联。同时,由于 Kubernetes 版本的所限,本框架在 Helm chart 中配置的 apiVersion 均需要支持 Kubernetes 1.18 版本及其以上。
- 在框架评估方面,虽然基本满足五层成熟度模型的能力要求,但目前原型工具尚未具备数据备份和恢复的能力,未对当前原型工具及所搭建的区块数据进行远端备份。在可扩展性方面, Kubernetes Operator 云化策略集中包含支持 HPA、VPA 的可扩展处理,由于区块链本身特性,本文未对 Peer 进行 HPA。但区块链云化框架原则上可以支持 Peer 的 VPA,并且针对 Ca 以及 Orderer 节点,区块链云化框架未提供任何可扩展的处理方式。
- 本文虽然与 Cello 进行了验证评估,但仅对 Docker 环境进行对比分析,未能使用 Cello 在 Kubernetes 中进行部署对比。Cello 部署的 Hyperledger Fabric 为 1.4 版本,原型工具部署的版本为 2.X 版本,在版本上有些许不同。
- 本文在测试评估阶段选取了官方链码 asset 进行案例研究,但这并非企业的完整大型应用,缺乏在大规模 HF 网络测试评估。并且,原型工具能够通过命令行的方式进行 HF 网络及其链码的搭建,这虽然对于软件工程师而言门槛较低,但未提供图形化的方式。

7.3 展望

本文提出的面向 Hyperledger Fabric 的区块链云化框架及其原型工具能够有效的利用 Kubernetes Operator 对 HF 网络进行云化,但依旧存在不足,本文仍需在以下方面进一步优化提升。

- 深入调研学术界以及工业界关于 Kubernetes Operator 赋能其他领域提升

质量属性的策略,尤其增加对于灰色文献的调研,对本文的策略集进行补充,最终应用到区块链云化框架及其原型工具中。

- 依托于本文的原型工具,进一步抽象封装提供图形化界面支持;同时,结合可扩展性策略,对 HF 网络的 Peer 增加 VPA 配置,对 Ca、Orderer 节点增加 HPA 以及 VPA 配置,使它们具备弹性伸缩的能力以提供更加稳定的服务;进一步扩充不同账本存储单元,并提供原型工具和区块账本数据远程备份的策略,增强数据备份与恢复机制;有效利用监控指标数据,进一步挖掘数据深层次的价值;智能合约微服务化流程方面纳入更多的工具。
- 进一步增加对框架及原型工具的评估手段,面向企业级大规模区块链业务场景,整理更多专家及 HF 工程师对框架及其原型工具的评价,不断汲取、筛选适合区块链场景的云化策略并对本框架升级改造。

参考文献

- [1] 邵奇峰, 张召, 朱燕超, et al. 企业级区块链技术综述 [J]. 软件学报, 2019, 30(9) : 2571 – 2592.
- [2] ONIK M M H, MIRAZ M H. Performance analytical comparison of blockchain-as-a-service (baas) platforms[C] // International Conference for Emerging Technologies in Computing. 2019 : 3 – 18.
- [3] 刘宏宇, 梁秀波, 吴俊涵. 基于 Kubernetes 的 Fabric 链码管理及高可用技术 [J]. 计算机应用, 2021, 41(4) : 956 – 962.
- [4] GERRITS L, KILIMOU E, KROMES R, et al. A Blockchain cloud architecture deployment for an industrial IoT use case[C/OL] // 2021 IEEE International Conference on Omni-Layer Intelligent Systems, COINS 2021, Barcelona, Spain, August 23-25, 2021. [S.l.] : IEEE, 2021 : 1 – 6.
<https://doi.org/10.1109/COINS51742.2021.9524264>.
- [5] GAI K, GUO J, ZHU L, et al. Blockchain meets cloud computing: A survey[J]. IEEE Communications Surveys & Tutorials, 2020, 22(3) : 2009 – 2030.
- [6] MCCORRY P, SHAHANDASHTI S F, HAO F. A smart contract for boardroom voting with maximum voter privacy[C] // International conference on financial cryptography and data security. 2017 : 357 – 375.
- [7] CHANG S E, CHEN Y. When blockchain meets supply chain: A systematic literature review on current development and potential applications[J]. IEEE Access, 2020, 8 : 62478 – 62494.
- [8] ZHANG Y, WEN J. The IoT electric business model: Using blockchain technology for the internet of things[J]. Peer-to-Peer Networking and Applications, 2017, 10(4) : 983 – 994.

- [9] LEKA E, SELIMI B, LAMANI L. Systematic literature review of blockchain applications: Smart contracts[C] // 2019 International Conference on Information Technologies (InfoTech). 2019 : 1 – 3.
- [10] CHRISTIDIS K, DEVETSIKOTIS M. Blockchains and smart contracts for the internet of things[J]. Ieee Access, 2016, 4 : 2292 – 2303.
- [11] XIE H, ZHANG Z, ZHANG Q, et al. HBRSS: Providing high-secure data communication and manipulation in insecure cloud environments[J/OL]. Comput. Commun., 2021, 174 : 1 – 12.
<https://doi.org/10.1016/j.comcom.2021.03.018>.
- [12] SUN J, WU C, YE J. Blockchain-based Automated Container Cloud Security Enhancement System[C/OL] // IEEE International Conference on Smart Cloud, SmartCloud 2020, Washington, DC, USA, November 6-8, 2020. [S.1.] : IEEE, 2020 : 1 – 6.
<https://doi.org/10.1109/SmartCloud49737.2020.00010>.
- [13] TOSH D, SHETTY S, FOYTIK P, et al. CloudPoS: A Proof-of-Stake Consensus Design for Blockchain Integrated Cloud[C/OL] // 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). 2018 : 302 – 309.
<http://dx.doi.org/10.1109/CLOUD.2018.00045>.
- [14] LIANG X, ZHAO Q, ZHANG Y, et al. EduChain: A highly available education consortium blockchain platform based on Hyperledger Fabric[J]. Concurrency and Computation: Practice and Experience, : e6330.
- [15] WAN Z, CAI M, YANG J, et al. A novel blockchain as a service paradigm[C] // International Conference on Blockchain. 2018 : 267 – 273.
- [16] 才丽. 面向 BaaS 平台的资源调度算法研究与实现 [D]. [S.1.] : 浙江大学, 2018.
- [17] SHI Z, JIANG C, JIANG L, et al. HPKS: High Performance Kubernetes Scheduling for Dynamic Blockchain Workloads in Cloud Computing[C/OL] // 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). 2021 : 456 – 466.
<http://dx.doi.org/10.1109/CLOUD53861.2021.00060>.

- [18] PORRU S, PINNA A, MARCHESI M, et al. Blockchain-oriented software engineering: challenges and new directions[C] // 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). 2017 : 169 – 171.
- [19] TONELLI R, PINNA A, BARALLA G, et al. Ethereum smart contracts as blockchain-oriented microservices[C] // Proceedings of the 19th International Conference on Agile Software Development: Companion. 2018 : 1 – 2.
- [20] SHI Z, ZHOU H, HU Y, et al. Operating Permissioned Blockchain in Clouds: A Performance Study of Hyperledger Sawtooth[C/OL] // 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC). 2019 : 50 – 57.
<http://dx.doi.org/10.1109/ISPDC.2019.00010>.
- [21] HENNING S, WETZEL B, HASSELBRING W. Reproducible Benchmarking of Cloud-Native Applications with the Kubernetes Operator Pattern[J], 2021.
- [22] HUANG W, ZHOU J, ZHANG D. On-the-Fly Fusion of Remotely-Sensed Big Data Using an Elastic Computing Paradigm with a Containerized Spark Engine on Kubernetes[J]. Sensors, 2021, 21(9) : 2971.
- [23] ZHOU N, GEORGIOU Y, POSPIESZNY M, et al. Container orchestration on HPC systems through Kubernetes[J]. Journal of Cloud Computing, 2021, 10(1) : 1 – 14.
- [24] AROUK O, NIKAEIN N. 5G Cloud-Native: Network Management & Automation[C] // NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. 2020 : 1 – 2.
- [25] WIRANATA F A, SHALANNANDA W, MULYAWAN R, et al. Automation of virtualized 5g infrastructure using mosaic 5g operator over kubernetes supporting network slicing[C] // 2020 14th International Conference on Telecommunication Systems, Services, and Applications (TSSA. 2020 : 1 – 5.
- [26] ROUZBEH F, GRAMA A, GRIFFIN P, et al. A Unified Cloud-Native Architecture For Heterogeneous Data Aggregation And Computation[C] // Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics. 2020 : 1 – 1.

- [27] 袁勇, 王飞跃. 区块链技术发展现状与展望 [J]. 自动化学报, 2016, 42(4): 481–494.
- [28] 朱昱锦, 姚建国, 管海兵. 区块链即服务: 下一个云服务前沿 [J]. 软件学报, 2020, 31(1): 1–19.
- [29] ZHANG S, YAN H, CHEN X. Research on Key Technologies of Cloud Computing[J/OL]. Physics Procedia, 2012, 33 : 1791 – 1797.
<https://www.sciencedirect.com/science/article/pii/S1875389212015994>.
- [30] 赵添喜, 王静, 陈天琪. 云原生技术剖析 [J]. 科学与信息化, 2021(1): 1.
- [31] 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.
- [32] BHAGAVAN S, BALASUBRAMANIAN S, ANNEM P R, et al. Achieving Operational Scalability Using Razee Continuous Deployment Model and Kubernetes Operators[J]. arXiv preprint arXiv:2012.10526, 2020.
- [33] 王骏翔, 郭磊. 基于 Kubernetes 和 Docker 技术的企业级容器云平台解决方案 [J]. 上海船舶运输科学研究所学报, 2018, 41(3): 7.
- [34] BEN-KIKI O, EVANS C, INGERSON B. Yaml ain't markup language (yamlTM) version 1.1[J]. Working Draft 2008-05, 2009, 11.
- [35] SPILLNER J. Quality assessment and improvement of helm charts for kubernetes-based cloud applications[J]. arXiv preprint arXiv:1901.00644, 2019.
- [36] CARTAXO B, PINTO G, SOARES S. Rapid Reviews in Software Engineering[G] //Contemporary Empirical Methods in Software Engineering. [S.l.]: Springer, 2020 : 357 – 384.
- [37] LISBOA L B, GARCIA V C, LUCRÉDIO D, et al. A systematic review of domain analysis tools[J]. Information and Software Technology, 2010, 52(1): 1 – 13.
- [38] P.S.P. S, S.S. V, R.P. K, et al. Enhancement of observability using Kubernetes operator[J/OL]. Indonesian Journal of Electrical Engineering and Computer Science, 2022, 25(1): 496 – 503.

- [https://www.scopus.com/inward/record.uri?eid=2-s2.0-85121999433&doi=10.11591%2fijeeecs.v25.i1.pp496-503&partnerID=40&md5=0d91c7bebd3feeac26336b32a85cc4fc.](https://www.scopus.com/inward/record.uri?eid=2-s2.0-85121999433&doi=10.11591%2fijeeecs.v25.i1.pp496-503&partnerID=40&md5=0d91c7bebd3feeac26336b32a85cc4fc)
- [39] KANSO A, PALENCIA E, PATRA K, et al. Designing a Kubernetes Operator for Machine Learning Applications[C] // Proceedings of the Seventh International Workshop on Container Technologies and Container Clouds. 2021 : 7 – 12.
- [40] PINO A, KHODASHENAS P, HESSELBACH X, et al. Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond[C] // 2021 International Conference on Computer Communications and Networks (ICCCN). 2021 : 1 – 9.
- [41] YUE W, CHOONHWA L. A Role-Based Orchestration Approach for Cloud Applications[C] // 2021 International Conference on Information Networking (ICOIN). 2021 : 693 – 698.
- [42] LEE J, KIM Y. A Design of MANO System for Cloud Native Infrastructure[C] // 2021 International Conference on Information and Communication Technology Convergence (ICTC). 2021 : 1336 – 1339.
- [43] KOZIOLEK H, BURGER A, ABDULLA P, et al. Dynamic Updates of Virtual PLCs deployed as Kubernetes Microservices[C] // European Conference on Software Architecture. 2021 : 3 – 19.
- [44] MAHAJAN A, BENSON T A. Suture: Stitching safety onto kubernetes operators[C] // Proceedings of the Student Workshop. 2020 : 19 – 20.
- [45] SHARMA A, YADAV S, GUPTA N, et al. Proposed model for distributed storage automation system using kubernetes operators[G] // Advances in Data Sciences, Security and Applications. [S.l.]: Springer, 2020 : 341 – 351.
- [46] BAUMANN L, BENZ S, MILITANO L, et al. Monitoring resilience in a rook-managed containerized cloud storage system[C] // 2019 European Conference on Networks and Communications (EuCNC). 2019 : 89 – 94.

- [47] PATEL J, TADI G, BASARIR O, et al. Pivotal Greenplum[©] for Kubernetes: Demonstration of Managing Greenplum Database on Kubernetes[C] // Proceedings of the 2019 International Conference on Management of Data. 2019: 1969–1972.
- [48] SHAH J, DUBARIA D. Building modern clouds: using docker, kubernetes & Google cloud platform[C] // 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). 2019: 0184–0189.
- [49] SUKHIJA N, BAUTISTA E. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus[C] // 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCAL-COM/UIC/ATC/CBDCom/IOP/SCI). 2019: 257–262.
- [50] LOPEZ J, RUBIO J E. Access control for cyber-physical systems interconnected to the cloud[J]. Computer Networks, 2018, 134: 46–54.
- [51] SUN Y, FAN L, HONG X. Technology development and application of blockchain: Current status and challenges[J]. Strategic Study of Chinese Academy of Engineering, 2018, 20(2): 27–32.
- [52] YANG X, LI X, WU H, et al. The application model and challenges of blockchain technology in education[J]. Modern distance education research, 2017, 2: 34–45.
- [53] RODLER M, LI W, KARAME G O, et al. Sereum: Protecting existing smart contracts against re-entrancy attacks[J]. arXiv preprint arXiv:1812.05934, 2018.
- [54] TIANSONG L, YU L. Design and implementation of second-hand goods renting system based on Ethereum smart contract[C] // Proceedings of the 2019 4th International Conference on Intelligent Information Processing. 2019: 346–351.
- [55] BUMBLAUSKAS D, MANN A, DUGAN B, et al. A blockchain use case in food distribution: Do you know where your food has been?[J]. International Journal of Information Management, 2020, 52: 102008.

- [56] HAN X, YUAN Y, WANG F-Y. Security problems on blockchain: the state of the art and future trends[J]. *Acta Automatica Sinica*, 2019, 45(1): 206–225.
- [57] LI S, XU Q, HOU P, et al. Exploring the challenges of developing and operating consortium blockchains: a case study[G] // Proceedings of the Evaluation and Assessment in Software Engineering. 2020 : 398–404.
- [58] WEBER I. Blockchain and Services—Exploring the Links[G] // Service Research and Innovation. [S.l.] : Springer, 2018 : 13–21.
- [59] LUU L, CHU D-H, OLICKEL H, et al. Making smart contracts smarter[C] // Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016 : 254–269.
- [60] YILMAZ O. Extending the Kubernetes API[M/OL] // Extending Kubernetes: Elevate Kubernetes with Extension Patterns, Operators, and Plugins. Berkeley, CA : Apress, 2021 : 99–141.
https://doi.org/10.1007/978-1-4842-7095-0_4.
- [61] D'AQUINO A. Design, prototyping and validation of a Kubernetes operator for an IoT platform based on Red Hat OpenShift.[J], 2020.
- [62] YU J, NAGANUMA Y, IDE T. System Operator: A Tool for System Management in Kubernetes Clusters[J]. *CLOUD COMPUTING* 2020, 2020 : 83.
- [63] LI W, LEMIEUX Y, GAO J, et al. Service mesh: Challenges, state of the art, and future research opportunities[C] // 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). 2019 : 122–1225.
- [64] LARSSON L, TÄRNEBERG W, KLEIN C, et al. Impact of etcd deployment on Kubernetes, Istio, and application performance[J]. *Software: Practice and Experience*, 2020, 50(10) : 1986–2007.
- [65] VUKASOVIć M, VESELINOVIć B, STANISAVLJEVIć . A development of a configurable system for handling X509 certificates[C/OL] // 2017 25th Telecommunication Forum (TELFOR). 2017 : 1–4.
<http://dx.doi.org/10.1109/TELFOR.2017.8249485>.

-
- [66] TASHAKKORI A, TEDDLIE C, TEDDLIE C B. Mixed methodology: Combining qualitative and quantitative approaches : Vol 46[M]. [S.l.]: sage, 1998.
 - [67] IONITA M T, HAMMER D K, OBBINK H. Scenario-based software architecture evaluation methods: An overview[C] // Workshop on methods and techniques for software architecture review and assessment at the international conference on software engineering. 2002 : 1 – 12.
 - [68] 胡红雷, 毋国庆, 梁正平, et al. 软件体系结构评估方法的研究 [J]. 计算机应用研究, 2004, 21(6) : 11 – 14.
 - [69] DUAN R, ZHANG F, KHAN S U. A Case Study on Five Maturity Levels of A Kubernetes Operator[C] // 2021 IEEE Cloud Summit (Cloud Summit). 2021 : 1 – 6.