产业动态 | 技术热点 | 行业趋势 | 实战分享 | 人物专访

云原生

Cloud Native

2022.12

第 5 期

供创原会会员
内部交流使用

创原会 CLOUD NATIVE ELITE CLUB

目 录

CONTENTS

1 产业动态 01-12

2 技术热点 13-42

出版方
创原会

出版时间
2022年12月

投稿邮箱
member@cnec.io

3
行业趋势
43-70

4
实战分享
71-94

5
人物专访
95-100

云原生 Cloud Native

产业动态

标准与开源动态 -- 03-05

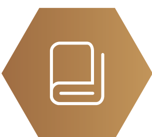
云原生时代，如何培养适合企业的工程师人才？ ------------------------ 06-08

云原生时代的中间件能力度量准则 -- 09-12

标准与开源动态

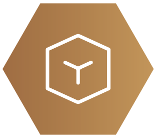
1、我国主导的分布式云全局管理国际标准正式发布

国际电信联盟（ITU）于2022年9月正式发布分布式云全局管理框架标准：ITU-T Y.3538 Cloud computing - Global management framework of distributed cloud。该项国际标准由中国信息通信研究院（以下简称“中国信通院”）、中国联通、中国移动等单位牵头制定，旨在从通用架构和功能要求等方面定义和规范分布式云全局管理能力。标准在资源、数据、平台服务、应用服务、运维管理、安全方面规范管理要求。资源管理包括计算（虚拟机、容器等）、存储、网络等分布式基础资源管理能力；数据管理包括数据处理、传输、加密、存储等分布式数据管理能力；平台服务管理包括平台服务、接口、功能等平台层管理能力；应用管理包括面向用户的业务应用等应用层管理能力；运维管理包括资源调度、部署、监控运维等分布式运维管理能力；安全管理包括分布式云安全风险管理能力。ITU-T Y.3538国际标准的发布是分布式云领域重要标准化成果，通过国际和国内标准协同联动，进一步增强我国在分布式云标准化上的主导权。



2、中国信通院发布《中国算力发展指数白皮书（2022 年）》

2022年11月5日，中国信息通信研究院发布了《中国算力发展指数白皮书（2022年）》。白皮书系统研究了全球算力发展情况，全面剖析了我国算力总体发展态势，并对中国算力发展情况进行客观评估，并结合当前我国算力发展现状和评估结果提出了我国算力发展建议。白皮书观点显示，全球算力进入新一轮快速发展期，我国也已经开启算力赋能数字经济新篇章，主要呈现以下五个特点：算力规模持续扩大，智能算力成为增长动力；供给水平大幅提升，先进计算创新成果涌现；发展环境持续优化，网络体系打通数据动脉；赋能效应不断激发，智改数转进入纵深发展；算力助推经济增长，数字经济实现量质齐升。



3、《人工智能模型风险管理框架》正式发布

2022年11月14日，《人工智能模型风险管理框架》正式发布，该框架是由中国银行业协会指导、中国信息通信研究院支持，中国工商银行、中国农业银行、交通银行、招商银行、广发银行、网商银行、浦发银行、兴业银行、渤海银行、中央国债登记结算有限公司共同起草，以探索人工智能模型风险的使用原则与方法论，该框架制定和发布将有助于银行业相关机构有效开展风险管理能力建设、快速灵活 应对模型风险、更好地将人工智能技术应用于数字化转型。中国工商银行管理信息部资深专家陈道斌表示，新技术应用往往伴随着新风险，如何在兼顾质量、效率、成本的原则下，快速搭建起有效的模型全生命周期管理和风险防控体系，成为金融机构面临的重要课题，《管理框架》能够满足商业银行开展模型管理的需要，为未来实践提供了清晰的指引。希望银行业协会继续组织各成员单位，持续深化人工智能模型应用和风险管理的标准建设，工行也将积极做好落实和实践。

4、Forrester 发布 2022 十大新兴技术分析报告

Forrester认为，目前目前有三分之二的企业都在增加新兴技术支出，而这可能导致炒作、加剧对错失良机的担心，每位客户都应关注十种新兴技术。本次报告按照大多数客户可期望获得的显著正投资回报率（ROI）对其进行梳理，报告显示，云原生计算、自然语言处理（NLP）、边缘智能、可解释 AI 和隐私保护技术这六项技术即刻或顶多在接下来四年内就可带来显著的 ROI，而扩展现实、图灵机器人（TuringBots，也即 AI 编程）、Web3 和零信任边缘这四项前沿技术则至少需要五年时间才能为大多数客户带来正投资回报。

5、Gartner 发布 2023 年十大战略技术趋势

日前，Gartner发布企业机构在2023年需要探索的十大战略技术趋势，Gartner 2023年战略技术趋势围绕优化、扩展和开拓这三大主题，主要包括：可持续性、元宇宙、超级应用、自适应AI、数字免疫系统、应用可观测性、AI信任、风险和安全管理、行业云平台、平台工程、无线价值实现。这些技术能够帮助企业机构优化韧性、运营或可信度、扩展垂直解决方案和产品交付并利用新的互动形式、更加快速的响应或机会进行开拓。"

Gartner杰出研究副总裁David Groombridge表示："但在2023年，仅仅提供技术还不够。这些主题受到环境、社会和治理（ESG）期望与法规的影响，而这会转换成使用可持续技术的共同责任。为了我们的子孙后代，企业机构每进行一项技术投资，就需要抵消它所产生的环境影响，并且需要使用可持续技术来实现'默认可持续性'这一目标。"

6、Istio 正式成为 CNCF 孵化项目

9月28日，云原生计算基金会 CNCF 宣布，Istio 正式成为 CNCF 的孵化项目[1] 。Istio 指导委员会于今年4月25日向CNCF提交申请，由CNCF TOC（技术监督委员会，Technical Oversight Committee ）最终投票通过。作为一个开源的服务网格，Istio透明地提供一种统一和有效的方式来保护、连接和监控云原生应用中的服务。它提供零信任网络、策略执行、流量管理、负载均衡和监控，而不需要重写应用程序。

7、首次！产学研机构联合发布云原生边缘计算公开课

作为业界首个云原生边缘计算开源项目，KubeEdge自开源以来受到了产业界和学术界广泛的关注和支持，吸引了全球来自30+国家的70+贡献组织及16万+开发者，项目已广泛应用于交通、工业制造、智能CDN、智慧园区、金融、物流、航天、汽车、电力、煤矿、油气等行业，为用户提供一体化端边云协同解决方案，KubeEdge社区联合联合产、学、研领域的重量级导师，共同开发并发布了首个云原生边缘计算系列公开课。

8、第一届云原生边缘计算学术研讨会成功举办

2022年11月16-17日，由KubeEdge社区发起，并联合CNCF、华为云等单位共同举办的第一届云原生边缘计算学术研讨会成功召开。本次研讨会聚合学界领军专家和技术爱好 者，汇集边缘计算学术研究及实践案例，旨在进一步帮助更多云原生边缘计算开发者、用户了解边缘计算最新技术突破与未来挑战，洞察边缘计算前沿技术趋势，开拓行业发展新机遇。

云原生时代，如何培养适合企业的工程师人才？

前言

当今时代下，工程师人才培养还有哪些可能 / 新突破？工程师如何理解自己工作的意义？人才培养的新未来是什么？北京邮电大学计算机学院副院长王尚广、普华永道合伙人万彬、福佑卡车 CTO 陈冠岭、华为云 PaaS 技术创新 Lab 主任 王千祥齐聚创原会第三期读书会，一起为您揭晓答案。

金秋 9 月，由创原会与人民邮电出版社共同筹划举办的创原会第三期读书会活动在北京举办，活动以 “人才培养　点亮科技创新” 主题，特邀人民邮电出版社学术中心总经理王威担任主持，携手来自产、学、研、投领域的创原会会员一起探讨如何加速企业人才培养，助力企业科技创新。

王尚广教授

北京邮电大学
计算机系副院长

梦想：云原生工程师素质教育之路

北京邮电大学计算机系副院长王尚广教授认为，人才培养之路，是挖掘每个人身上的潜能，教育的实质是指向成长的 ，构建人才培养梯队去平衡注重学生的基础教育与素质教育是值得思考的问题。

在日常教学中，王教授注重从动脑、动心、动脸、动身四个层面来培养学生的综合素质。他认为："学生选择了云原生技术或者某个领域，研究过程也许会有不顺，但只要坚持就有一定的收获，成功的人往往是能坚持到最后的人，工程师培养的关键在于，不仅要培养解决问题的能力，更要培养发现问题的能力。"

王教授最后总结到："云原生技术会在中国有一个很大的爆发趋势，对整个行业是一个新的推进，高校更多的任务在提升原始的创新能力以及人文素养的培养上，聚集于创新人才培养、高精尖缺人才培养、产学沿用人才培养，现在的人才培养，将决定20年后的工程师素质。"

万彬

普华永道合伙人

开端：云原生产业布局洞察及人才牵引

普华永道合伙人万彬在分享中表示，云原生作为云计算领域新变量，加快了企业的云化进程，但缺乏资深、专业的人才是行业在加速上云、全面布局云原生过程中，所面临的普遍难题。全球43%的企业执行管理层希望在2022年招聘具有云解决方案能力的新员工，而深度理解云原生，并能较深入应用这一技术的开发者占比却只有7%。

对于云原生时代的人才战略，万彬补充道，组织需要调整人力资源战略，匹配组织转型，通过"顶端"人才吸纳，互联网企业经营性裁员，"Z时代"人才聚集个人品质规划来完成，这里涉及到两条战略，内部转型培养，关注能力转型 ，鼓励内部人员数字化能力提升，对创新活动提供及时正向激励，外部招聘，吸引云原生专家，保护人才梯度持续平稳发展。

尤其是在疫情之下，用好数字化工具可以不用去在乎人才在哪里，更合理配置人才成本，企业通过持续提升、引入新的技术、文化培育和身心赋能来打造可持续的人才梯队，通过新的技术，新的整体文化的培养， 并且通过这个专业化的领域把这个人才的资源池建立起来。

陈冠岭

福佑卡车CTO

价值：学术投身企业转型职业规划分享

福佑卡车CTO的陈冠岭先生曾在美国马萨诸州立大学从事教育工作12年，对于东西方教育、制度与思想都有着自己的见解。他将这些经验与见解，通过九本书和自身的转型历程进行了详细分享。

陈冠岭通过分享《转行》、《凤凰项目》、《好战略坏战略》三本书来解读工作的意义。人们通过转行进入新的环境、体验新的经历，来拓展生命的宽度，对于其自身而言，则是完成了从技术科

研向商业创新的转变，面对转变中的挑战与不确定性，他提倡先行动 ，再计划，只要大的方向正确就放胆去做，在做的过程中不断优化、调整。

对于如何组建和管理团队、实现战略目标、加速商业创新，陈冠岭在《创新的艺术》、《跨越鸿沟》、《亚马逊逆向工作法》的分享中给出了答案，“企业只有二个基本功能，商业创新和市场营销，而创新始于观察，在头脑风暴产生想法，去实际生产中的验证这些想法。”对于理解管理的关键，陈冠岭建议大家读《哈佛决策课》、《执行》、《领导力》这三本书，可以帮助大家扭转技术和工程的思维，站在用户和客户的角度来思考，进而更好地与公司战略对齐。

最后，陈冠岭总结说：“自己的能力是什么，自己感兴趣的是什么，外部的机会又是什么，三者进行匹配，就是一个人将来要做的事情，从真实的自我走向可能的自我，不给自己的人生设限，可以去追寻可能的自我。”

王千祥
华为云Paas技术
创新Lab主任

趋势：云计算时代的软件人才培养

关于云计算时代的软件人才培养，华为云PaaS技术创新Lab主任王千祥认为，中国经济数字化转型，急剧拥有专业数字技能的人才，“以职业需求为导向、以实践能力培养为重点“是未来数字化人才培养的方向，人才培养体系的构建则是一项系统工程，需要各方共同支持和参与。
华为云Paas技术创新Lab主任王千祥现场分享照片

王千祥表示，联合高校推动开源与计算机教育相结合，培养和发掘人才，构建人才培养体系，拉通高校、教育部、企业、出版社输出配套的技术培训及认证教材。华为云人才培养方式，通过对云计算领域的核心技术的全覆盖，并基于真实的企业案例，构建一站式实训平台，实现校企合作、产教融合，构建成熟的教育生态，助力企业云化转型。华为云软件人才培养方案将软件教学与工程实践无缝结合，通过认证体系，提供清晰的开发者成长路径；通过华为云KooLabs体验真实的云服务环境，提升云上技术能力，通过面授理论培训+上机演示实操，感受贴近真实业务的场景实战。

王千祥补充到，华为云通过与高校人才生态合作，实现产教融合协同育人，提升高校教学质量，并联合用人企业、产业园区共同构建完善的产业人才生态，减低产业人才培养成本，推进产业和城市良性发展。

在最后的讨论环节，来自人民邮电出版社、高等教育出版社、美团、软通动力、环信科技等单位的技术管理者也深度分享并讨论了人才市场的宏观环境、人才洞察、人才培养实践经验。

创原会•读书会以阅读洞见智慧，鼓励大家通过书籍认识更广袤的世界，看见更深邃的文明，激发更强烈的探索精神，窥探万物本质，触摸无限未来。

云原生时代的中间件能力度量准则

周丹颖 中国信通院云大所云计算部 研究员

随着高速运转高频竞争时代的来临，新场景新需求不断涌现，业务类型持续丰富，中间件作为系统软件与应用软件之间的桥梁，在业务构建过程中起着至关重要的作用。从功能上看，中间件是业务通用需求的抽象，可以避免重复造轮子；从方式上看，中间件通过提供标准的接口、协议，屏蔽具体实现细节；从效用上看，中间件能够提高应用可移植性，降低“连接”成本。

在云原生化转型的大趋势下，中间件自身需要顺应云原生环境的特点对产品进行迭代优化，同时云原生平台也需要提供完善的中间件纳管能力，使中间件与平台充分融合。

政策红利持续释放，中间件发展势头良好

中间件、操作系统、数据库是基础软件的三驾马车，2006年2月国务院发布的《国家中长期科学和技术发展规划纲要(2006-2020年)》提出要在未来15年加快发展16个重大专项，第一项就是信息领域的“核高基”专项，即核心电子器件、高端通用芯片及基础软件产品，这代表着中间件真正被放到战略地位。

2017 年 1 月

工信部和发改委联合发布《信息产业发展指南》，将基础软件列为“十三五”时期我国信息产业发展的重点领域。

2020 年 7 月

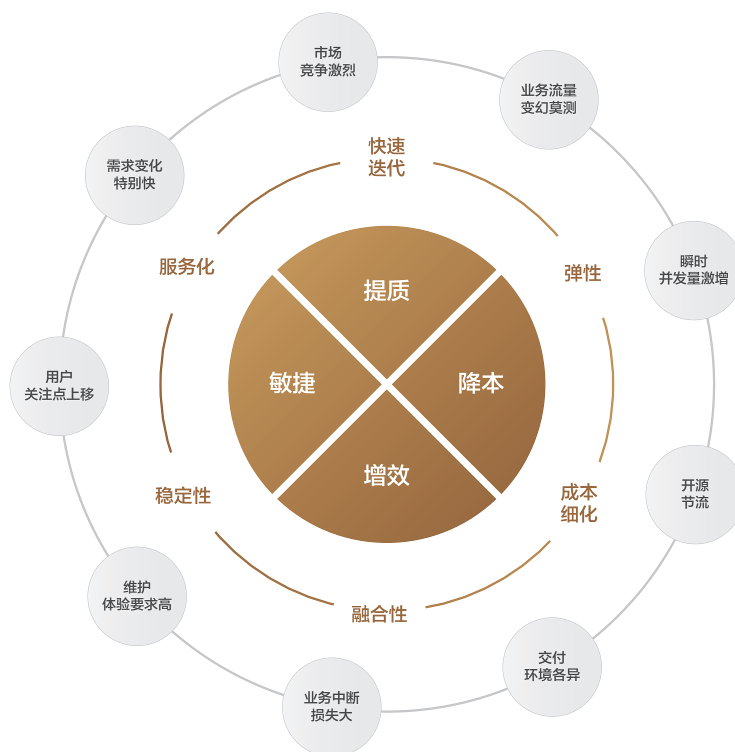
国务院在《新时期促进集成电路产业和软件产业高质量发展的若干政策》中提出要聚焦基础软件、工业软件、应用软件等关键核心技术研发，不断探索构建社会主义市场经济条件下关键核心技术攻关新型举国体制。

2021 年 12 月

中央网络安全和信息化委员会在《“十四五”国家信息化规划》中提出发展目标：数字技术创新体系基本形成。关键核心技术创新能力显著提升，集成电路、基础软件、装备材料、核心元器件等短板取得重大突破。

多重需求驱动，云原生时代中间件应积极求变

数字化浪潮改变了社会的生产生活方式，用户需求快速变化，市场机遇转瞬即逝，企业的管理、运营、组织模式必须进行根本性的变革。极速迸发的社会发展需求驱动创新技术不断涌现，云原生作为云时代的技术内核是企业优化转型的底层支撑，是企业在高频竞争中保持优势的关键要素。



从“上云”到“用云”，云原生已成为用云的标准路径，能够最大程度发挥云的价值。随着底层技术架构的云原生化转型，传统应用也在向云原生方向逐步演进，对支撑应用运行的关键要素—中间件提出了适配要求。根据当前的市场、业务、用户现状，中间件需具备快速迭代、弹性、服务化、稳定性、融合性、成本可控等特性，从而使用户达成敏捷、提质、增效、降本的核心收益。

中间件并非独立运行的个体，用户视角的中间件能力等同于中间件自身能力加上平台对中间件的支撑管理能力。在此背景下，中国信通院牵头联合各相关企业，历经半年制定了面向中间件的《云原生能力成熟度模型 第5部分：中间件》和面向平台的《云原生平台中间件管理能力要求》两项行业标准。

云原生中间件成熟度度量模型

云原生中间件成熟度模型基于云原生的典型特性，从弹性、可观测性、可移植性、可维护性、高可用性、安全性、开放性七个方面衡量中间件的云原生化程度，并将云原生中间件的成熟度水平划分为5级，有助于定位中间件当前的云原生化阶段，为下一步建设方向提供切实指引。

1级 初始级：开始云原生化尝试，可部署在云原生环境；
2级 基础级：初步云原生化，可运行在云原生平台上并执行简单的管理操作；
3级 全面级：基本云原生化，利用云原生技术对自身架构进行部分改造，与云原生平台适配度高；
4级 优秀级：深度云原生化，利用云原生技术对自身架构做深度改造，与云原生平台能力深度融合与协作；
5级 卓越级：全面云原生化，完全基于云原生技术构建，可灵活交付并完全融入到任意云原生平台，可实现智能化运维与治理。

云原生中间件成熟度模型

能力域	弹性	可观测性	可移植性	可维护性	高可用性	安全性	开放性
能力项	弹性伸缩	日志	制品形态	版本管理	架构设计	权限对接	能力扩展
		监控	部署	配置管理	容灾	资源隔离	开放式API
		链路追踪	卸载	状态管理	数据可靠性	安全增强	
			环境兼容	客户端	故障处理		

云原生平台中间件管理能力度量模型

为满足不同场景的业务需求，企业通常需要使用不同功能不同种类的中间件，云原生平台中间件管理能力度量方法从接入管理、生命周期治理、运维支撑、运营服务、高可用、兼容性、开放性、安全性八个部分考察平台的中间件管理能力，既适用于具备中间件管理能力的通用云原生平台，也适用于独立的云原生中间件管理平台。

接入管理方面

平台应提供所纳管中间件的制品包规范，包括对制品目录、中间件基本信息、依赖组件、日志与监控接入方式等方面的要求，并按照质量规范对待接入的中间件进行验收，从而能够灵活纳管来自开源、自研、ISV厂商等多渠道的中间件。

生命周期治理方面

平台需要对中间件的全生命周期链路的各核心环节提供完善的支撑能力，包括规范化线上化的上下架、订阅、退订流程，可视化自动化的部署、变更方案，合理的集群内、集群外服务暴露能力等。

运维支撑方面

平台需要从事前、事中、事后三个角度为中间件提供运维保障措施。事前体现为数据采集，包括日志、监控等关键信息；事中表现为分析与排查，结合巡检结果，以及已采集的数据提供告警、故障诊断、根因分析和价值挖掘等服务；事后表现为运维操作，包括对中间件的资源容量及配置的管理，有状态数据的备份与恢复等措施。

运营服务方面

平台需提供用于优化服务水平的丰富能力，包括权限、多租户等资源控制方案，细致的审计记录，多维度的计量模型与计费方案，清晰的多层级总览视图以及详尽的统计报表。

高可用方面

平台需保障在故障场景下，依旧能够正常对外提供服务，包括中间件的高可用方案，以及平台自身的高可用方案；在兼容性方面，平台需支持基于丰富的底层异构架构之上的中间件管理；在开放性方面，平台需提供各类开放接口实现与外部系统的便捷交互；在安全性方面，平台需提供中间件相关的如密钥、证书、安全漏洞识别等安全保障措施。

总结

传统中间件由于出现时间长，设计初衷是用来解决所诞生时代的特性问题，无法避免地存在一定的时代局限性，对新场景新生态的适配能力较差。云原生作为数字时代的关键技术，能够有效赋能中间件的转型升级，提升差异化竞争力。中间件与云原生的融合是必然趋势，未来的中间件都将为云原生环境而设计，基于云原生技术而构建。

云原生 Cloud Native

技术热点

云原生数据库的进化逻辑 ---------- 15-17

为什么 DevOps 需要一个内部开发者平台 ---------- 18-20

K8s + Volcano：在线离线混部，打造降本增效的银弹 ---------- 21-25

Karmada 跨集群优雅故障迁移特性解析 ---------- 26-31

Serverless 容器启动加速新方法 ---------- 32-36

从日活千万的应用改造谈以 Hudi 打造数据湖新范式 ---------- 37-39

数据领域难题（一）：企业级本体与数据语义图谱构建技术 ---------- 40-42

云原生数据库的进化逻辑

“我觉得传统主备数据库挺好，同样能够解决当前的数据问题，不一定非要用云数据库。”在某些企业中，负责数据库工作的一些传统DBA仍固守这样的想法。在实践中，虽然一些企业在某些场景中还在使用传统主备数据库，但在遇到瓶颈时，也会自然地想到，能不能实现更快捷的容器化部署？当CPU使用率较低时将资源释放掉，而当CPU占用率较高时，能不能实现弹性扩容？当机群中的数据库实例较多时，能不能有一个更高效的管理工具？

不管你是采用传统的数据库，还是实现了数据库的云化部署，归根结底，用户的诉求是一致的，即更好地解决数据同步、数据存储和处理、数据库的高效管理等问题。以应用为中心设计的云原生数据库，能够让应用更智能、更高效地使用数据库，从而构建起敏捷智能的企业数字化业务。特别是进入云原生2.0时代，新应用、新场景驱动数据库实现三大转变，即从以资源为中心到以应用为中心、从以地域为中心到以流量为中心、从以负载为中心到以数据为中心转变。云原生数据库与生俱来的优势与特征，促使越来越多的用户更加坚定地选择了云原生数据库。

云原生不能穿新鞋走老路

作为一种新的趋势或者说潮流，当云原生向你走近时，你选择保守地远远观望，还是积极地拥抱，其实到最后推动你做出最终选择的力量还是“这一趋势是否真正有利于应用的进步”。在实际应用中有这样的例子，有些企业“不得不”选择了云原生数据库，因为他们发现只有云原生数据库的某些能力才能解决其业务面临的问题。举例来说，华为云云原生数据库GaussDB for MySQL，其单实例存储容量可以轻松达到128TB，而开源MySQL数据库当数据超过几个TB，备份就会出现问题，但是基于云原生数据库的快照能力，大数据量的备份和恢复都不在话下。

面对技术和业务转型带来的压力，有些企业会担心，如果采用云原生数据库这样的新生事物，可能会带来较大的挑战和难度。这种想法固然无可厚非，但是穿新鞋走老路，既希望获得云原生数据库的种种先进能力，同时又不想改变传统的部署和应用方式，现实中是没有这样的折中道路的。实际上，采用云原生数据库，对于应用的改变是非常小的，企业原有的应用开发其实不会有什么变化。云原生数据库从技术的角度讲虽然是一种颠覆，但从开发者使用的层面来说，反而是一种更友好的方式。随着时间的推移，也许在一两年之后，大多数企业用户都会慢慢接受云原生数据库的理念，习惯以云原生的方式解决遇到的业务问题。中国信通院的调查显示：57.9%的受访企业表示，会考虑使用云原生数据库，并将其应用到主要业务系统中去；80% 以

上的企业认为，云原生数据库是未来的发展方向。云原生数据库与传统数据库云服务化的形式在未来一段时间将会并存。但是，随着数据库与云计算技术的不断发展，云原生数据库因其智能运维、弹性伸缩及安全稳定等特性，在企业核心业务和高负载场景下的占比会逐渐提升，从而成为企业数据库使用的标准模式。Gartner预测，到2025年，基于云原生平台的数字化业务比例将达到95%，这无疑将激发云原生数据库市场的快速增长。

架构重构 数据融合

不知道大家是不是与我有同感，以前数据库厂商发布新产品时，总是会大谈特谈数据库的性能，比如创造了某性能测试的世界纪录等，让人有一种"唯性能论"的感觉。但是现在，数据库厂商更愿意谈的是数据库如何从应用出发，更好地解决数据的问题，为业务创造价值。

"谈性能不能脱离资源的约束情况，用无穷的资源堆出一个高性能，与数据库在实际项目中体现出的性能是两回事。"华为云数据库服务产品部总经理苏光牛曾指出，"并不是说数据库的性能过时了，而是谈论性能一定结合具体的业务场景。另外，单纯追求高性能，而忽视可靠性、可用性是没有意义的；而抛开性能，可靠性与可用性同样是空中楼阁。"

企业当前更加关注数据的增长、多样性以及业务，如何做好数据的关联，以便更好地使用数据库，需要做出权衡。企业应该从实际业务场景出发，结合自身的数据与业务流的关系，考虑如何解决实际问题，而不是只关注简单的性能指标。针对用户的需求，华为云数据库架构做了三个方面的升级：首先实现Severless化，打造极致的弹性；其次推动数据库的Regionless，即从本地到Global级服务的扩展，全域可用；最后实现Modeless，多模融合，将体验、效率提升到极致。

云原生数据库并不是简单地将数据库服务化，而是对整个架构进行重构。云原生数据库最典型的特质是存算分离的架构，即计算和存储都具有足够的弹性，这其实也是云的基本特性。因为采用了存算分离的架构，所以容量也就有大幅度提升。而容量变大之后，亟需解决的就是数据处理，因为原有的处理逻辑会发生变化，尤其是采用分布式处理后，必然要实现算子下推和并行计算能力的结合，将一个请求分解成多个并行的查询，分解到不同的节点上去执行，只有这样才能充分发挥存算分离架构的优势。

在面对数据多样性带来的挑战时，就要利用到多模技术。“现在数据库的多模分两种，一种是OLTP与OLAP的融合，另一种是在不同层次下多种数据模型的存储与处理的融合。从发展趋势看，业务分类其实越来越模糊化，交易型事务和分析型事务相互交叉，兼而有之。”苏光牛表示，“云原生数据库必须具有支持HTAP混合负载的能力。用户不必再区分什么逻辑放到事务型数据库，什么逻辑放到分析型数据库，逻辑相对应用是透明、无感知的。这是云原生数据库必须很好解决的问题。”

随着云原生时代的到来，包括大数据、社交、地理信息、人工智能和高性能计算等新兴应用和负载也在考验着数据库的“抗压”能力。在2008年之前，市场上主要是交易型数据库，而今天非关系型数据库已经大行其道，数据库类型更加多样化。华为云的思路就是用NoSQL多模数据库架构来解决支持KV、文档、时序、宽表等数据类型。所有这些类型的数据库都可以放到一个存储资源池中，统一接口，基于开源的协议或通过华为云自己的SDK生态、基于统一的结构来访问，这样也更便于应用开发。现在，越来越多的厂商开始追逐云原生数据库的潮流，不管这些不同的云原生数据库产品底层采用的是什么编码技术，最核心的是要对架构进行重构，实现存算分离，并与云的相关技术充分结合，包括分布式存储、容器化部署等，而不是拿一个开源数据库仅仅对其进行服务化而已。

向“一切皆服务”迈进

得益于领先的技术能力和大量独创性，华为云GaussDB系列数据库产品已连续两年入选Gartner全球云数据库魔力象限，在IDC发布的报告中，也处于国产数据库本地部署市场第一的位置。目前，华为云GaussDB已经在超过2500多家客户中实现了规模化商用，行业覆盖金融、电商、社交文娱、汽车、制造、能源等，也正是因为经历了大量的行业应用场景的洗礼，结合自身大量的技术和人才储备，华为云数据库才能在遇到实际的业务挑战时，不断帮助客户克服困难、持续创新，做到行业领先甚至独创，未来，在技术及服务理念的牵引下，华为云数据库将在Serverless、Regionless、Modeless三个方向持续创新演进，让客户能够像使用水和电一样，更便捷地使用云原生数据库。

为什么 DevOps 需要一个内部开发者平台

软通动力云原生研究中心

越来越多的组织开始搞敏捷和 DevOps 转型，打造了很多的 DevOps 基础设施，比如有管理需求的 Jira， 有持续集成的 Jenkins，有容器编排的 K8S 等等。可是这纷繁复杂的 DevOps 工具链，同时也给企业带来新的困扰。

想象一下如果企业有10几个产品研发团队，每一个团队独立维护自己的技术栈、工具链和流程。这些团队大概率都会去解决类似的问题，导致在技术栈的集成、基础设施维护等方面投入太多的时间和人力，这些成本本应该投入到团队实际负责的产品价值创造中。同时缺少对技术和流程的标准化，也会产生其他问题：跨团队的知识共享会变得非常困难；在云原生时代，许多产品开发团队实际上并不具备基础设施架构运维的专业知识和技能。

因此需要有一个具备自服务 API、工具、服务、知识库和技术支持能力的内部数字化平台产品，应用产品开发团队可以利用该平台以更快的速度交付产品功能，减少协调。

CI/CD Pipeline 在 DevOps 中的重要性

DevOps 就是为了打破开发人员和运维人员之间的“Fence”，而CI/CD Pipeline 可以帮助实现这个目的。同时，CI/CD Pipeline 有利于缩短应用的发布周期，提升应用的发布效率，为市场需求做出及时的响应。

Gene Kim 主创的小说体 IT 管理读物《凤凰项目》是公认的 DevOps 入门第一本书，书中比尔团队的故事很好佐证了上述观点，当比尔团队厘清业务部门（ Dev ）与 IT 部门（ Ops ）在凤凰项目中的部署流程，并通过自动化“流水线”改造后，极大提升了凤凰项目发布频率和成功率的同时，业务部门和 IT 部门的矛盾也变得越来越少。

DevOps 为什么需要可观测

企业管理协会（ Enterprise Management Associates ）小组最近的研究表明，DevOps 团队在2021年面临的第一大挑战是可观测性。报告显示，开发和运维团队 50% 的时间都花费在了查明问题的根本原因上面。此外，专注于可观测性的团队能够将开发速度提高 70%，并以将近四倍的功能数量保持更高的产品研发速度。

如何打造 CI/CD Pipeline 可观测性

目前 Pipeline 虽然有广泛的应用，但低效率的、不稳定的 Pipeline 往往会影响变更的部署频率，降低交付质量，更严重的甚至会阻塞开发流程。实际使用 Pipeline 过程中，会有很多因素导致运行一次 Pipeline 的时间会很长，比如：

- 可以并行的任务没有并行执行，等待的任务拉长了整体执行时间
- 运行 Pipeline的运行时（RunTime）的资源不足，导致任务排队时间太长
- 没有使用缓存，虽然是运行相同的构建，但每次都需要重新下载全部依赖

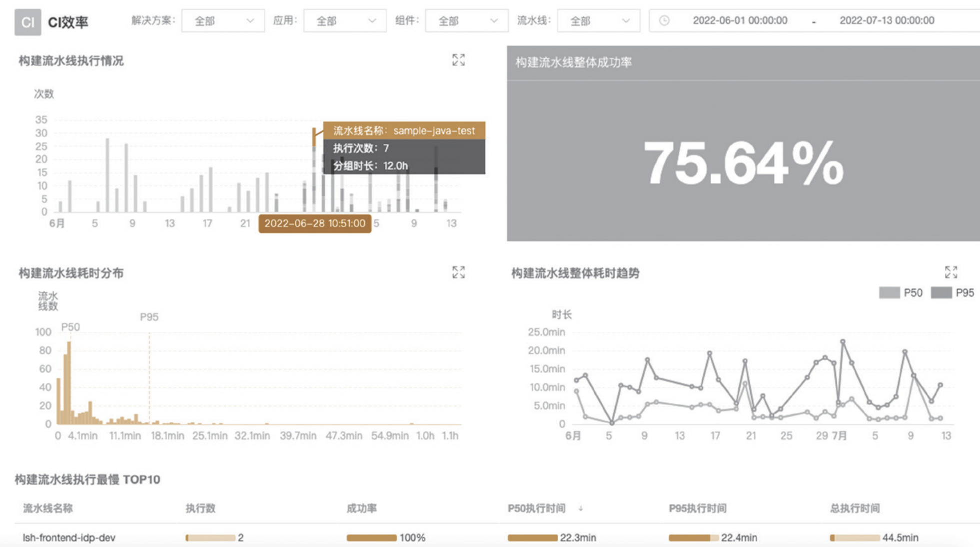
因此商用的 CI/CD Pipeline 必须具备良好的可观测性。

下图来源于 MiK Kersten 的《Project to Product》，Flow Time 是衡量一件事从开始到结束需要多长时间。在 DevOps 中，Flow Time 是指客户需求（Request）被接纳时，计时开始，当对应该需求的变更生效并在生产环境中运行时，计时结束。

价值流示意图

具备价值流能力的CI/CD Pipeline可以自动化得跟踪和统计组织内组件的迭代 FLow Time，直观反应组织内不同团队的 E2E 软件交付效率。

CI/CD Pipeline 自身的数据可视化，度量指标包括构建频率、故障率、平均或P95执行分布时间等，不仅可以让开发人员能过深入了解 CI 测试和 pipeline 执行失败的原因、监控测试集的实际耗时，同时也可以了解 CI/CD 工程本身的运行健康状况和性能。



可观测性不是“银弹”

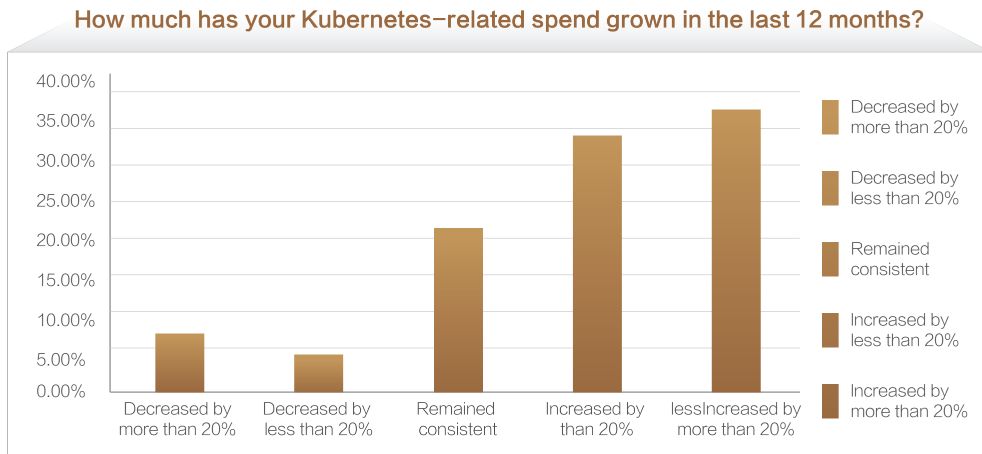
数据驱动的DevOps 工程化虽然有助于研发效能提升，但是真正研发效能的提升的核心还是在于研发团队工程技能的提升，比如需求分析技能、架构设计技能、编码技能等。所谓 DevOps 就是将人、流程和技术结合起来，持续为客户提供价值。

K8s + Volcano：在线离线混部，打造降本增效的银弹

牛杰　华为云容器专家

随着云原生技术的成熟，各大企业意识到云原生是云计算发展的必然趋势，也在不断向云原生基础设施方面不断投入，以及推动企业产品架构向云原生架构的演进。

CNCF Annual Survey 2021中显示，k8s已经成为全球主流技术。目前有560万开发人员使用k8s，96%的企业正在使用或评估使用k8s，在FinOps Kubernetes Report 2021中显示，越来越多的用户将自己的应用迁移到Kubernetes平台。在过去的一年中，68%的受访者表示在云原生基础设施上的投入有所增加，其中有一半的受访者表示增加超过20%。



进一步调查发现用户应用在云原生化后，K8S集群节点的CPU利用率不足15%。是什么因素阻碍了资源利用率的提升呢？

在调研不同类型客户，排除一些空闲等干扰因素后，我们发现造成资源利用率低的主要因素可归纳为如下四点：

1）集群规划粒度过细，节点分布过散

集群规划粒度过细，节点分布在多个不同的k8s集群中，使得计算资源无法共享，计算资源碎片数量增加。

2) 节点规格没有跟随应用迭代而变化，资源分配率低：

初期节点规格与应用规格匹配度较好，资源分配率较高；随着应用版本迭代，应用申请资源发生变化，与节点规格比例差异较大，使节点分配率降低，计算资源碎片数量增加。

3) 业务“潮汐”特性明显，预留资源较多：

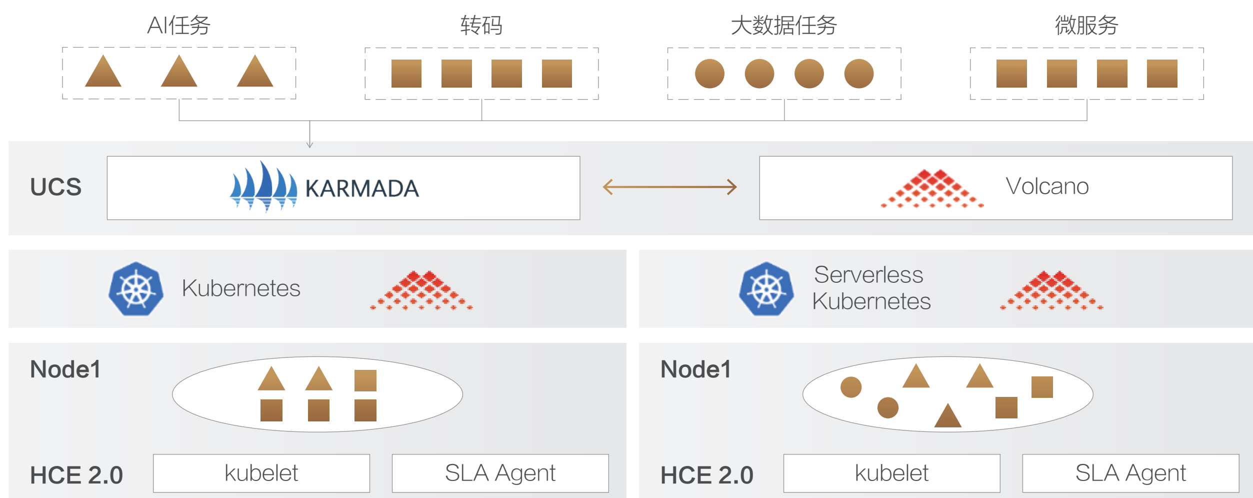
在线业务具有明显日级别波峰、波谷特性，用户为保证服务的性能和稳定性按照波峰申请资源，集群的大部分资源处于闲置状态。

4) 在线和离线作业分布不同集群，资源无法分时复用：

用户为在线和离线作业划分不同的k8s集群中，在线业务在波谷时，无法部署离线作业使用这部分资源。

这些都是云原生应用粗狂发展阶段的典型表现。在业务云原生化过程中，不同的业务架构有着不同的部署方案，不同架构的应用有着不同的演进节奏，不同的团队有着性能和服务质量的平衡点。面对这样复杂的场景，我们应该如何化繁为简，帮助用户有步骤的提升资源利用率和控制成本呢？

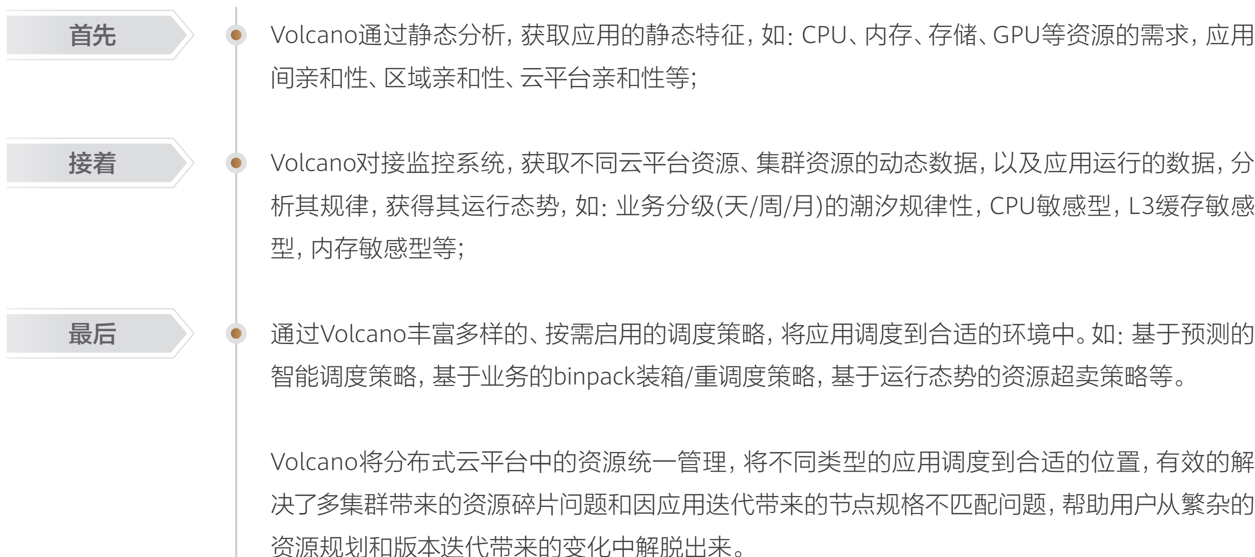
华为云容器团队通过多年在混合部署领域的探索和实践，围绕Volcano和Kubernetes生态，构建帮助用户提升资源利用率，实现降本增效的云原生混部解决方案。



如上图所示，混部不是简单将小集群合并成一个大集群，然后将多个不同的业务部署在同一个集群中那么简单。我们需要确保用户的应用能够部署到合适的位置，并能保障其需要的资源。这也是云原生混部解决方案中的两个核心设计：全域统一调度和资源分级管控。

全域统一调度

应用的全域统一调度的核心是全域和统一，比如：分布式云场景中跨云、跨集群的统一调度，以及不同在线应用、离线任务的统一调度。



首先

Volcano通过静态分析，获取应用的静态特征，如：CPU、内存、存储、GPU等资源的需求，应用间亲和性、区域亲和性、云平台亲和性等；

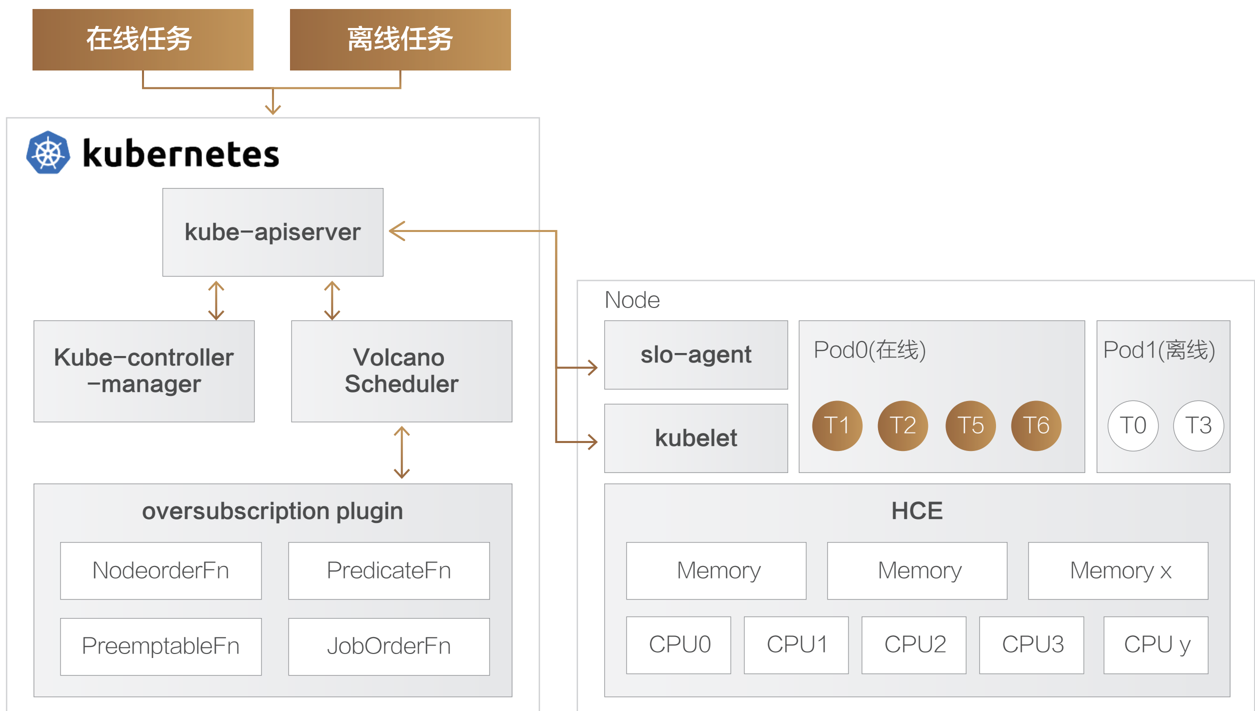
接着

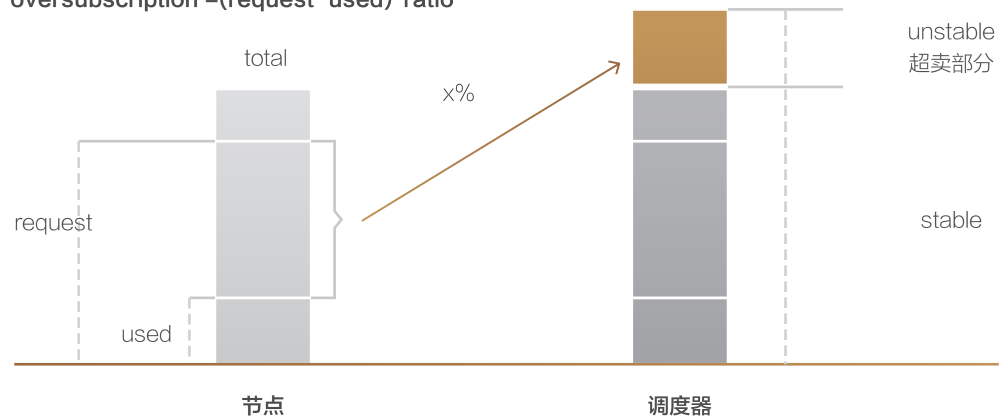
Volcano对接监控系统，获取不同云平台资源、集群资源的动态数据，以及应用运行的数据，分析其规律，获得其运行态势，如：业务分级(天/周/月)的潮汐规律性，CPU敏感型，L3缓存敏感型，内存敏感型等；

最后

通过Volcano丰富多样的、按需启用的调度策略，将应用调度到合适的环境中。如：基于预测的智能调度策略，基于业务的binpack装箱/重调度策略，基于运行态势的资源超卖策略等。

Volcano将分布式云平台中的资源统一管理，将不同类型的应用调度到合适的位置，有效的解决了多集群带来的资源碎片问题和因应用迭代带来的节点规格不匹配问题，帮助用户从繁杂的资源规划和版本迭代带来的变化中解脱出来。





资源分级管控

应用被调度到合适的运行环境后，如何来保障其所需要的资源呢？

我们基于HCE2.0操作系统，从CPU、L3缓存、内存、网络、存储等全方位提供资源隔离能力，并以内核态为主，辅以用户态，通过快速抢占（毫秒）和快速驱逐（秒级），保障在线业务的服务质量。

1) 资源隔离的措施

如：CPU的绑核、NUMA亲和性、潮汐亲和特性，网络带宽控制等，有效的保障资源敏感型业务的SLO；

2) 资源优先级控制的措施

如：CPU分级控制、内存分级压制、网络优先级控制、磁盘IO的优先级控制等，在提升资源分配率的同时，又少影响或不影响优先级高的业务SLO。

资源分级管控为业务潮汐明显的在线业务间混部、在线和离线业务混部奠定了基础。解决了应用预留资源较多、资源无法分时复用的问题。

在线和离线业务的混部是提升资源利用率的一个有效的解决方案。华为云也推出了CCE Turbo服务，支持在线离线应用混部的商用，为用户提供一种保障在线业务SLO前提下的云原生应用混部解决方案。

主要为用户提供：

1) 在线离线云原生应用的混部

为用户提供一种保障在线应用SLO的分时复用，提升单位成本利用率的措施；

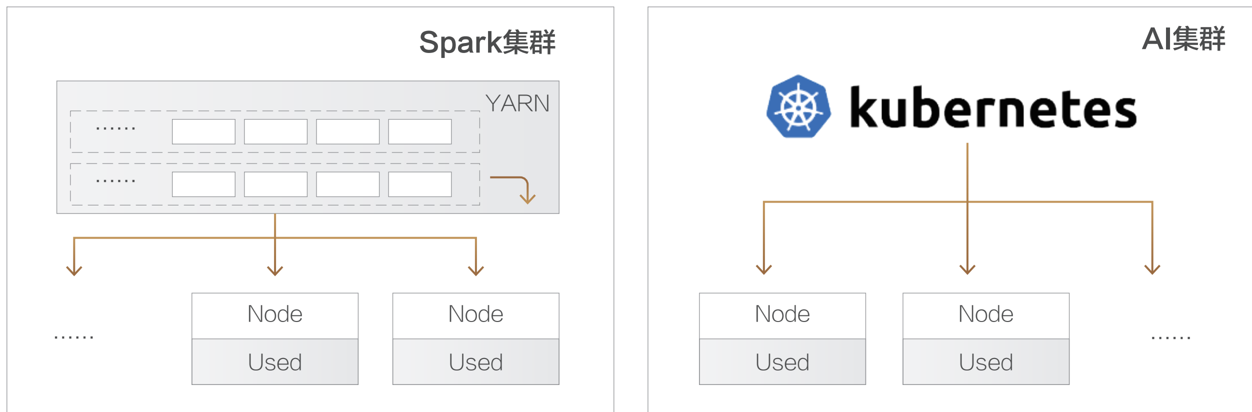
2) 资源超卖

通过统计和分析CPU/Memroy的使用率，智能的将空闲资源按照一定比例分配给离线作业使用，进一步提升单位成本利用率的解决方案；

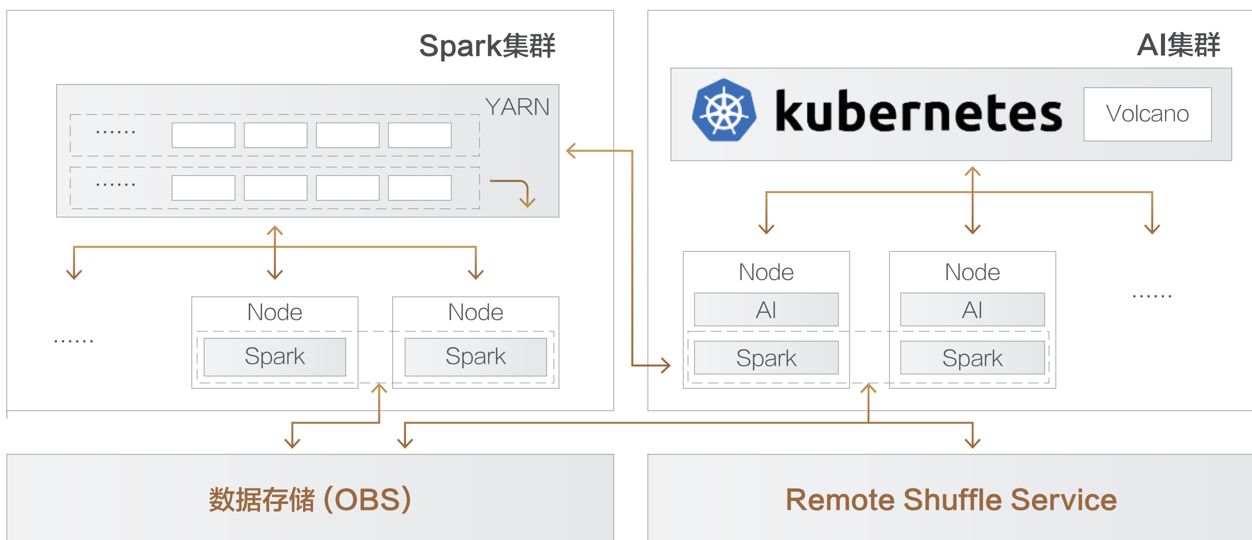
3) QoS保障

业务定级，智能感知节点负载，高负载场景，智能驱逐或压制低优负载，保障高优（在线）业务SLO。在线业务的性能（时延&吞吐）劣化幅度控制在单独部署在线业务性能的5%之内。

云原生混部解决方案已经为很多华为云客户提供服务，如：Q企业的AI和Spark业务，分别部署在不同的K8S集群中，如下图所示：



随着业务的发展，出现了资源不均衡的问题，也即：Spark 集群资源不足，资源利用率 几乎 100%，且有大量作业排队等待。而AI 集群资源平均使用率在20%左右，且存在周期性波峰波谷。用户在AI集群中启用在线离线混部和资源超卖能力，并将spark任务部署到AI集群中，让Spark业务（离线任务）和AI业务（在线任务）分时复用AI集群的资源，以分担Spark集群负荷，提升AI集群资源利用率的目标。经过多次优化，Spark集群平均资源利用率下降到75%-90%之间，AI集群平均资源利用率提升至35%-40%之间。



在这世界局势风云变幻、全球经济困难重重、新技术不断推陈出新的复杂环境下，华为云容器团队将继续在帮助用户降本增效之路上推陈出新，不仅推出用户可直接使用的商业化服务，更是将核心能力开源到社区，与社区一起完善混部这颗“银弹”，为客户提供一个易观测、能推荐、可治理的云原生成本管理解决方案。

Karmada 跨集群优雅故障迁移特性解析

常震　华为云云原生团队/Karmada社区

在多云多集群应用场景中，为了提高业务的高可用性，用户的工作负载可能会被部署在多个集群中。然而当某个集群发生故障时，为保证业务的可用性与连续性，用户希望故障集群上的工作负载被自动的迁移到其他条件适合的集群中去，从而达成故障迁移的目的。

Karmada 在 v1.0 版本发布之前便已支持跨集群故障迁移能力，经历过社区多个版本的开发迭代，跨集群故障迁移能力不断完善。在 Karmada 最新版本 v1.3（https://github.com/karmada-io/karmada/tree/release-1.3）中，跨集群故障迁移特性支持优雅故障迁移，确保迁移过程足够平滑。下面我们对该特性展开解析。

回顾：单集群故障迁移

在 Kubernetes 的架构中，Node 作为运行 Pod 实例的单元，不可避免地面临出现故障的可能性，故障来源不限于自身资源短缺、与 Kubernetes 控制面失去连接等。提供服务的可靠性、在节点故障发生后保持服务的稳定一直是 Kubernetes 关注的重点之一。在 Kubernetes 管理面，当节点出现故障或是用户不希望在节点上运行 Pod 时，节点状态将被标记为不可用的状态，node-controller 会为节点打上污点，以避免新的实例调度到当前节点上、以及将已有的 Pod 实例迁移到其他节点上。

集群故障判定

相较于单集群故障迁移，Karmada 的跨集群故障迁移单位由节点变为了集群。Karmada 支持Push 和 Pull 两种模式来管理成员集群，有关集群注册的信息可以参考Cluster Registration（http://karmada.io/docs/next/userguide/clustermanager/cluster-registration/）。Karmada 根据集群的心跳来判定集群当前的状态。集群心跳探测有两种方式：1.集群状态收集，更新集群的 .status 字段（包括 Push 和 Pull 两种模式）；2.控制面中 karmada-cluster 命名空间下的 Lease 对象，每个 Pull 集群都有一个关联的 Lease 对象。

集群状态收集

对于 Push 集群，Karmada 控制面中的 clusterStatus-controller 将定期执行集群状态的收集任务；对于 Pull 集群，集群中部署的 karmada-agent 组件负责创建并定期更新集群的 .status 字段。集群状态的定期更新任务可以通过 --cluster-status-update-frequency 标签进行配置（默认值为10秒）。集群的 Ready 条件在满足以下条件时将会被设置为 False ：• 集群持续一段时间无法访问；• 集群健康检查响应持续一段时间不正常。上述持续时间间隔可以通过 --cluster-failure-threshold 标签进行配置（默认值为30秒）。

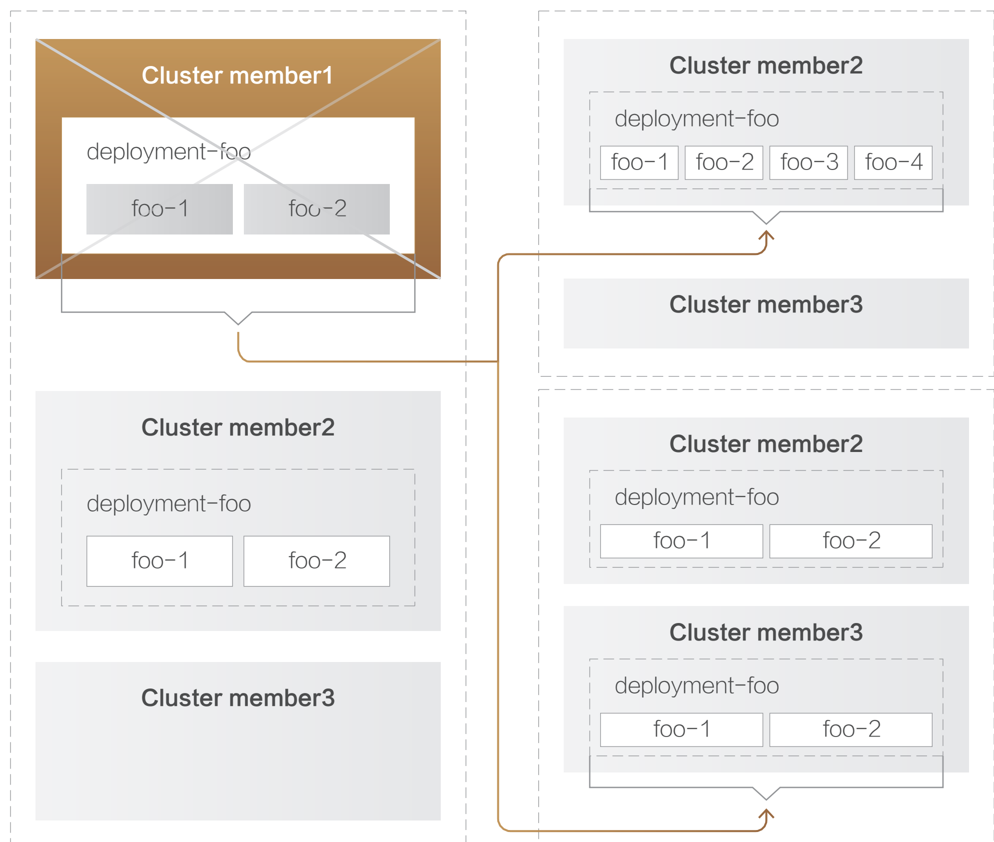
集群 Lease 对象更新

每当有 Pull 集群加入时，Karmada将为该集群创建一个 Lease 对象和一个 lease-controller。每个 lease-controller 负责更新对应的 Lease 对象，续租时间可以通过 --cluster-lease-duration 和 --cluster-lease-renew-interval-fraction 标签进行配置（默认值为10秒）。由于集群的状态更新由 clusterStatus-controller 负责维护，因此 Lease 对象的更新过程与集群状态的更新过程相互独立。Karmada 控制面中的 cluster-controller 将每隔 --cluster-monitor-period 时间（默认值为5秒）检查 Pull 集群的状态，当 cluster-controller 在 --cluster-monitor-grace-period 时间段（默认值为40秒）内没有收到来着集群的消息时，集群的 Ready 条件将被更改为 Unknown 。

检查集群状态

你可以使用 kubectl 命令来检查集群的状态细节：kubectl describe cluster

故障迁移过程



集群污点添加

当集群被判定为不健康之后，集群将会被添加上Effect值为NoSchedule的污点，具体情况为：

- 当集群 Ready 状态为 False 时，将被添加如下污点：key: cluster.karmada.io/not-ready effect: NoSchedule

当集群 Ready 状态为 Unknown 时，将被添加如下污点：key: cluster.karmada.io/unreachable effect: NoSchedule 如果集群的不健康状态持续一段时间（该时间可以通过 --failover-eviction-timeout 标签进行配置，默认值为5分钟）仍未恢复，集群将会被添加上Effect值为NoExecute的污点，具体情况为：

- 当集群 Ready 状态为 False 时，将被添加如下污点：key: cluster.karmada.io/not-ready effect: NoExecute
- 当集群 Ready 状态为 Unknown 时，将被添加如下污点：key: cluster.karmada.io/unreachable effect: NoExecute

容忍集群污点

当用户创建 PropagationPolicy/ClusterPropagationPolicy 资源后，Karmada 会通过 webhook 为它们自动增加如下集群污点容忍（以 PropagationPolicy 为例）：

```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: nginx-propagation
  namespace: default
spec:
  placement:
    clusterTolerations:
    - effect: NoExecute
      key: cluster.karmada.io/not-ready
      operator: Exists
      tolerationSeconds: 600
    - effect: NoExecute
      key: cluster.karmada.io/unreachable
      operator: Exists
      tolerationSeconds: 600
  ...
```

其中，tolerationSeconds 值可以通过 --default-not-ready-toleration-seconds 与--default-unreachable-toleration-seconds 标签进行配置，这两个标签的默认值均为600。

故障迁移

当Karmada 检测到故障群集不再被 PropagationPolicy/ClusterPropagationPolicy 容忍时，该

集群将被从资源调度结果中移除，随后，karmada-scheduler 重调度相关工作负载。重调度的过程有以下几个限制：•对于每个重调度的工作负载，其仍然需要满足PropagationPolicy/ClusterPropagationPolicy 的约束，如 ClusterAffinity 或 SpreadConstraints 。• 应用初始调度结果中健康的集群在重调度过程中仍将被保留。

复制 Duplicated 调度类型

对于 Duplicated 调度类型，当满足分发策略限制的候选集群数量不小于故障集群数量时，将根据故障集群数量将工作负载重新调度到候选集群；否则，不进行重调度。

```
...
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
        - member3
        - member5
    spreadConstraints:
      - maxGroups: 2
        minGroups: 2
    replicaScheduling:
      replicaSchedulingType: Duplicated
  ...
```

假设有5个成员集群，初始调度结果在 member1和 member2 集群中。当 member2 集群发生故障，触发 karmada-scheduler 重调度。

需要注意的是，重调度不会删除原本状态为 Ready 的集群 member1 上的工作负载。在其余3个集群中，只有 member3 和 member5 匹配 clusterAffinity 策略。由于传播约束的限制，最后应用调度的结果将会是 [member1, member3] 或 [member1, member5] 。

分发 Divided 调度类型

对于 Divided 调度类型，karmada-scheduler 将尝试将应用副本迁移到其他健康的集群中去。

```
  ...
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
```

```
  replicaScheduling:
    replicaDivisionPreference: Weighted
    replicaSchedulingType: Divided
    weightPreference:
      staticWeightList:
        - targetCluster:
            clusterNames:
              - member1
          weight: 1
        - targetCluster:
            clusterNames:
              - member2
          weight: 2
...
```

Karmada-scheduler 将根据权重表 weightPreference 来划分应用副本数。初始调度结果中，member1 集群上有1个副本，member2 集群上有2个副本。当 member1 集群故障之后，触发重调度，最后的调度结果是 member2 集群上有3个副本。

优雅故障迁移

为了防止集群故障迁移过程中服务发生中断，Karmada 需要确保故障集群中应用副本的删除动作延迟到应用副本在新集群上可用之后才执行。ResourceBinding/ClusterResourceBinding 中增加了 GracefulEvictionTasks 字段来表示优雅驱逐任务队列：

```
// GracefulEvictionTasks holds the eviction tasks that are expected to perform
// the eviction in a graceful way.
// The intended workflow is:
// 1. Once the controller(such as 'taint-manager') decided to evict the resource that
//    is referenced by current ResourceBinding or ClusterResourceBinding from a target
//    cluster, it removes(or scale down the replicas) the target from Clusters(.spec.Clusters)
//    and builds a graceful eviction task.
// 2. The scheduler may perform a re-scheduler and probably select a substitute cluster
//    to take over the evicting workload(resource).
// 3. The graceful eviction controller takes care of the graceful eviction tasks and
//    performs the final removal after the workload(resource) is available on the substitute
//    cluster or exceed the grace termination period(defaults to 10 minutes).
//
// +optional
GracefulEvictionTasks []GracefulEvictionTask `json:"gracefulEvictionTasks,omitempty"`
```

当故障集群被 taint-manager 从资源调度结果中删除时，它将被添加到优雅驱逐任务队列中。gracefulEvction-controller 负责处理优雅驱逐任务队列中的任务。在处理过程中，gracefulEvction-controller 逐个评估优雅驱逐任务队列中的任务是否可以从队列中移除。判断条件如下：

» 检查当前资源调度结果中资源的健康状态。如果资源健康状态为健康，则满足条件。• 检查当前任务的等待时长是否超过超时时间，超时时间可以通过graceful-evction-timeout 标签进行配置（默认为10分钟）。如果超过，则满足条件。

总结

Karmada 跨集群优雅故障迁移特性提升了集群故障后业务的平滑迁移能力，希望通过上述分析过程能帮大家更好的理解和使用Karmada 跨集群故障迁移能力。有关该特性的更多详细信息可以参考 Karmada 官网。大家也可以查看 Karmada release （https://github.com/karmada-io/karmada/releases）来跟进 Karmada 最新版本动态。如果大家对 Karmada 跨集群故障迁移特性有更多兴趣与见解，或是对其他特性和功能感兴趣，也欢迎大家积极参与到 Karmada 社区中来，参与社区讨论与开发。

附：Karmada社区技术交流地址

项目地址： https://github.com/karmada-io/karmada

Slack 地址： https://karmada-io.slack.com

社区官网 https://karmada-io

Serverless 容器启动加速新方法

张嘉伟 华为云容器专家

随着Serverless逐渐成为云平台的下一代默认计算范式，越来越多学术研究开始着眼于提升Serverless平台的性能。其中，如何提升收到用户请求后、平台启动容器实例的速度成为研究焦点。基于近两年云原生相关顶级学术会议，笔者按照无请求时是否还维护一定数量的容器将方案分为热启动和冷启动两类进行分析。

热启动

维护一定量运行中的容器，以应对突发的请求能够实现极致的相应时间，但是大量预热的容器会给云平台带来大量的资源开销，不符合当前“降本增效”的需求。

为降低资源开销但同时又最大程度上保留热启动的好处，最先想到的方法就是选择性地进行预热。例如，在计算力不同的异构集群中，可以依据计算任务的请求概率分配预热容器的机器，

高访问概率的使用高端机器预热，而低概率的使用低端的机器预热[1]。该方案中核心设计为基于FFT的请求周期预测算法以及对应预热所使用机型的选择方法。最终相比不加区分的预热降低了45%的成本。

另一个角度则是选择性地预热部分容器[2]，经常被访问的容器则可以进行更多的预热。以上过程与文件和对象的缓存有相似性，即在缓存中准备最可能被访问的文件以保证高hit ratio。借鉴此想法，FaasCache（如图2所示）采用Greedy-Dual文件缓存框架来决策需要预热的容器，并基于hit-ratio曲线来决策虚机的资源配置。经验证该方法可以降低30%资源开销。

相比上述方法，Pagurus [3]采取了另一种思路。考虑到系统存在其他应用已经执行完的空闲容器，可以复用这些空闲容器给待创建容器的任务，来减少启动时延。但重复利用存在三个问题：原容器是否空闲难以判断、函数需要不同的软件包、安全问题。为解决上述问题， Pagurus具有三个核心功能（如图3所示）：检测容器是否为idle、将容器清空并换成带有其他函数包的zygote容器、识别需要扩容的函数并预先分配zygote容器。基于以上设计则可以充分利用空闲的容器。

为了热启动的资源开销，Medes[4]则从内存入手。通过调研发现执行相同应用的热容器有80-90%内存是相同的，这些内存称为reusable sandbox chunk (RSC)，如图4中蓝色标识。基于此发现，Medes对执行相同应用的容器在page-level进行去重，即相同的内存只存储一份，每个sandbox仅保存自己特有的内存，恢复时再将特有内存和相同的内存进行合并，以此达到减少预热容器内存占用。经验证，该方案成功消减67%的内存占用。

图 1

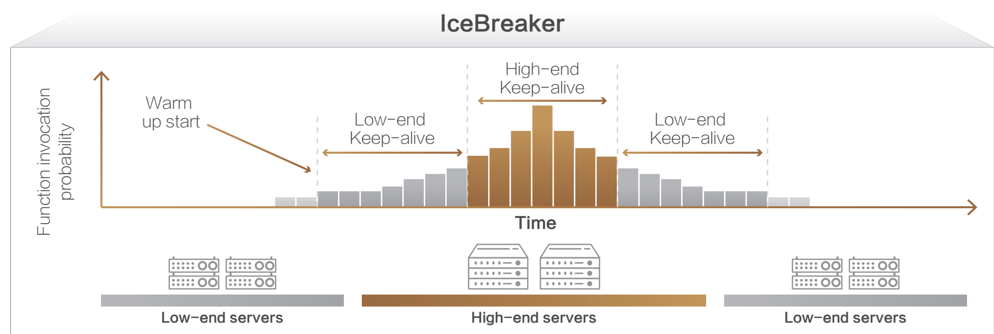


图 2

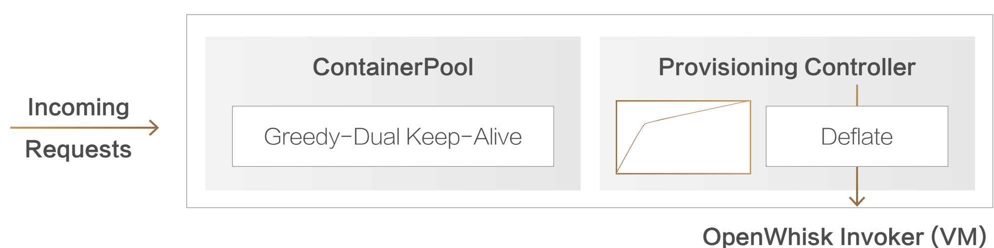


图 3

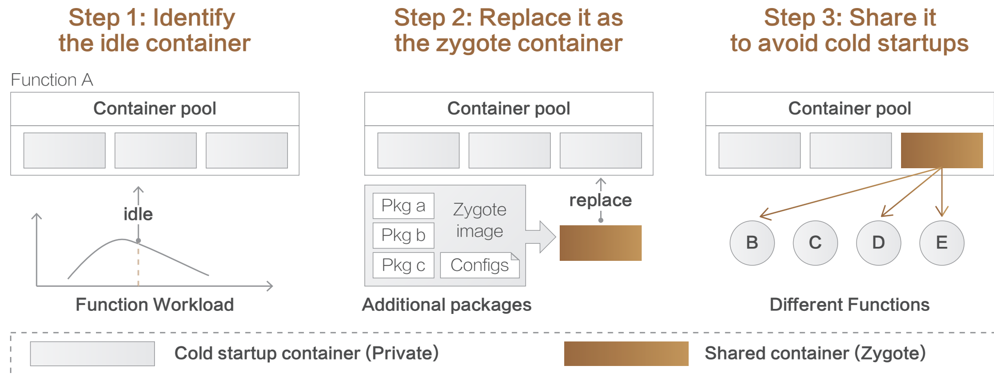
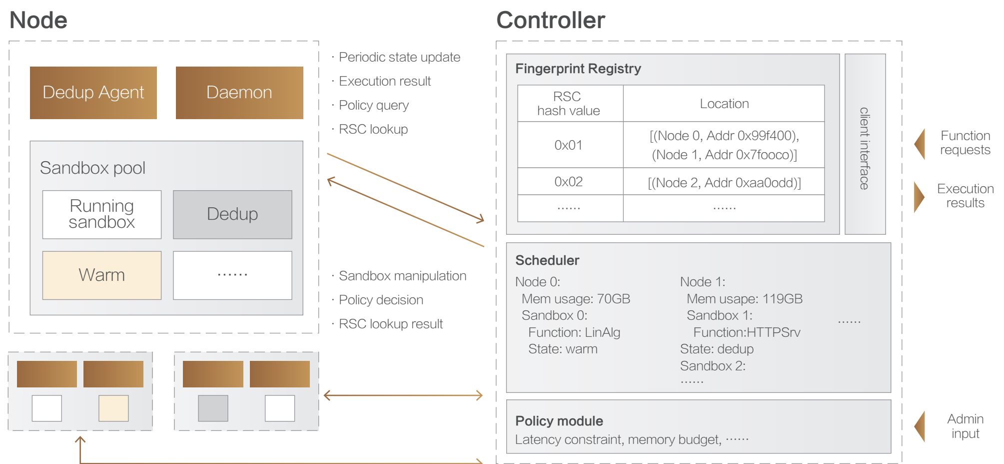


图 4



冷启动

与热启动不同，冷启动不需要维护运行中的容器以接收用户请求，降低了平台成本。但是，仅依靠现有方法在用户请求到来时启动容器，则会带来极大时延，给用户的体验就是E2E时延大。

为提升冷启动中容器启动的速度，爱丁堡大学团队首先设计了性能验证平台vHive [5]以探索启动过程中各阶段的耗时。通过此平台，该团队得出三个结论：1）从snapshot启动容器比冷启动减少61-96%的开销，2）从snapshot启动主要的时延在处理page faults，3）实例在多次调用中访问的内存有97%相同。基于以上结论，该团队设计了基于内存重用的启动方案REAP。如图5所示，REAP在第一次启动实例时跟踪并记录page fault的内存页，存入snapshot中。在后续基于snapshot启动实例时一次性读取并置入用户内存空间。通过上述设计，容器冷启动速度提升了3.7倍。

类似的思想在[6]中也有体现。不同的是，Fireworks主要关注虚机层面snapshot与恢复。Fireworks的核心思想是虚机在OS、JIT语言运行时、用户代码等加载完成之后进行snapshot，如图6所示。当该应用再次被调用并创建多个实例时，虚机实例都从snapshot中恢复并以CoW的方式加载snapshot中已经预加载的内容，最后再在不同实例中输入不同的用户数据。基于上述设计，相比现有平台提升了20.6倍的启动速度，并提升了7.3内存效率。

图 5

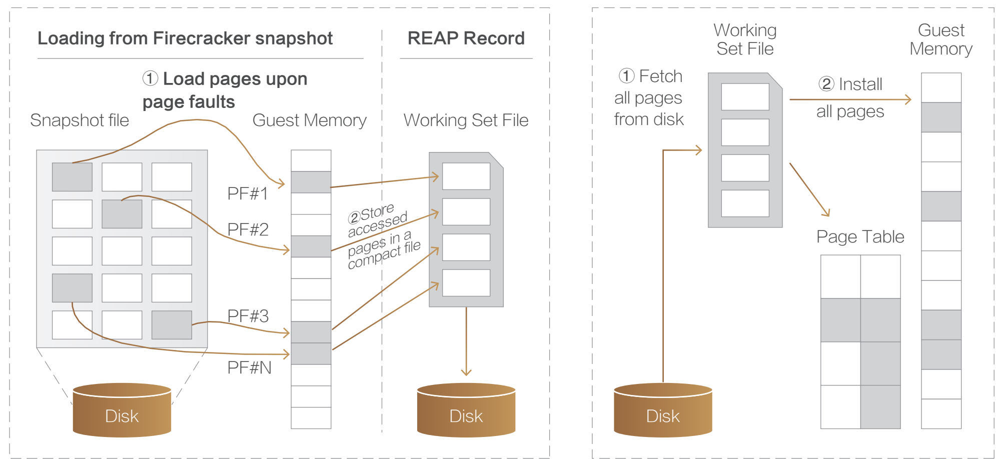
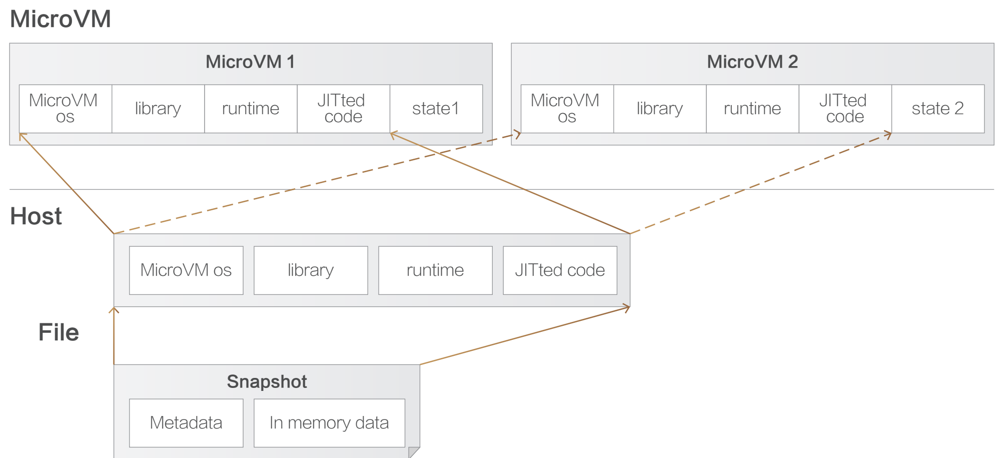


图 6

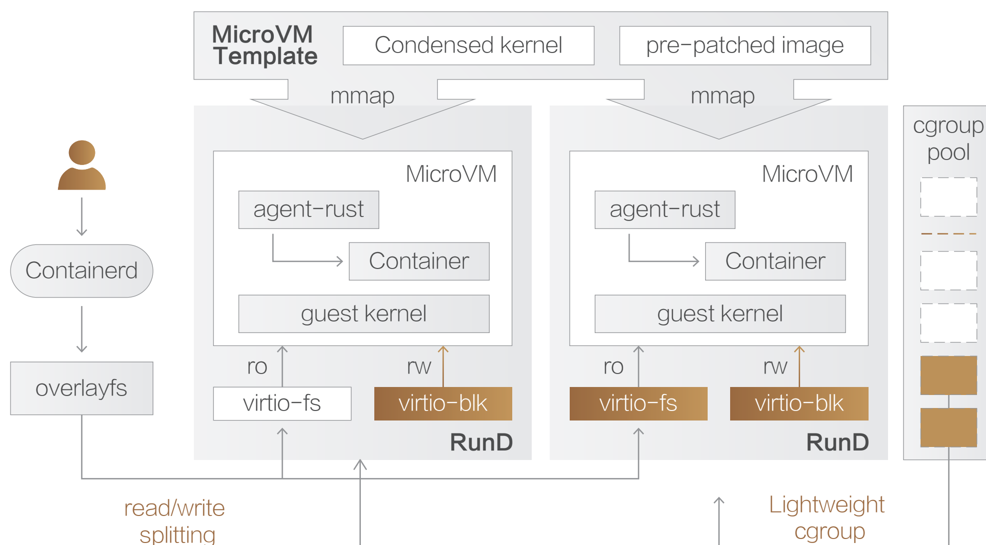


针对MicroVM形态的安全容器，上海交大团队发现冷启动慢主要源于三个因素：1）rootfs在Host OS和Guest OS重复创建page、 2） guest kernel过于笨重、3）cgroups的创建过程是串行的。为了解决上述问题，RunD[7]采取了三个对应的解决方案（如图7所示）： 1）通过virtio-fs和virtio-blk分别将RO和RW层映射进容器，实现从外部挂载读写分离的rootfs，避免重复创建；2）通过裁剪OS特性以及预先执行self-modifying来减少创建VM时Guest OS的开销，3）通过预先创建cgroups并维护cgroups池，在使用时仅需重命名，以此减少创建cgroups的时间。经验证，RunD 能够在88ms内启动容器，并在1s内创建 200个sandboxes。

总结与启示

纵观上述各种容器启动方案，其本质思想是，以系统能够容忍的方式，预先 “缓存”好容器所需要的组件。例如，在CPU和内存相对不紧缺时可以选择热启动，缓存整个容器；若内存资源紧张，则可以在热启动基础上进行内存去重；若CPU资源也紧缺，则可选择冷启动，并利用存储来缓存一些需要加载的内存。

图 7



对于具体某个Serverless平台而言，并非直接使用上述某种技术就可以，而是需要因地制宜，分析当前系统中的最紧缺的资源和相对富裕的资源，进而利用相对富裕的资源来缓存容器启动过程中最耗时步骤所需的组件。

近几年容器启动加速仍然会是学术研究的热点，除了针对具体系统的方案分析和设计外，还可能会针对新出现的runtime（如WASM）、底层硬件变化（如分离式数据中心disaggregated data center）、与周边服务的协同加速（如网关、存储等）等方面进行探索。

参考文献

- R. B. Roy, et.al., "IceBreaker: warming serverless functions better with heterogeneity," in Proc. ACM ASPLOS 2022.
- A. Fuerst, et. al., "FaasCache: keeping serverless computing alive with greedy-dual caching," in Proc. ASPLOS 2021.
- Z. Li, et al.,"Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing Through Inter-Function Container Sharing," in Proc. USENIX ATC 2022.
- D. Saxena, et al., "Memory deduplication for serverless computing with Medes," in Proc. EuroSys 2022.
- D. Ustiugov, et al., "Benchmarking, analysis, and optimization of serverless function snapshots." Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021.
- W. Shin, et al., "Fireworks: a fast, efficient, and safe serverless framework using VM-level post-JIT snapshot," in Proc. EuroSys 2022.
- Z. Li, et al., "RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing," in Proc. USENIX ATC 2022.

从日活千万的应用改造谈以 Hudi 打造数据湖新范式

晓隽　华为云大数据高级架构师

采用 Hudi 改造的应用场景分析

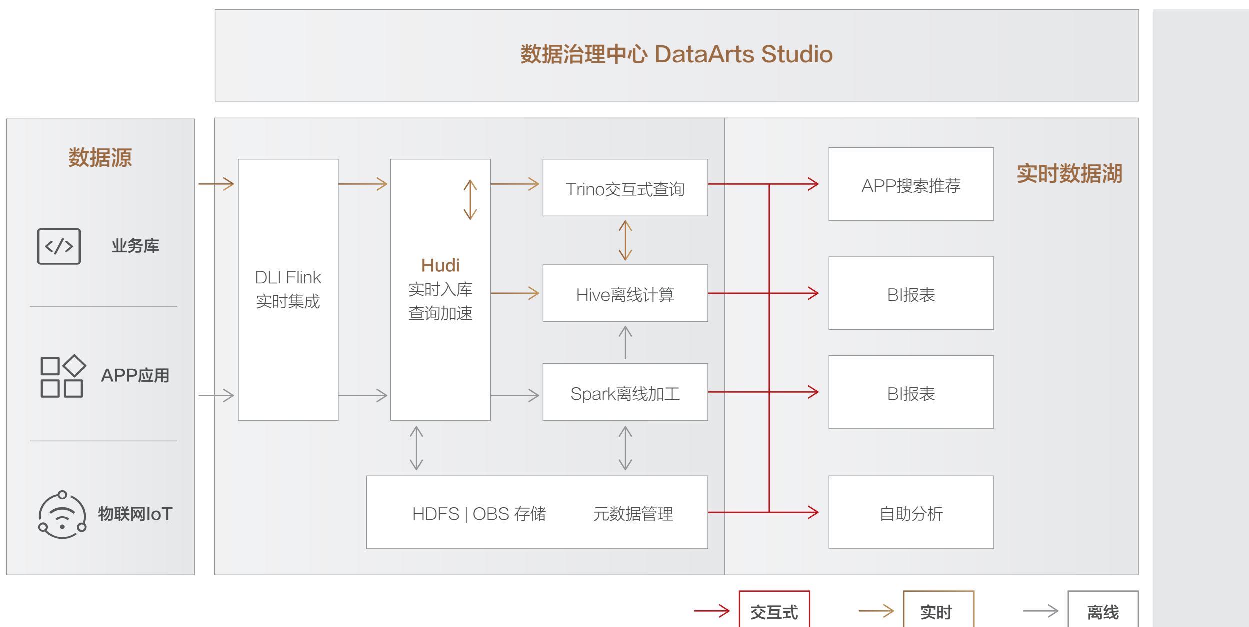
某互联网行业App拥有千万级日活用户，平均每天产生约500亿条用户行为数据，一年的数据存储量约为10PB。该用户最近使用Hudi构建全新的数据湖架构，借助“DB数据同步”、“实时数据入库”和“高性能查询和分析”等特性，轻松实现PB级别、甚至EB级别大数据分析处理能力。

事实上，在使用Hudi改造之前，该用户也曾在众多技术选择中做过对比分析和权衡：

传统 NoSQL 数据库虽然具有较好的数据索引机制，但是“太贵”。

因为要查询响应快速，用户通常会首先考虑HBase、ElasticSearch等自带索引的NoSQL数据库。以HBase为例，在存储方面，每1PB数据的云硬盘存储成本为70万/月；在计算方面，由于单台RegionServer支持的数据量上限是10TB，因此1PB的数据需要100台RegionServer，按照每台成本为500元/月计算，RegionServer的硬件成本为500元/每月/每台*100台=5万/月。所以，当采用HBase等NoSQL数据库，1PB的总成本为70万/月+5万/月=75万/月。

图表 1 基于 Hudi 的数据湖架构



“对象存储+文件”虽然具有较好的成本优势，但是“太慢”。

使用Parquet、ORC等列存，可以将数据存储在对象存储中，大大降低成本。这种方式下，1PB数据的对象存储成本约为8万/月；由于100台RegionServer计算节点可以按需启停，按照每天工作8小时计算，硬件成本为5万/月*0.33=1.67万/月。所以，每PB总成本约9.67万/月。相比上一种方式，成本大幅下降。但是，由于无索引，用户只能通过暴力扫描的方式进行查询和计算。在这种情况下，系统往往受限于对象存储的带宽，假设对象存储带宽为20GB/s，对10PB全量数据查询一次通常需要4~5个小时（视业务查询条件而定）。

“对象存储+Hudi”让性能与成本优势兼得，诠释“物美价廉”

Hudi兼具NoSQL的索引性能优势和Parquet、ORC等文件存储的成本优势，又快又便宜。首先，用户可以利用Hudi的索引、缓存等查询优化技术，使查询时间从4个小时下降到30秒内，查询性能提升480倍；其次，由于Hudi 支持ACID（Atomicity, Consistency, Isolation, Durability）事务和DB数据同步能力，使数据入湖可见周期从T+1缩短到T+0；第三，Hudi 基于存算分离架构，使用对象存储+100计算节点可以按需启停，每PB总成本约9.67万/月，费用不到NoSQL模式的1/7。

Hudi 应用场景背后所反映的数据湖架构的不足之处

上述互联网用户大数据架构改造的例子，让我们看到了在数据重要性日益提高的形势下，数据湖架构所面临的新挑战。虽然，我们看到近年来越来越多的企业选择了以数据湖为中心构建大数据处理平台，利用其计算和存储分离的特征，既降低成本，又可以获得更好的系统可扩展性。但是，在应用过程中也存在如下问题：

无事务能力，数据实时入库难。

数据湖依赖对象存储，但对象存储一般都没有ACID事务能力，导致在此之上构建的Hive表格、Spark表格等不支持基于事务的数据入库，更不用说数据更新了。这个弊端极大制约了数据湖的使用场景，企业无法将不断变化的数据快速注入到数据湖内，常常需要在业务层做大量预处理后，才能进入数据湖做分析，处理时延往往在一天以上。

分析性能依赖于暴力扫描，即费资源又太慢

数据湖的对象存储方式使得数据分析必须采用暴力扫描来做，性能完全依靠对象存储自身的吞吐能力，而且只适用于ETL、批量计算等对时延不敏感的应用，无法支撑如秒级数据检索、时序数据分析等低时延分析的应用。

对象存储+Hudi构建新一代数据湖成为企业的技术选择

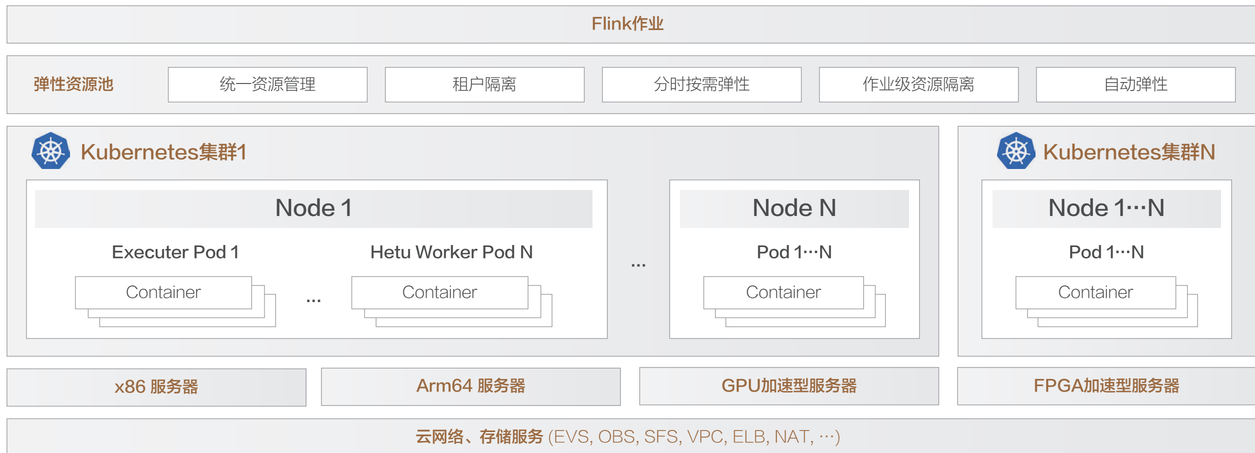
为了解决这些问题，业界出现了基于对象存储+Hudi构建的新一代数据湖，使云化数据湖可以真正成为企业数据架构的底座。

如图1所示，基于Hudi的新一代数据湖架构，Kafka完成数据收集后，由Flink、Spark Streaming等流计算引擎完成数据清洗、预处理等业务逻辑，将处理后的数据注入到Hudi表格中。然后，用户可以使用Spark、Hive、Presto等大数据引擎对表格中的数据进行交互分析、详单查询和ETL等业务。这种架构设计具备如下两大特点：

实时数据入库和 DB 数据同步。

如图2所示，基于Serverless的Flink，以云原生打造弹性流式入库新范式，Flink CDC 2.x无缝链接关系型数据库Hudi，使能数据实时入库。

图表 2 Serverless Flink



首先

云原生模式支持动态扩缩容，具备基于阈值和负载动态横向、纵向扩缩容能力，尤其面对周期性动态负载情况能极致降本。例如，车联网场景中白天一般属于负载高峰期，而晚上负载较低，利用弹性策略，可以白天分配较多算力，晚上缩减算力，某新能源车企有效降低成本40%以上。

其次

Flink支持作业优先级，限定资源下有效保障高SLO作业稳定性。面对部分作业负载洪峰，全局算力视图可以实现自适应算力分配和自由流动，提升作业稳定性。

最后

Flink支持同步MySQL Binlog，实时同步MySQL Binlog到Hudi数据实时集成，让关系型数据库和数据湖无缝实时流通。Hudi支持MOR格式增量同步，相比Hive使用的数据重写策略，数据同步性能提升10倍。基于Hudi的数据快速同步能力，企业可以轻松实现关系型数据库到数据湖的数据实时同步，数据入湖可见周期从T+1优化为T+0，消除数据入湖壁垒。

高性能查询和分析。

Hudi支持为对象存储的数据构建索引，实现10倍以上的查询性能提升。根据业务需求，用户可选择多种索引加速能力，包括主索引、二级索引（社区孵化中）、全文索引（社区孵化中）等。Hudi在构建这些索引的时同样遵循ACID事务性，确保索引构建过程中不会对业务查询造成影响，并可以利用云计算的按需扩展能力加速索引和物化视图的构建性能。基于Hudi最新版本的异步索引构建能力，在数据入库实时性要求较高的业务场景，用户可通过“先入库再建索引”的方式，平衡数据入库延迟和查询性能，实现数据入库后即可被查询，并使用周期任务或等到业务闲时再对数据建立索引，大幅提升查询性能。

未来展望

Apache Hudi是一个高性能、EB级别、原生Hadoop的分析型数据仓库，能提供海量数据的高性能明细和交互式查询、流数据接入和DB数据实时同步和更新、ETL业务的支持和加速，以及机器学习、深度学习等AI引擎的对接和优化等能力，这些都是在对数据的体量、实时性、价值释放要求越来越高的新形势下所必需的能力，所以，会有越来越多的开发者和企业使用Hudi，其生态发展也会相应的越来越完善。

数据领域难题(一):企业级本体与数据语义图谱构建技术

华为云数据智能创新Lab

前言

数据是企业的战略资产，是数字化转型的基础，但是数据要素价值的充分释放，还存在关键的堵点和难点问题。例如，“数据孤岛”仍是政府治理面临的挑战，在2022年，国务院和卫生健康委都分别提出加快推进“跨省通办”和健康码互通互认的要求；数据质量问题不降反增，根据毕马威的报告 ，在2021年，人民银行和银保监会开出的罚单中有805张与数据质量相关，占罚单总量的76%以上，位列处罚事由榜首；因为数据安全，在2022年，某打车软件被罚几十亿人民币……

问题背后也藏着机会，根据工信部测算，截止2021年，我国大数据产业规模达1.3万亿元，逐渐步入高质量发展阶段，到2025年，将突破3万亿元。身处这样一个蓬勃发展、体量巨大的市场，我们需要联合起来，解决关键核心技术短板，不断做大做强数据产业。因此，华为云数据智能创新Lab把在实际项目中遇到的、客户需求迫切的、价值含量高的挑战，以难题的方式公布出来，希望跟创原会的各位同仁一起解决。

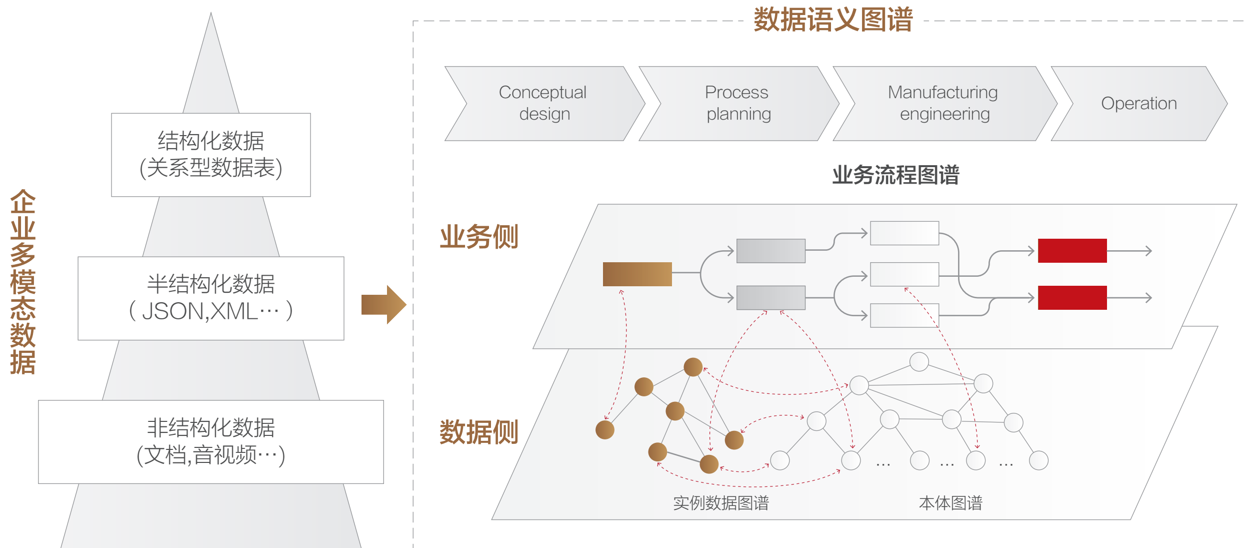
问题描述与价值

企业内部数据规模庞大且来源复杂，这些跨源、跨域、跨模态数据之间存在壁垒，“数据在，找不到”，无法有效挖掘数据内在语义价值，数据要素潜能难以释放。

企业本体图谱具有高保真、强表达、易推理等能力，是实现企业多模态数据价值沉淀、企业业务对象/业务流程建模、系统智能仿真、分析与决策的关键核心。尽管目前已有一些知识图谱构建技术，然而并不能很好地描述真实业务流程，且缺乏企业级多模态数据语义图谱的构建能力。

如何根据规模庞大、动态可变的企业多模态数据构建：本体图谱，驱动数据主动管理，构建业务对象；实例数据图谱，实现数据价值充分变现；业务流程图谱，分解业务信息，提升业务理解？

图表 1 数据语义图谱构建示意图



技术挑战

挑战一：动态本体图谱构建。企业业务日益复杂，数据种类日益增多，如何自动识别企业多模态中的业务对象，并构建自适应动态更新的本体图谱是一大难点。

挑战二：多模态实例数据图谱构建。企业源源不断变化的数据使得实例数据图谱构建需满足大规模、即时性等要求，以确保数据语义的正确性。现有技术未能有效捕获数据的动态特征，易导致灾难性遗忘。

挑战三：真实业务流程图谱构建。业务流程反应了企业实际业务活动，产生了跨系统、跨部门、多模态的数据，如何打破不同领域、不同部门之间的认知壁垒，构建基于统一认知框架的业务流程图谱极具挑战。

当前进展

本体图谱构建：业界强依赖于人工专家构建本体，通常只针对企业部分数据进行本体抽象建模，构建的本体模型无法反应企业全部业务对象的动态变化情况。

实例数据图谱构建：业界目前关注于构建静态图谱，图谱构建规模较小且缺乏对多模态数据的有效支持。此外，动态图谱构建技术尚在起步阶段，难以有效支撑真实且复杂的动态场景。

业务流程图谱构建：业界尚缺乏对于业务流程图谱构建的有效手段，存在较大的技术空白。

技术合作诉求

基于开放世界假设的多模态本体图谱自动构建技术：借助人在环路的ML/DL技术，从动态可变的企业多模态数据中准确识别并推理本体和关系，并构建标准化的多模态本体图谱。其中，图谱构建的关键路径——本体对齐的精确率不低于90%。

基于分布式计算的多模态实例数据图谱自动构建技术：实现基于Spark或Flink等分布式计算平台的亿级多模态实例数据图谱动态构建技术，在32cu环境下的图谱构建效率相比SOTA提升1-2个数量级，达成低时延、可扩展、抗遗忘的技术目标。

基于统一认知的业务流程图谱自动构建技术：基于业务流程识别业务对象，并构建业务对象在流程之中的关联关系（关联准确度不低于90%），反映企业活动，以便基于流程进行分析和决策。

参考文献

[1] Clarkson K, Gentile A L, Gruhl D, et al. User-centric ontology population[C]. European Semantic Web Conference. 2018: 112-127.
[2] Ilyas I F, Rekatsinas T, Konda V, et al. Saga: A Platform for Continuous Construction and Serving of Knowledge At Scale[C]. SIGMOD Conference. 2022: 2259-2272.
[3] Weikum G, Dong X L, Razniewski S, et al. Machine knowledge: Creation and curation of comprehensive knowledge bases[J]. Foundations and Trends® in Databases, 2021, 10(2-4): 108-490.

云原生 Cloud Native

行业趋势

“软”饭“硬”吃的计算 ——重识云原生系列第三篇 ---------------------45-50

Kappital 项目将加速云原生应用的服务化趋势 -------------------------- 51-56

FinOps 新探索--57-64

畅聊云原生：云原生应用上云与迁移实战 --------------------------------65-70

“软”饭“硬”吃的计算
——重识云原生系列第三篇

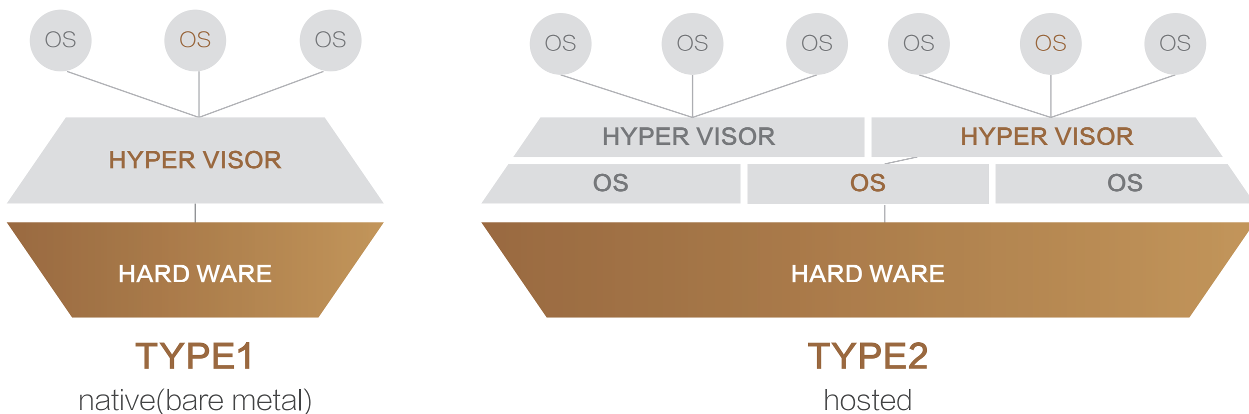
黄俊　招商证券云原生转型项目调研负责人

虚拟化技术定义

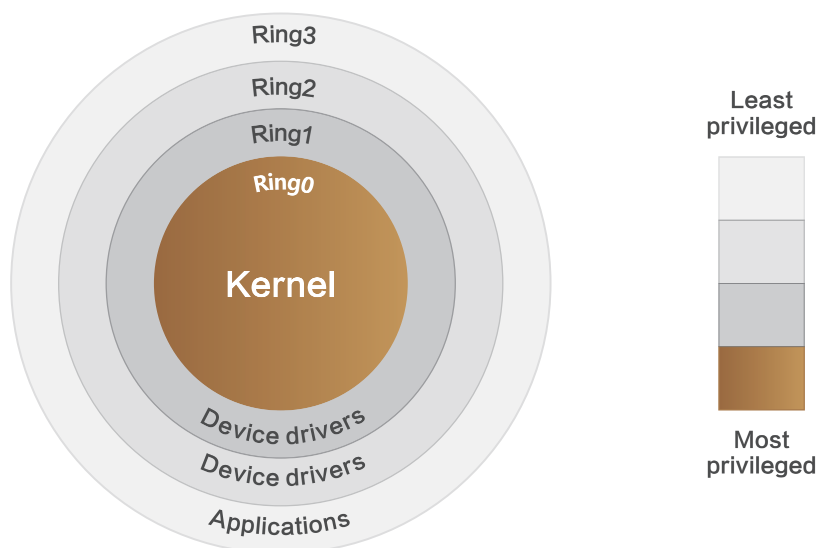
首先援引一段《虚拟化技术发展编年史》中针对虚拟化技术的定义：在计算机科学中，虚拟化技术（Virtualization）是一种资源管理（优化）技术，将计算机的各种物理资源（例如CPU、内存、磁盘空间，以及网络适配器等 I/O 设备）予以抽象、转换，然后呈现出一个可供分割并任意组合为一个或多个（虚拟）计算机的配置环境。虚拟化技术打破了计算机内部硬件实体结构不可分割的物理障碍，使用户能够以更灵活、更精细化的配置方式来使用硬件资源，即现下常提到的硬件变的软件“可定义”了。

其实，早在1974 年，Gerald J. Popek（杰拉尔德•J•波佩克）和 Robert P. Goldberg（罗伯特•P•戈德堡）在合作论文《Formal Requirements for Virtualizable Third Generation Architectures》中便已提出了一组称为虚拟化准则的充分条件，又称Popek and Goldberg virtualization requirements，其所述虚拟化系统结构有三个基本满足条件——资源控制（Resource Control）、等价性（Equivalence）、效率性（Efficiency），满足这些条件的控制程序才可以被称为虚拟机监控器（Virtual Machine Monitor，简称 VMM），并首次在此文中提出了两种Hypervisor类型，分别是类型 I虚拟机和类型II虚拟机：

类型II（寄居或托管型Hypervisor）虚拟机的VMM作为普通应用程序运行在主操作系统上，虽然有利于最大程度屏蔽底层硬件差异，但其运行效率也比类型I低，其排在前面是因为此类型VMM技术更早面世。市面上典型产品包括VMware 5.5以前版本、Xen 3.0以前版本、Virtual PC 2004等。



类型I（原生或裸机型Hypervisor）虚拟机的VMM则是直接运行在宿主机的硬件上，以此来控制硬件和管理客户机操作系统，此类虚拟机依赖硬件支持，故运行效率也更高，目前市面上主流产品均是此类，例如VMware 5.5及以后版本、Xen 3.0 及以后版本、Virtual PC 2005、KVM。



x86 体系虚拟化技术诞生背景

不过，在1999年VMware的x86虚拟化产品面世以前，虚拟化技术一直都只属于大型机这类高端玩家，PC机领域基本无缘。

当时，CPU为了保证程序代码执行的安全性、多用户的独立性以及操作系统的稳定性，提出了CPU执行状态的概念。它有效地限制了不同程序之间的数据访问能力，避免了非法的内存操作，同时也避免了应用程序误操作计算机物理设备。

Intel x86架构使用了4个级别来标明不同级别的执行状态权限。最高级别R0实际就是内核态，拥有最高权限，而一般应用程序处于R3状态（即用户态）。在权限约束上，高权限态应用可以阅读低权限态应用的运行内容，例如进程上下文、代码、数据等等，反之则不行。而当时处于一般应用程序等级的VMM，显然无法越级访问只有操作系统内核才能访问的内存、磁盘、鼠键等外围硬件设备中的数据。

而且，当时的Intel x86架构缺乏专门针对虚拟化技术的硬件支持，也就难以直接满足波佩克与戈德堡提出的虚拟化系统架构要求，所以从这两个方面而言，x86架构是一个天然不适合虚拟化的架构。具体而言，当时的x86架构CPU上有17条指令是虚拟化实现的直接阻碍，错误执行这些指令会导致操作系统显示警告、终止应用程序甚至完全崩溃。

VMware最先提供了此问题的解决方案——在虚拟机生成这些特殊指令时将它们“困住”，然后将它们转换成可虚拟化的安全指令，同时确保其他指令能不受干扰地执行。如此，便产生了一种与主机硬件匹配并能保证软件完全兼容的高性能虚拟机方案。

这就是第一代纯软件实现的全虚拟化（Full virtualization）方案诞生的背景，VMware首先完成了此方案的商业落地，即1999年推出的企业虚拟化产品VMaware Workstation，并由此一举奠定企业虚拟化解决方案服务商行业龙头的地位。

从此，虚拟化技术进入蓬勃发展期，不论是技术研究深度还是领域扩展广度都得到了大幅延伸，并由此直接推动了第一个云计算时代的到来（针对三个云计算时代的界定可参见笔者在第三期发表的《重识云原生系列第一篇——不谋全局不足以谋一域》）。

虚拟化技术发展简史

不过，全软虚拟方案毕竟受限于其纯软件实现的先天性方案缺陷，即需要由软件层的VMM完成大量指令转换，其实际运行性能以及宿主机资源利用效率一直备受诟病。于是人们开始将目光转向硬件，希望通过减少主要处理场景的指令转换、增加硬件直连的执行比例，由此提升虚拟机的整体运行性能。终于，在2003年，英国剑桥大学计算机实验室发布了开源虚拟化项目Xen，标志着虚拟化技术进入第二代半虚拟化（Para-virtualization）技术时代。

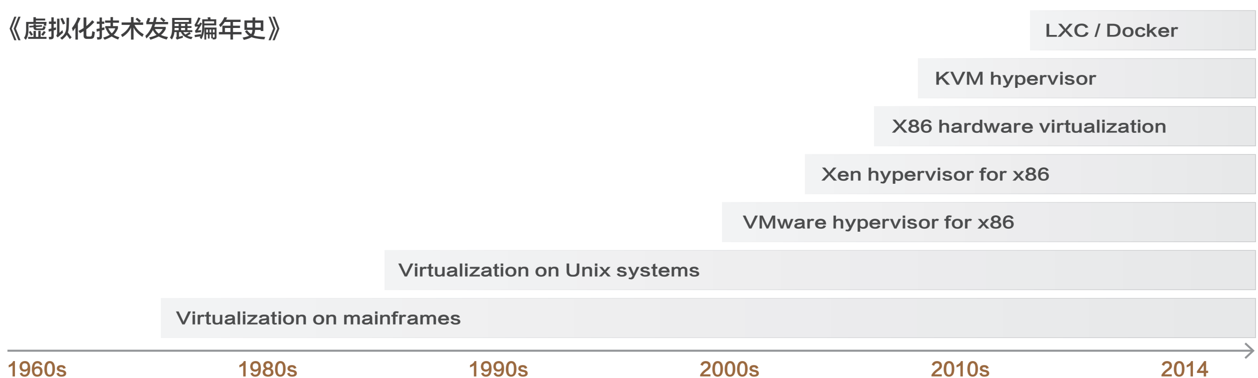
此后，随着微软、Oracle、Redhat、Intel、AMD、思科、惠普等软硬件头部企业的先后进场，虚拟化技术终于来到了基于硬件辅助虚拟（Hardware Virtual Machine，简称HVM）的第三代虚拟化技术时代，此标志便是2006年，Intel和AMD相继推出硬件辅助虚拟化方案Intel VT-x与AMD-V，通过引入新的指令和运行模式，省去了二进制指令转换环节，由此大幅提升虚拟机运行性能，顺便将虚拟化方案的演进方向由半虚拟化重新拉回到全虚拟化赛道上来（只是由纯软件虚拟变成了硬件辅助虚拟）。

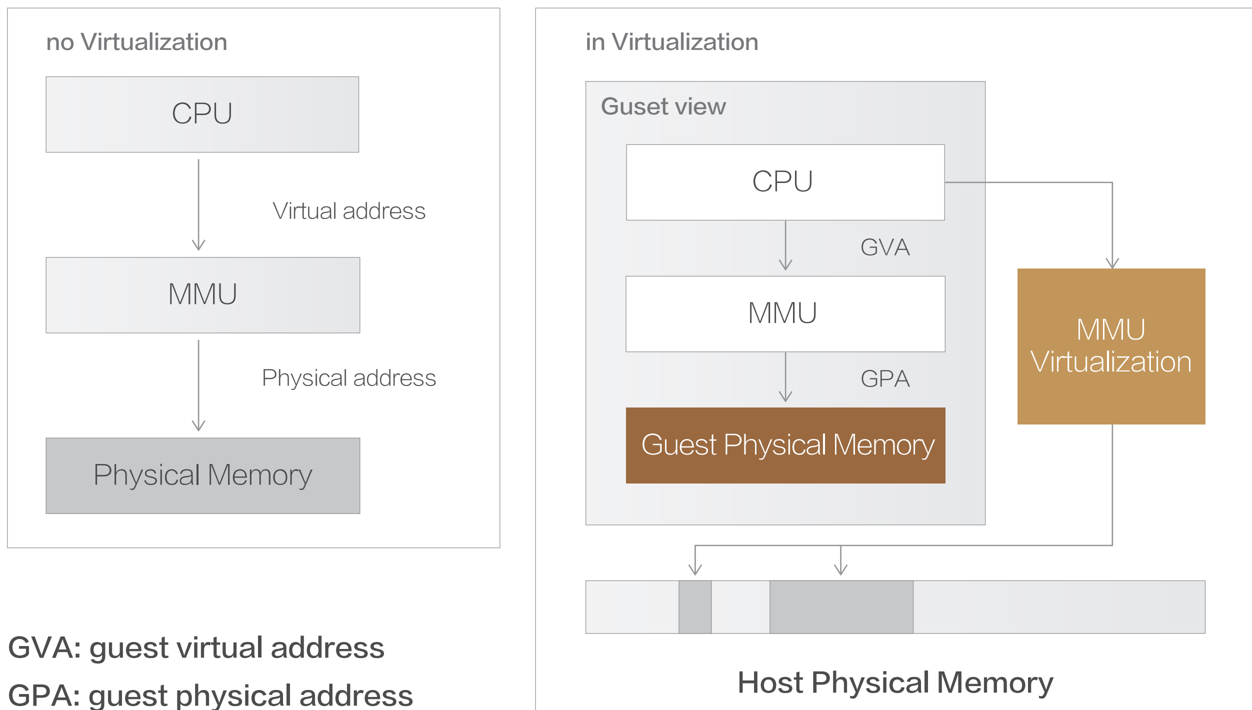
而到了2007 年，随着Linux Kernel 2.6.20合入虚拟化内核模块KVM（Kernel-based Virtual Machine，由以色列公司Qumranet开发），OS内核虚拟方案进入大众视野。凭借着Linux在企业服务器市场一统天下的地位以及免费开源的技术生态， KVM方案迅速成为当今主流云厂商计算虚拟化方案的事实标准，而KVM实现的前提也是CPU必须支持硬件辅助虚拟技术。

而就在大家以为KVM已经包打天下的时候，Docker的出现又似一次“时空折跃”，将虚拟化技术的竞争瞬间带进了云原生的赛道。

此图援引自

《虚拟化技术发展编年史》





从“软”到“硬”的虚拟化技术

虽然虚拟化技术就是为了将硬件变得“软件可定义化”，但技术方案的演进却是一个从“软”到“硬”的过程，接下来我们遵循冯•诺依曼计算机体系模型（即运算器、存储器、控制器、输入设备和输出设备五大部件组成的计算机系统），拆分来看各大部件的虚拟化演进之路，也可以看到类似的从“软”到“硬”的演进思路。

CPU虚拟化

在CPU虚拟化方面，上文其实已经详细阐述了，从VMare到Xen，再到KVM，其实主要就是CPU指令处理的虚拟化进程，我们很明显得可以看到，其经历的是全软件虚拟化（Full-virtualization）、半虚拟化（Para-virtualization）、硬件辅助虚拟化（Hardware Virtual Machine，简称HVM）三个过程。

内存虚拟化

对于内存而言，内存的地址映射与分页机制的实现本身就有类似虚拟化的技术实现——屏蔽底层物理内存地址的不连续性，以连续的虚拟逻辑地址对外（此处特指应用程序）提供服务。在非虚拟化环境中，虚拟逻辑地址与真实物理地址的映射与转换是通过CPU的MMU（memory management unit）管理单元来实现。

而在虚拟化环境中，由于客户机中的虚拟逻辑地址不能直接用于宿主机真实物理地址的MMU寻址，所以需要先把客户机虚拟逻辑地址GVA（Guest Virtual Address）转换为虚拟机物理地址GPA（Guest Physical Address），再把虚拟机物理地址转换成宿主机上的虚拟逻辑地址HVA（Host Virtual Address），然后运行在宿主机上的VMM再将宿主机虚拟地址HVA最终映射为真正宿主机物理地址HPA (Host Physical Address)，以供上层虚拟机使用。显然，此种映射方式，虚拟机的每次内存访问都需要VMM介入，并由软件进行多次地址转换，其效率是非常低的。

为了提高GVA到HPA转换的效率，先后诞生了两种直接转换的实现方案：第一种方案是影子页表（Shadow Page Table）技术，基于软件方式，实现客户机虚拟地址到宿主机物理地址之间的直接转换（KVM虚拟机支持）；第二种方案是基于硬件辅助虚拟化对MMU的支持来直接实现两者之间的转换,即Virtualation MMU方案，Intel的EPT(Extent Page Table) 技术和AMD的NPT(Nest Page Table) 技术均是此方案在两家CPU厂商的独立实现。很明显，有了CPU厂商的硬件辅助方案做性能加持，内存的虚拟化才具备真正商业落地的可能性。

I/O虚拟化

从处理器的角度看，外设是通过一组I/O资源来进行访问的，所以外设相关的虚拟化技术被统称为I/O虚拟化。
从下图可以看到，其实IO虚拟化也是经历了全软虚拟、半虚拟、硬件直通三个阶段：

第一代 模拟I/O方案

完全使用软件来模拟，VMM给Guest OS模拟出一个IO设备以及设备驱动，Guest OS要想使用IO设备需要先访问内核，然后通过驱动访问到VMM模拟的IO设备，最后到达VMM模拟设备区域，这是实现方式性能最低。VMware Workstation、Qemu均是此实现思路。

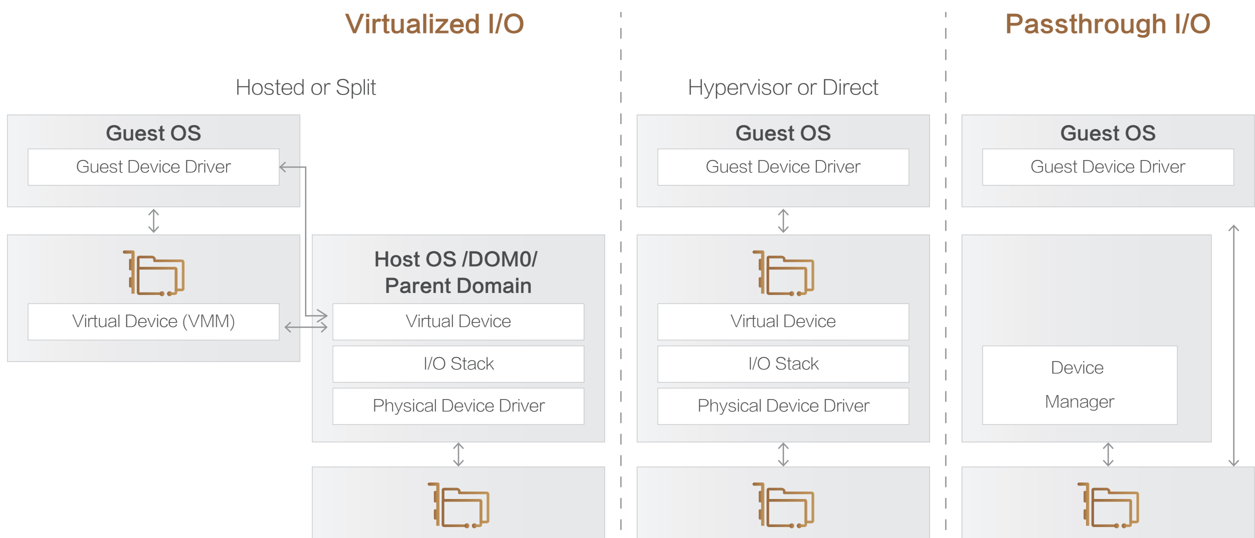
第二代 半虚拟化方案

半虚拟化比模拟性能要高，其通过系统调用直接使用I/O设备，与CPU半虚拟化方案类似，VMM给Guest OS提供了特定的驱动程序，使Guest OS自身的IO设备不需要处理IO请求，而是直接发给VMM进行处理，由此提升IO处理性能。典型实现例如Xen、Virtio。

第三种 I/O直通方案

I/O直通技术（I/O through）比模拟和半虚拟化性能都好，性能几乎等价于硬件设备直通，但灵活性较差。其实现思路就是提供多个物理I/O设备，如硬盘提供多块、网卡提供多个，然后规划好宿主机上运行的Guest OS数量，通过协调VMM来达到Guest OS与物理设备直接映射绑定的效果。

在当前主流云平台的实现方案中，半虚拟化方案与I/O直通方案因各有明显优劣势，呈一时瑜亮之势，可根据灵活性要求、性能要求选用。



后续技术演进展望

上文有提及，就在大家以为KVM已经一统江湖的时候，Docker的出现又将虚拟化技术的竞争带到了云原生的赛道。

Docker容器技术基于依赖自包含的运行标准化设计思想，使人们得以重新站在应用程序的视角（而不是资源视角）来审视虚拟化技术的初衷——构造一个尽可能与底层硬件无关的标准化运行环境，使得软件能一次打包、到处运行，但这个答案并不只有VM（只是在当年软硬件标准化程度相对较低的时代背景下，虚拟化技术成为了当时可见的最佳方案）。

然而云计算发展至今，软硬件厂商在被云“赶着”走上标准化之路之后，基于LXC、cGroup、Namespace等Linux内核隔离技术衍生出的容器技术立马就蜕变成了新时期的更佳解决方案，这是一种比传统虚拟机技术更加轻量、快捷、细粒度的资源虚拟方案。这也是继Linux打败Windows Server、KVM打败VMare之后，开源生态的又一次胜利。当然这也有赖于Linux庞大成熟的开源生态，才能如此迅速地推动容器技术标准的快速发展与成熟。

当然，需要指出的是，容器技术发展至今，容器（Container）玩家早已不仅仅只有Docker，还包括Kata container、Rocket container等等。

更进一步，从云计算的角度来说，不论是VMM还是Container都只解决了单机硬件资源的逻辑切分问题，如果要大规模商用，还需要一个大型、至少数据中心级的统一资源调度系统来确保巨量计算资源实例的快速、自动化编排，于是Kubenetes“适时”地出现了。其实，早期选项是Docker Swarm和Apache Mesos，是谷歌担心Docker发展太快、威胁到自己的云计算市场地位，于是将自己内部使用了近十年的编排工具Borg用Go语言重写并做了开源发布，一如当年Android之于智能手机市场，K8S一经推出便将Docker Swarm和Apache Mesos打的溃不成军，K8S一跃成为容器编排领域的行业事实标准。

由此，“Kubenetes+容器”的组合正式宣告了云计算的第三个时代——云原生时代的正式到来。

当然，从目前来看，容器技术依然处在快速发展的早期阶段，依然在重复“用软件方案解决一切问题”的早年虚拟化技术趟过的“老路”。不过，随着容器技术在公有云的大规模商用、大型企业的迅速普及，伴随着各类大规模、高性能场景的导入，其也开始走软硬结合的务实路线，以期通过底层硬件的卸载能力加持，抵消容器层网络、存储资源适配转发的性能损耗，以期早日实现全面替代虚拟机技术的目标。

于是，我们可以看到，大型云厂商的容器网络方案，无一不是直接与云平台的Overlay网络拉平，并通过智能网卡等技术来实现网络流量的硬件卸载，又是一幕“软饭硬吃”的案例重现。

所以说，计算虚拟化这碗看上去已经“够软”的饭，最终还是得来“硬吃”。

Kappital 项目将加速云原生应用的服务化趋势

朱玉华 华为云容器专家

1. 云原生应用的趋势

随着云原生的发展，企业的数字化转型已经从云化资源阶段，快速进入到云原生阶段。企业的关注点从以资源管理转向应用管理，开始考虑如何将云原生基础设施与业务应用融合，通过云原生使能应用的部署和运行，实现云原生应用的演进。在这过程中，云原生应用经历了以下三个阶段：

第一阶段：应用的标准化打包部署

第一阶段是应用打包和运行。Docker公司开源的Docker容器引擎，通过镜像和容器两个核心能力，解决了应用的打包和部署问题。标准化OCI镜像，让开发者将各种应用及应用依赖文件封装在镜像文件中，确保的应用的基础环境的一致性。同时通过容器引擎，可以在任何物理设备上运行容器，让应用彻底脱离底层设备，实现了一次打包，处处运行，完美解决了应用的打包和部署。容器技术打开了云原生的发展大门。

第二阶段：应用的自动化编排部署

第二阶段的核心问题是应用编排及如何部署和管理应用。Google开源的Kubernetes以良好的可扩展性、强大的社区，成为了容器编排的事实框架。通过Kubernetes的编排功能，用户可以构建跨多个容器的应用服务、跨集群调度容器、扩展这些容器，并长期持续管理它们的健康状况。Kubernetes解决了应用编排和调度问题，实现了应用的自动化部署和大规模部署。

与此同时，社区也在Kubernetes基础上，涌现出多个面向应用管理的优秀开源项目，帮助开发者更好管理应用，其中最为知名的是Helm、Operator-Framework、KubeVela等应用开源框架：

Helm

专注于Kubernetes包管理器，帮助开发者解决复杂应用的部署，通过HelmChart统一服务包格式，解决不同类型的应用统一部署，卡位Kubernetes应用复杂部署标准，是Artifact Hub上最大的应用类型，该框架也是开发者首选的云原生应用部署框架。

Operator-Framework

基于Kubernetes Operator技术，卡位有状态应用的开发标准，专注于实现有状态应用的安装和部署，解决了大量有状态应用实现了容器化部署，丰富了云原生应用。

KubeVela

专注于多云环境应用交付与管理平台，基于开放应用模型（OAM），实现应用的开发和运维的关注点分离。提出了应用定义的规范，实现对应用本身和应用所需的运维能力进行定义与描述。同时，通过声明式交付工作流能力，解决应用在多环境、多集群应用交付。

从上述开源项目看出，业界也在不断围绕应用部署、应用管理和应用运维，提供不同的解决方式，实现云原生应用编排和部署。

第三阶段：应用的服务化部署

在前两个阶段，业界的重点都在解决应用的标准化构建和自动化部署，以更好的方式部署应用。随着云原生进入到 “以应用为中心”的2.0时代，应用也迎来新的挑战：

1. 云原生应用的种类越来越多，从最初的Web应用、中间件应用，到如今的Serverless应用、大数据AI应用等。每种类型应用都有自身的部署形态和部署方式，应用管理方式也不尽相同。如何实现不同应用的统一打包和统一管理，是目前面临的挑战。

2. 产生了云原生应用的供给关系，以前开发者都聚焦如何开发应用，如何部署应用，完全依赖自身研发。如今随着越来越多的企业和开发者向社区开源应用，成为了应用的提供者。而另外部分的开发者直接从开源社区获取应用，成为了应用的使用者。这种供给关系极大促进了云原生应用生态，同时也给应用的双方带来挑战，例如应用的提供者如何描述应用的能力以便向社区推广，使用者如何准确获取符合自身诉求的应用。

因此，我们认为云原生应用将进入到服务化阶段，通过应用的服务化，描述不同类型的应用如何更好使用云原生技术，打通应用提供者和应用使用者的桥梁，实现应用的可管、可装、可维的全生命周期管理。

2. 什么是服务化

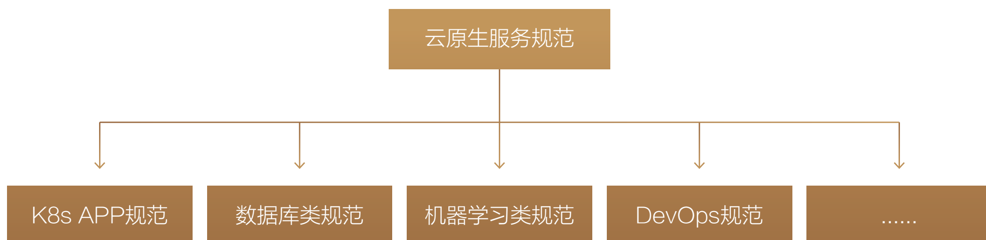
面对不同类型的云原生应用的部署和管理，统一的应用定义规范难以描述每种应用及其部署运行诉求，因此我们尝试从应用的服务化角度出发，将每种应用都作为一种服务，对外提供能力。我们提出了"云原生服务规范"，以云原生服务的角度来描述的云原生应用的能力，指导如何将应用进行服务化。同时，我们围绕云原生服务，提供服务的"全生命周期管理"。下面分别介绍一下云原生服务规范和全生命周期管理。

2.1 云原生服务规范

云原生服务规范旨在给出一种与云平台解耦的云原生服务的标准定义，以云原生服务定义不同应用在分布式云原生环境中的全生命周期管理，下面通过应用类型、应用部署、应用能力三个维度对服务规范进行介绍。

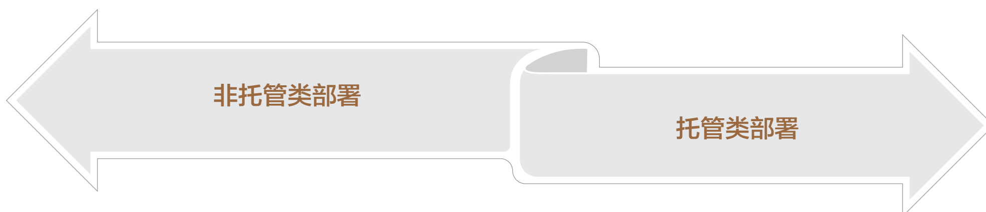
从应用类型维度

云原生服务规范涵盖了Kubernetes App应用，大数据类应用、Machine Leaning应用，以及DevOps应用等，服务规范将根据不同类型的应用，制定相应的接入和部署要求，指导开发者如何将应用改造，符合云原生服务化的要求。



从应用部署维度

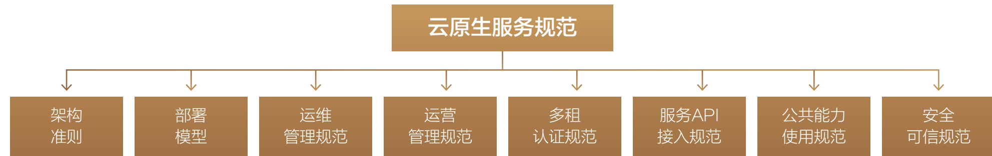
云原生服务规范描述两种应用部署方式：非托管类部署和托管类部署，支持不同类型的部署形态。
1.非托管类部署：指将应用部署到用户集群上，资源和实例均归属用户，用户负责管理和运维。
2.托管类部署：指服务提供商先将服务部署，然后以实例、API等方式，向用户提供服务能力，用户无需感知服务的部署，服务提供商负责整体服务的管理和运维。



从应用能力维度

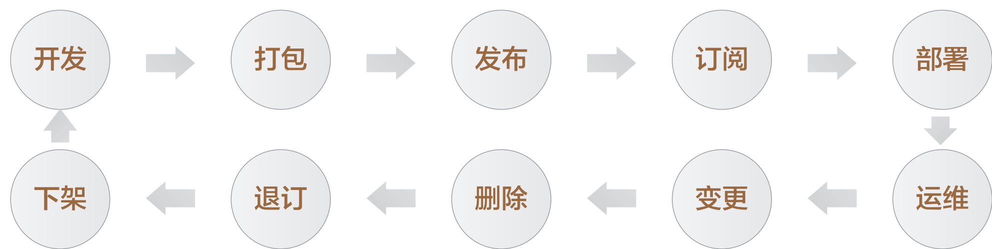
云原生服务规范总体范围包含以下8个方面：
1.架构准则：镜像支持OCI规范，应用支持微服务化。
2.部署模型：包括非托管方式部署（如web应用），托管方式部署（如RDS、数据库应用）。
3.运营管理规范：描述应用的计量、计费等运营策略。
4.运维管理规范：描述应用的监控指标、日志审计及日志管理等相关运维。
5.多租认证规范：描述应用支持逻辑多租、物理多租的认证方式。
6.服务API接入规范：描述应用的API接入及暴露规范
7.公共能力使用规范：描述应用使用第三方服务或者公共能力的规范，例如第三方数据库、缓

存服务、消息服务等。
8.安全可信规范：描述应用的安全、韧性和隐私保护

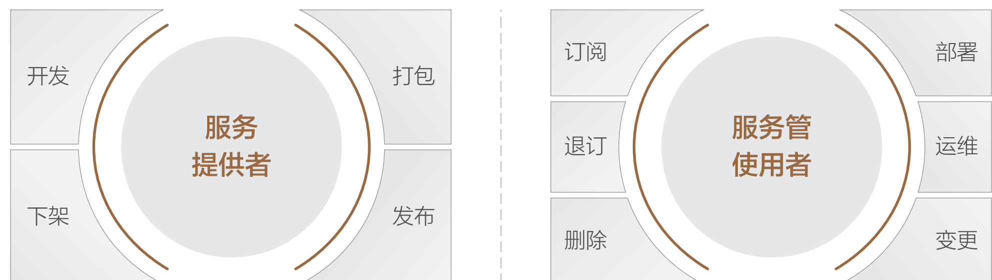


2.2 全生命周期管理

云原生应用的服务化是具备完整的全生命周期管理，而不仅仅包含开发和部署。当前业界开源框架主要考虑如何部署和运维，云原生服务的全生命周期包括以下核心步骤：



围绕云原生服务，我们还定义了2种角色：服务提供者和服务使用者，不同角色承担生命周期的不同阶段管理，实现了云原生服务的全生命周期管理。



3. 云原生服务Kappital

2022年6月华为伙伴暨开发者大会上，发布云原生服务Kappital开源项目，旨在帮助开发者把云原生应用进行服务化，快速构建跨云、跨边、跨集群的云原生应用。
Kappital是一个面向分布式云场景的云原生服务治理框架，基于云原生服务规范，为云原生应用提供统一的打包、部署、管理和运维，实现云原生服务的全生命周期管理。

3.1 核心概念

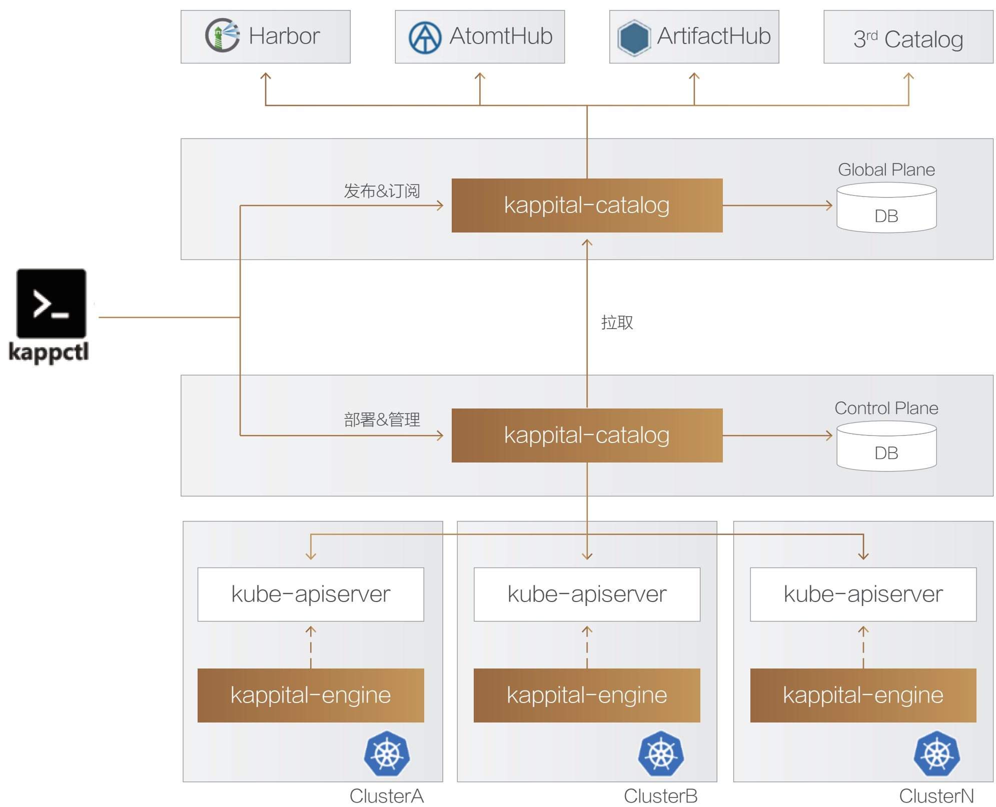
Kappital基于云原生服务规范，推出三个核心概念：

» **云原生服务(CloudNativeService)：**支持多云、云边等环境部署，使用云原生技术栈，具有可售、可装、可管、可维等的全生命周期管理能力的应用。

» **云原生服务包(CloudNativePackage)：**遵循云原生服务规范的服务制品，能够被托管在支持OCI格式的镜像平台中，如AtomHub、Harbor。

» **云原生服务实例(CloudNativeInstance)：**云原生服务部署到集群环境后，具有弹性可扩展云原生能力的服务实例，包含自身业务应用实例，平台增加的服务能力、日志监控等可观测性能力。

3.2 架构介绍



kappital-catalog Kappital服务制品管理组件

» 提供云原生服务上架、下架，订阅、退订

» 支持云原生服务的存储和解析，对接后端多制品源，包括Harbor、ArtifactHub、AtomHub以及第三方自建Catalog

» 支持服务分发到多集群、边缘集群等环境中

kappitalmanager: Kappital服务管理组件

» 提供云原生服务的部署、运维、变更、删除功能。

kappital-engine:

Kappital能力引擎组件

- 执行服务实例的创建和删除
- 执行服务实例的可观测性能力。

kappctl:

Kappital命令行工具，提供云原生服务的全生命周期的命令操作：

```
>>kappctl init --name={service_name}
>>kappctl build . -o {service_package.tar.gz}
>>kappctl push --pkg={service_package.tar.gz}
>>kappctl subscribe service {service_name}
>>kappctl deploy serviceinstance  {service_instance_name} --service={service_name}
>>kappctl get serviceinstance {service_instance_name}
>>kappctl upgrade serviceinstance {service_instance_name}
>>kappctl delete serviceinstance {service_instance_name}
>>kappctl unsubscribe service {service_name}
>>kappctl remove service {service_name}
```

3.3 服务包模板

Kappital提供服务包模板，作为云原生服务规范的载体。服务包给出了相关描述云原生服务的部署文件，规范用户开发服务包。

```
cloud-native-package /
├── metadata.yaml      # 元数据文件
├── manifests/         # 资源集合目录
├── capability/        # 平台能力目录
├── operator/          # Operator 目录
└── raw/               # 兼容第三方开源服务包
```

- metadata元数据文件：描述云原生服务详细信息，包括服务名称、类型、架构等
- capability目录：服务公共功能配置文件，如监控、日志等配置信息
- manifest目录：服务的crd文件、扩展csd文件
- operator目录：operator相关部署文件
- raw目录：存放三方开源服务包，如helmchart包。

4. 未来展望

我们希望通过Kappital开源项目，能够帮助开发者完成应用到云原生应用的改造，解决云原生应用的服务化过程中遇到的挑战，享受云原生技术给应用的赋能。未来Kappital项目将持续拓展“云原生服务规范”，使得该服务规范能引导用户构建成熟可扩展的云原生应用。同时，Kappital还将围绕云原生服务的“全生命周期管理”，提供服务开发套件，构建服务能力引擎，帮主用户更好的部署应用。我们也希望社区参与该项目的设计和开发，把Kappital打造为云原生服务的最佳实践平台，促进云原生应用生态。

FinOps 新探索

历川 华为云 Serverless 研发专家　**平山** 华为云中间件 Serverless 负责人　**冯嘉** 华为云中间件首席专家

1. 问题引言

Serverless 精确到毫秒级的按用付费模式使得用户不再需要为资源的空闲时间付费。然而，对于给定的某个应用函数，由于影响其计费成本的因素并不唯一，使得用户对函数运行期间的总计费进行精确的事先估计变成了一项困难的工作。

以传统云资源的周期性租赁模式为例，通过周期数乘以周期单价，用户可以很容易地估计出租赁期间的总费用，形成清晰的心理账户预期，即使在云平台采用阶梯定价或价格歧视策略的情形下，计算租赁总成本也不是一件难事。

但在 Serverless 场景中，事先估计函数总成本仍缺乏有效的理论指导。一方面，影响函数计费的关键因素不唯一，如包括函数内存规格、单实例并发度、函数执行时长等；另一方面，函数调用流量的波动通常具有随机性和非平稳性，使得基于流量的“按用计费”具有较大的不确定性。当然，寻找函数计费的理论指导主要是为用户评估函数总成本提供一种有效依据，但更加重要地，如何进一步利用估计模型，帮助用户优化应用函数及其配置选择，进而显著降低用户函数总成本，是 Serverless 领域中，FinOps 亟待回答的问题。

FinOps 聚焦云上资源管理和成本优化，通过有机链接技术、业务、和财务专业人士，来优化用户、企业、组织的云资源成本，提高云上业务的投入 - 产出比[1]。本文结合华为云 FunctionGraph 在 Serverless 领域的 FinOps 探索和实践，剖析 Serverless 场景下的函数计费模式和关键影响因素，介绍一种对函数运行期间总计费进行事先估计的模型框架；更重要地，该模型为帮助用户优化函数运行总成本、提升用户云上 Serverless 资源管理效能，实现经济型 (Economical) Serverless 提供有效依据。
首先对表 1 所列的几个概念做简要说明。

表 1: Serverless 函数常见名词

内存规格	Memory	MB
单实例最大并发度	Maximum Requests per Instance	/
函数执行时延	Function Execution Time	ms
单函数最大实例数	Maximum Instances per Function	/

内存规格 (Memory)

内存规格也即函数规格、函数实例规格，表示 Serverless 平台为函数的单个实例所分配的资源大小，一般表示为函数可使用的内存大小，由用户指定；实例可使用的 CPU 份额与内存大小成正比。

函数执行时延 (Function Execution Time)

这里指完成一次调用请求响应的过程中，函数本身执行所消耗的时间，主要由函数代码逻辑决定。一般地，对于 CPU 密集型的函数，增大函数资源规格（内存 -CPU Share），可以显著降低函数执行时延。但对于消耗大部分时间在网络 IO 等操作上的函数，增大资源规格对执行时延的改善则非常有限。

单实例最大并发度 (Maximum Requests per Instance)

函数的单个实例可以同时处理的最大请求数，主要适用于函数执行过程中有显著时间在等待下游服务返回的场景，如访问数据库操作或磁盘 IO 等。对于相同的流量负载，提高函数的单实例并发度可以降低按量实例个数，为用户节省计费，同时，也可以降低函数调用请求的冷启动比例。

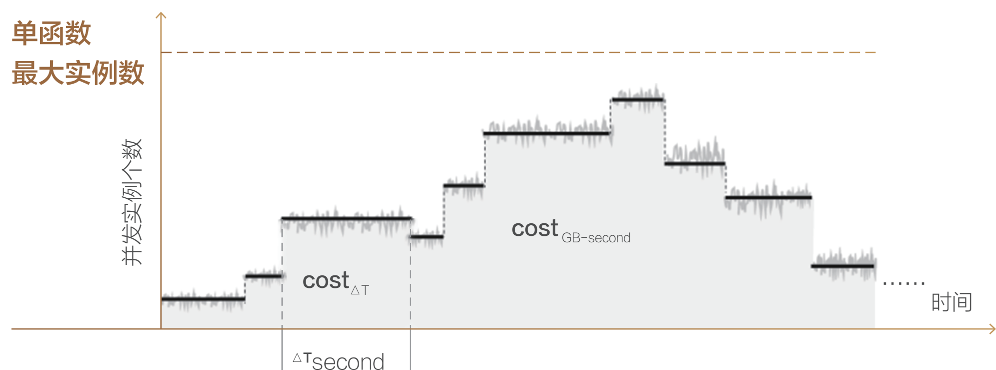
单函数最大实例数 (Maximum Instances per Function)

指同一函数同一时刻下同时运行的实例数上限。对用户来说，最大实例数可以防止异常流量洪峰下或函数发生故障时由于云平台的过度扩容而导致的费用失控；对云平台来说，最大实例数可以防止异常情况下平台资源被部分函数耗光，从而保障不同函数间的性能隔离。

2. 函数计费与成本模型

单实例视角下的函数计费估计模型，可参考[2]。在真实生产环境中，除异步函数外，Serverless 云平台通常采用 FCFS（First Come First Serve）的方式响应调用请求，对于函数流量的潮汐波动，平台通过自动扩缩容实例进行自适应，系统中运行的并发实例数随时间的变化，可以由一个分段常线性函数完全刻画，如图 2 所示。

图 2：函数并发实例数随扩缩容过程的变化



尽管不同 Serverless 云厂商之间的计费方法存在差异，函数计费一般主要包括两部分：对函数所使用资源的计费以及对请求次数的计费，表示如下：

$$TotalCost = Cost_{GB\text{-}second} + Cost_{TotalRequest},$$

其中，表示对资源使用的计费，单位为 GB- 秒（GB-second），表示对调用次数的计费。

为方便计算 TotalCost，用m_{GB}表示函数的资源规格，单位为 GB，例如，对于 128MB 规格的函数，其$m_{GB} = 128/1024$；c 表示该函数的单实例并发数，μ 表示函数的平均执行时延，单位为毫秒；并用 α (0<α<1) 表示 Serverless 平台的调用链路性能，在最理想的情况下，该指标为 1，表示在当前 Serverless 平台上，该函数响应单个请求的端到端时延等于函数执行时延 μ 本身，不同 Serverless 平台的 α 值可能略有不同，但通常在 0.9 以上。给定上述指标，可以得到单实例在理想状况下的请求处理能力，即理论上每秒可以响应的调用次数为：

$$\frac{c}{\mu}\ (request/s)$$

因此，单实例的实际请求处理能力则为：

$$\frac{c}{\mu}\alpha\ \ (request/s)$$

我们以一个月作为估计周期。假设一个月内，函数共经历了 n 次扩、缩容，形成了 n 个常线性子区间（如图 2 所示）。先考察单个子区间 ΔT_{second} 内的计费成本模型，总成本模型则为各个连续子区间的加和。

在时间窗口 ΔT_{second} 内，假设函数调用次数为 $Q_{\Delta T}$，则该时间窗内的并发实例数为：

$$\frac{Q_{\Delta T}\cdot\mu}{c\alpha\Delta T_{second}}$$

对应的资源计费部分则可表示为：

$$\text{Cost}_{\Delta T_{second}} = p_{\text{GB-second}}\cdot\frac{Q_{\Delta T_{second}}\cdot\mu}{c\alpha\Delta T_{second}}\cdot\Delta T_{second}\cdot m_{GB}$$

其中，$p_{\text{GB-second}}$ 表示每 GB- 秒的资源的计费单价。现在，记第 i 个子区间为 ΔT_i，则一个月内的总成本模型可以估计为：

$$\begin{aligned}
\text{TotalCost} &= \text{Cost}_{\text{GB-second}} + \text{Cost}_{\text{TotalRequest}}\\
&= \sum_{i=1}^{n}\text{Cost}_{\Delta T_i} + p_{Request}\cdot\left(\sum_{i=1}^{n}Q_{\Delta T_i} - Q_{free}\right)\\
&= p_{\text{GB-second}}\cdot\left(\sum_{i=1}^{n}\frac{Q_{\Delta T_i}\cdot\mu}{c\alpha\Delta T_i}\cdot\Delta T_i\cdot m_{GB} - \text{Cost}_{GB\text{-}second\text{-}free}\right) + p_{Request}\cdot\left(\sum_{i=1}^{n}Q_{\Delta T_i} - Q_{free}\right)\\
&= p_{\text{GB-second}}\cdot\left(\sum_{i=1}^{n}\frac{Q_{\Delta T_i}\cdot\mu}{c\alpha}\cdot m_{GB} - \text{Cost}_{GB\text{-}second\text{-}free}\right) + p_{Request}\cdot(Q - Q_{free})
\end{aligned}$$

其 中 ， $p_{Request}$ 表 示 每 次 调 用 的 计 费 单 价 ， $Q=\sum_{i=1}^{n}Q_{\Delta T_i}$ 表 示 函 数 该 月 总 流 量 ，$\text{Cost}_{GB\text{-}second\text{-}free}$为云平台提供的月度免费计量时间， 为月度免费计量调用次数。

在上式中，单实例并发度 c 和函数规格mGB可以认为在用户配置之后属于常数；α 属于平台侧参数，也可视作常数；对于函数执行时延 μ，实际中通常会由于冷热启动差异、网络抖动、调用请求入参等的不同而波动，且考虑到 Serverless 计费是精确到毫秒级别的，因此严格意义上不能被视作为常数。不过，作为估计模型，这里暂且假定 μ 也为常数，综上，总成本模型可以表示为：

$$\begin{aligned}
\text{TotalCost} &= \left(p_{\text{GB-second}}\cdot\frac{m_{GB}\cdot\mu}{c\alpha} + p_{Request}\right)\cdot Q\\
&\quad - \left(p_{\text{GB-second}}\cdot\text{Cost}_{GB\text{-}second\text{-}free} + p_{Request}\cdot Q_{free}\right) \qquad (1)
\end{aligned}$$

后半部分代表云平台提供的免计费总量，与函数调用流量以及函数配置无关。

3 成本优化方法讨论

有了函数成本的估计模型，就可以对影响用户成本的关键因素进行讨论。在估计式（1）中，忽略云平台提供的免计费总量，函数月度总成本的结构如下：

$$p_{\text{GB-second}} \cdot Q \cdot \frac{m_{GB} \cdot \mu}{c\alpha} + p_{Request} \cdot Q \qquad (2)$$

Point 1:
优化函数代码逻辑本身，降低函数执行时延

对于同样的函数流量负载，更低的执行时延　可以为用户节省更多计费成本。在用户业务逻辑允许的前提下，不断优化函数代码、提高函数执行效率是软件工程本身天然的诉求，但在 Serverless 场景下，这一点显得更为迫切。

具体地，考虑采用 Python、Nodejs 等轻量化编程语言，减少函数初始化配置中的非必要项，将连接其它服务如数据库等的操作尽量移到函数执行入口之前的初始化阶段完成，简化代码逻辑等。

Point 2:
优化函数代码包、依赖包、镜像大小

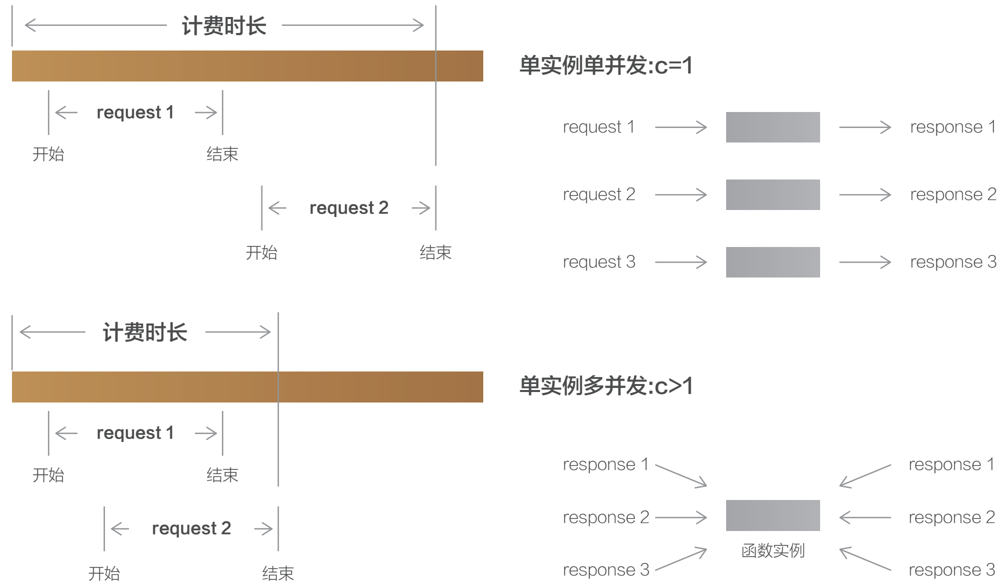
当函数调用触发冷启动的时候，从计费角度看，冷启动时延包含在执行时延 μ 中一起计费，而冷启动中有相当比例的时延消耗在云平台从第三方存储服务（如华为云对象存储服务 OBS）中下载用户的代码包、依赖包，或从镜像仓库服务中拉取用户应用镜像，如图 4 所示。尽管为了优化冷启动性能，目前大部分云平台均会采用各类缓存机制，对用户代码和镜像进行预缓存，但实例启动中消耗在用户代码加载上的时延仍然十分显著。因此，应尽可能优化函数代码包大小，包括对依赖包、镜像等进行瘦身，进而降低计费时长。

Point 3:
编写功能聚焦的轻量化函数

在 Serverless 编程框架下，尽可能将函数编写为轻量型的、功能聚焦的程序代码，即“functions should be small and purpose-built” [3]；让“一个函数只做一件事”，一方面，功能单一的函数，运行时延也更容易针对性地进行优化；另一方面，当一个函数内同时实现多个功能的时候，大概率会以所有功能都在性能上同时做出妥协为结果，最终提高了函数运行期间总计费。若应用函数的确需要提供多个功能，可以考虑将大函数分解为多个小函数，然后通过函数编排的方式实现整体逻辑,大函数分解也是 Serverless 计算中用户处理超时（timeout）等异常场景的最佳实践之一[4]。

图 4：冷热启动下的计费时长及优化点

图 6:
单实例并发度:
计费时长视角和
实例数视角



Point 4:
业务模型支持的前提下，采用单实例多并发

从公式（2）的函数成本结构中可以看出，在用户业务模型支持的前提下，配置一定的单实例并发度 ，可以有效降低函数月度总成本；若用户不进行配置，云平台默认值通常为 1，即单个实例同一时刻只能处理一个请求；因此，在函数被并发调用的情形下，平台会启动多个实例进行响应，从而增大了计费实例数目，如图 6 所示；同时，采用单实例多并发，也能改善调用请求处于等待状态的尾时延。

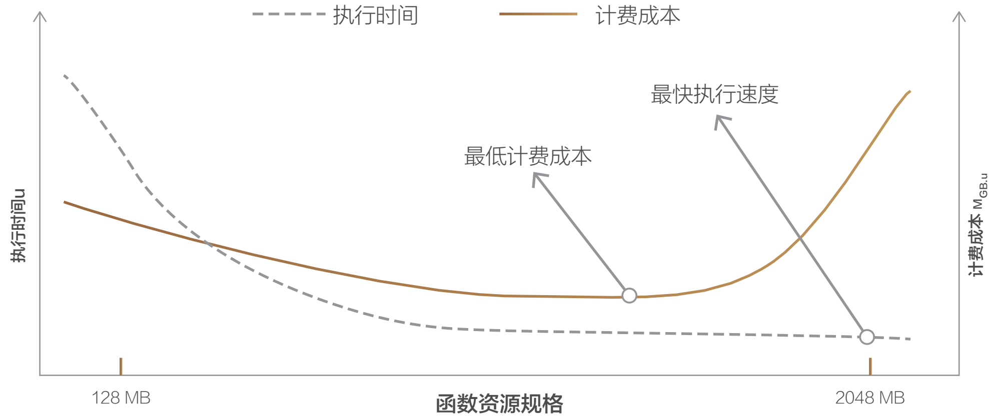
当然，单实例并发度并非越高越好，例如，过高的并发度设置会使得函数实例内多线程之间的资源竞争加剧（e.g., CPU contention），导致函数响应性能恶化，影响用户应用的 QoS 指标等。同时，如本文在背景知识中所提，并非所有的应用函数都适合设置单实例多并发。单实例多并发主要适用于函数执行过程中有相当比例的时延消耗在等待下游服务返回的场景，这类场景下，实例资源如 CPU 等有显著比例处于空闲等待状态，如访问数据库、消息队列等中间件、或磁盘 IO、网络 IO 等。单实例多并发也需要用户在函数代码中对错误捕获（e.g., 考虑请求级别的错误捕获粒度）和全局共享变量的线程安全（e.g., 加锁保护）问题进行适配。

Point 5:
函数资源规格的选择需考虑对执行时延的影响

最后讨论函数资源规格的选择问题。从公式（2）明显可以看出，更大规格的实例内存 对应更高的计费成本。但内存规格的选择，需要同时考虑对函数执行时延 的影响。从用户函数的角度看，函数执行时延除了由代码本身的业务逻辑决定之外，还受实例运行时可使用资源大小的影响。更大的实例规格，对应更大的可使用内存和更多的 CPU 份额，从而可能显著改善高内存占用型或 CPU 密集型函数的执行性能，降低执行时延；当然，这种改善也存在上限，超过某个资源规格后，资源的增加对降低函数执行时延的效果几乎可以忽略，如图 7 中虚线所表示的过程。上述事实表明，对于给定的用户函数，为降低总计费成本，需要配置合理的实例规格mGB，使得mGB•μ 尽可能取得最小值，如图 7 中实线所表示的过程。

图 7:

函数规格的选择需同时考虑对成本和执行时延的影响



例如，考虑实例规格的初始配置为 m_{GB}^{0}（例如从最小规格开始，i.e., 128MB），经测试该规格下函数执行时延为 μ^{0}，则可以得到基线 $m_{GB}^{0} \cdot \mu^{0}$，然后逐步增大资源规格，测试对应执行时延，直到某一组 (m_{GB}^{i}, μ^{i}) 出现，使得：

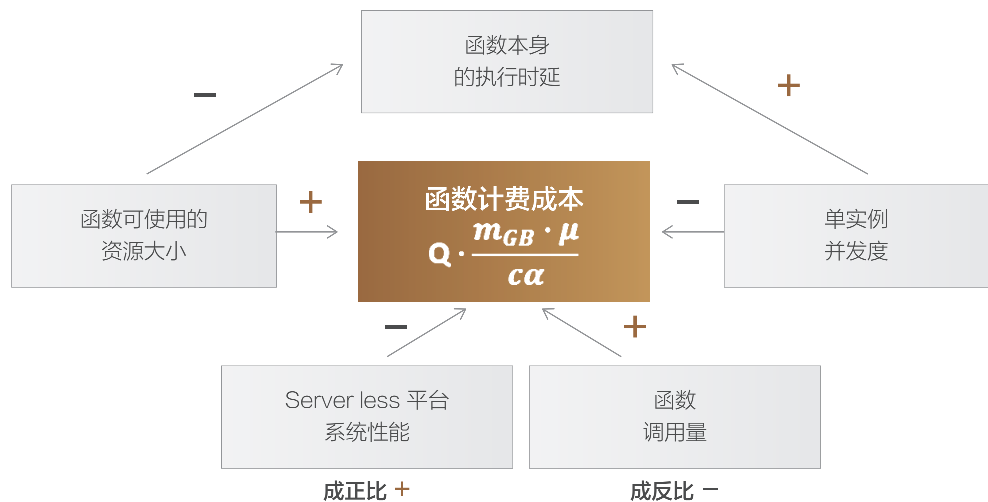
$$\frac{m_{GB}^{i+1}}{m_{GB}^{i}} \geq \frac{\mu^{i}}{\mu^{i+1}}$$

此时表明，资源增大对计费成本的边际提升已经超过了对执行时延的边际改善，因此，从成本的角度看，此时的 m_{GB}^{i} 为帕累托最优解，即最佳规格，对应执行时延为 μ^{i}。

最后，图 8 对上述几个决定函数成本的关键因素做了一个总结，其中，箭头方向表示元素之间的直接影响，"+" 号代表成正比，"-" 代表成反比。

图 8:

函数计费成本的关键因素分析

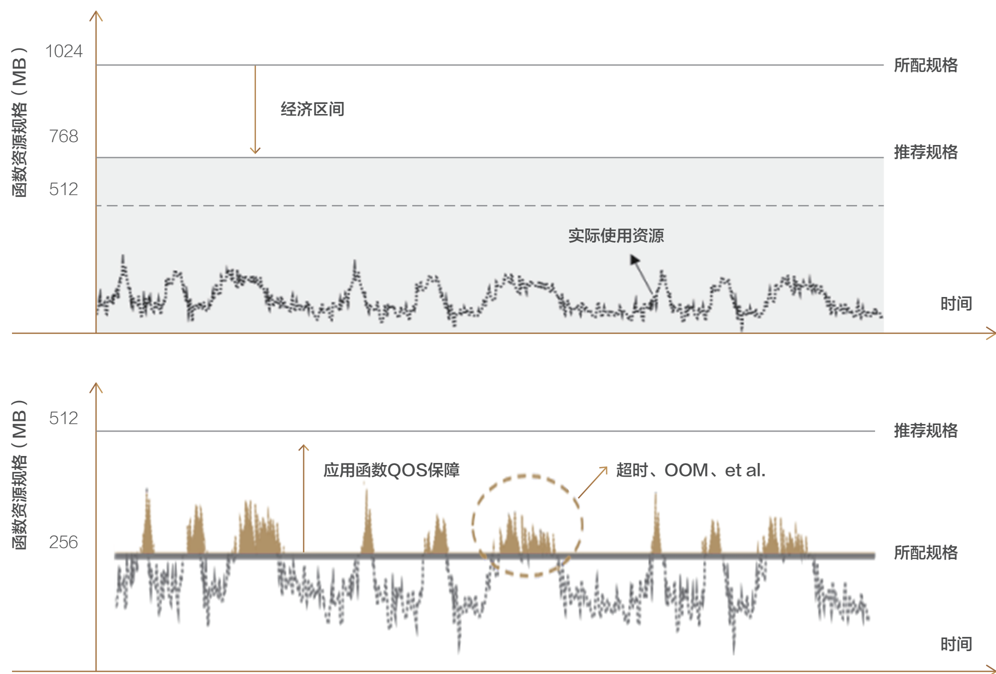


4. Serverless 函数成本研究中心

为用户降本增效，是 FunctionGraph 的核心理念。尽管前文分析的五种函数成本优化手段是站在用户视角下的讨论，但我们认为这些问题远不是只属于用户需要考虑的范围；相反地，FunctionGraph 在持续探索如何最大限度地帮助用户在 Serverless 领域实现最佳的 FinOps 效果，让用户能够真正享受到 Economical Serverless 的福利；例如，在实例级别的深度可视化、可观测性前提下，帮助用户实现函数 FinOps 全流程的自动化，为用户提供透明、高效、一键式的函数资源管理和成本优化服务。

图 9：

在线式资源消耗感知与规格动态推荐



5 总结与展望

本文主要讨论了 Serverless 计算场景下的 FinOps 问题，给出了业界首个用户函数总成本估计模型，并根据该模型，为用户优化应用函数、提升 Serverless 资源管理效能、降低总成本提供理论参考和实践依据。

一项新兴技术领域的兴起，首先需要回答的问题是“Why & Value”， FunctionGraph 作为华为元戎加持的下一代 Serverless 函数计算与编排服务，结合 FinOps 等技术理念，持续为用户提供经济型 Serverless 服务。

参考资料

[1] What is FinOps: https://www.finops.org/introduction/what-is-finops/

[2] Running Lambda Functions Faster and Cheaper: https://levelup.gitconnected.com/running-lambda-functions-faster-and-cheaper-416260fbc375?gi=4370e4c57684

[3] AWS Lambda Cost Optimizations Strategies That Work. https://dashbird.io/blog/aws-lambda-cost-optimization-strategies/

[4] Timeout Best Practices. https://lumigo.io/learn/aws-lambda-timeout-best-practices/

畅聊云原生：云原生应用上云与迁移实战

前言

常说管理者是团队的天花板，在推动团队、公司的发展过程中，我们躬身入局、身先士卒、披荆斩棘，然而，有时我们也需要停下来，与志同道合的人一起聊聊我们的‘心理话’。创原会，为志同道合的技术管理者们提供一个可以畅所欲言的交流平台。2022年第七期”畅聊云原生”联合陌陌云基础设施部门云平台团队负责人周峰先生、B站技术委员会主席毛剑先生、华为云容器服务首席架构师张琦先生共同出品，聚焦云原生应用上云与迁移话题开展研讨和剖析。

畅谈云原生应用挑战

企业全业务上云不仅是一个技术问题，同时也是一场变革，涉及到企业应用整体治理体系的变化、企业组织架构的适配和思维方式的塑造、项目实施的管理以及持续的运营运维优化等。这对企业全业务上云提出了很大挑战，企业迫切需要一套覆盖应用治理、方案实施等全面的方法论及技术体系来支撑。

如何快速云原生化践行企业可持续发展

周峰先生开启话题，他指出企业数字化转型是一个集业务、组织、技术与变革管理为一体的综合工程。在数字化转型和互联网化的冲击之下，企业面对的用户需求更加多样化、个性化、碎片化，企业应用上线、迭代速度加快，企业希望应用功能、服务都能快速交付给用户，快速产生市场价值。可持续发展的目标能够改进企业运营效率、敏捷性和驱动业务成果，成为新的企业增长策略，改进企业的运营和成本。企业实践可持续发展战略方面，数字技术和数字化转型正在成为必不可少的支柱和势能技术，云原生与大数据、人工智能、物联网、区块链等数字技术已经融合到企业的方方面面。云原生作为企业数字化转型的底座，安全的混合云，它的灵活性和敏捷性，使企业能够开发与伙伴合作，并充分开放技术潜力推动创新。

周峰
陌陌云基础设施团队
虚拟研发负责人

软通动力CTO刘会福表示，企业业务云原生化，涉及以下几个步骤：

（1）业务代码镜像化改造

（2）应用打包，将依赖和应用可以在容器化环境里快速部署

（3）编排和调度，通过k8s等实现应用集群统一编排和调度

刘会福
软通动力CTO

（4）应用赋能，应用接入后，可以利用云原生的能力（比如服务注册，限流，熔断，负载均衡，监控告警，链路追踪等）来更好的管理应用

周峰先生指出："总的来说，企业云原生化可以从二个方面展开考虑，传统技术架构微服务化和建设云原生能力评价体系，具体而言企业通过微服务轻便和自给自足的容器，应用采用自动化测试工具、持续集成和自动化部署工具来帮助团队开发和管理众多服务，形成微服务模式，从而逐步取代旧的系统，同时，基于云原生能力评价体系围绕云原生应用应该具备的松耦合、易于管理、可观察性、容错性好的四大特性。"

马达
英伟达高级技术经理

英伟达高级技术经理马达补充道："企业数据化转型不仅需要技术上的支撑及演进，对组织架构，流程规划也需要有相应的转型；可以更快的将云原生技术应用到企业中，核心系统经常有很强的业务属性，需要相应的专家与架构师不仅有较强的技术背景，还要精通业务。需要多方合作才能达成目标。"

Donald
CNCF开发者布道者

来自CNCF开发者布道者Donald进一步就开源在企业云原生实践中的价值做了说明，云原生生态中不仅有微服务，还有基础设施，可观测性等；传统应用在不进行微服务改造的情况下，可以考虑借助云原生基础设施等能力进行升级。

云原生改造过程中如何应对开发/部署/商业环境变化

在探讨云原生改造应对环境变化时，来自周峰先生表达了他的观点："企业在数字化转型进程中从底层基础设施、技术架构、研运管理、统一治理等视角出发，构建自下而上的完整敏捷链路，从而赋予企业实时洞察与快速响应个性化、场景化、定制化需求的能力。我们可以从开发，部署，商业环境角度出发开发技术匹配业务需求，在保证稳定性和可持续性的情况下，选用较新、较灵活和成熟的技术架构，通过部署多云、多region、多AZ的高可靠是设计理念，实现架构高可用，选用云厂商基于主流开源软件和架构的商业化产品，满足市场对商业环境的变化。"

赵旭华
华人运通云服务和
信息安全总监

华人运通云服务和信息安全总监赵旭华补充道："云原生最大的挑战自于传统应用架构的惯性思维，对于应用的云原生迁移，部署以及不同环境的配置落地策略，从新功能小模块的快速迭代，结合网关，统一身份认证等手段，在不影响用户体验的情况下小步快走实施。另外对业务层面，云原生转型的价值是需要规模效应来体现的，大量实践已经证明无论从应用交付效率，到服务

器资源利用率，还是监控，运维，问题排查的质量和效率都会带来很大受益。从实践来看，彻底的CICD，部署容器化，非入侵的服务治理和可观测性，对应用开发人员的要求降低了很多，开发只需要关心业务逻辑代码，对服务治理，配置，日志，监控做到零学习零投入，基本上做到了大量用供应商而不依赖供应商，简单的人头外包加少量自有核心开发骨干可以支撑核心系统的交付。

畅享云原生应用经验

企业设计云原生应用架构如何实现

毛剑
B站技术委员会主席

B站技术委员会主席毛剑先生表示："云原生应用面临的挑战，企业可通过企业战略和发展视角二大方面设计云原生应用架构来实现它的最大价值。从企业战略层面来说，任何架构都必须服务于企业战略，基于云 IT 战略，云原生架构可以帮助企业实现泛在接入技术，构建数字化生态系统，还可以从技术的角度（标准化）确保数字化业务的快速迭代，构建面向用户体验管理的数字基础设施，持续优化 IT 成本，降低业务风险。从发展视角来说，相较于传统业务，数字化业务具有更灵活的特性，依托云原生的标准化能力，同时把云当作基础设施，让研发专注在用户、商业价值上。云原生相当于降低了整个研发体系的门槛，统一技术、统一基础设施，统一运维。

刘会福先生补充道，企业上云之后，将为企业提升效率，降低成本，快速迭代，灵活响应。具体而言，从提升效率层面看，云原生的本质就是帮助业务快速迭代，并持续交付，从降低成本来看，云原生基于Docker容器技术的PaaS云平台可以加速业务应用的交付，降低运维成本。

如何平衡技术与团队管理

基于企业的不同阶段，技术管理者要将技术与管理有机结合在一起，做能面向未来、适应变化的架构和设计。结合现有人才梯队，合理做架构选型，组建技术团队，和管理技术团队，作为技术管理者左手抓管理、右手抓技术，应对高速变化的业务，关注行业技术动态，避免技术踏空。

毛剑先生表示：首先，关于团队管理，早期容易'事必躬亲'，不重视组织和团队建设。在团队规模小早期组建的带头干，身先士卒可以让团队快速成长了解技术规划和战略，但是当团队成长到一定阶段的时候，创业公司例如B站的招人和养人方法优先找自己熟悉的人，熟悉的团队，夯实自己的队伍，管理团队一定要做业务战略和产品规划上联动，整个队伍的综合能力才能成长快，为业务创造价值的技术才务实，把有限的资源用在刀刃上。

其次，关于技术管理者的个人成长和挑战的体会：
1. 阶段性放下自前的管理团队进入新的团队（充分轮岗），基于过去的经验和反思去改进自己的管理方式，到一线重新成长不为是一个很好的方式，驻扎到业务团队，把优秀的种子从一个团队带到另外一个团队，持续孵化。
2. 成为公司级有技术影响力的人，一定要深入一线解决一线的技术问题，首先要解决业务问题，推动技术方案落地，深入一线解决困难，带着团队一起攻关，这个很重要。

在谈到时间管理，毛剑先生接着说，可以分为四步：
1. Owning，主要关注和推进的事项，占比50%的时间，聚焦在公司业务、产品层面分解下来的关键动作，要抓好执行和落地，关注交付，重视结果，优化过程。
2. Studying，占比30%的时间，对整个技术体系、技术规划上，以及对业务有价值的技术方向上，

和一线的同事了解这背后的核心技术原理，针对特别有价值的方向研读paper，通过二八学习法，掌握最核心80%的细节。
3. Teaching 占比15%的时间，技术管理者擅长但给一些核心的工作思路和方向，让团队去执行。
4. Delegating，委派给团队，管理者不擅长，但是对业务有辅助作用帮助的。
从大的技术层面：大数据、测试、基础架构、工程架构、云原生、PMO、移动端等跟着团队一起学习与成长，技术管理者基于业务视野与工作模型给到全局最优方案参考给团队来讨论执行。

云原生改造实践经验分享

毛剑先生认为，云原生改造可分阶段进行，先把应用上了，比较关键，因为是第一道业务流量处理的地方，后端的基础设施可以逐步来，比如 SLB、ES、DB、Cache 等开始容器化部署再逐渐托管到 k8s 之中。传统的运维的岗位分为几个阶段转型：

第一个阶段是SRE，他们专注在平台开发，运维自动化。
第二个阶段是从过去 IaaS 层面的 cmdb，开始转到业务应用 cmdb，元数据，开始专注在业务可靠性。
第三个阶段是 SRE BP制度，深度参与到业务架构设计，标准化等工作中，助业务改造上云。对 DB 先PaaS化部署到物理机，使用域名等方式连接，对于容器带来的连接数量问题，是通过研发 DBProxy（集中式，也可以Sidecar），以及使用 类似 MySQL threading pool 方式解决连接的问题。

云原生技术未来发展的趋势

云原生技术未来演进

华为云容器服务首席架构师张琦先生表示："随着整个云计算的发展，其实更多的随着云的发展，稳步构建智能高效的融合基础设施，提升基础设施网络化、智能化、服务化、协同化水平。可以说从国家、政策、市场、产业各个角度来看，基于数字化转型云原生成为新产业、新业态、新模式加速成长的连接器，云原生架构已经被各行各业广泛接受。随着云原生技术的深入发展，我们可以看到一个非常明显的技术趋势就是垂直整合，云原生已经从On Cloud转为In Cloud。整个基础设施与云原生技术深度集成，为用户提供性能更强、支持规模更大的云原生基础设施。

张琦
华为云容器服务
首席架构师

从业务的角度看，为了支持业务的敏捷性，业务单元从单体应用到微服务，运行单元也从虚机发展为容器和server less。在应用上支持更敏捷，更通用部署和管理方式以及更低的运维成本。另外，伴随着以企业应用及数据云化为手段的千行百业数字化、智能化转型不断走向纵深，为确保各产业及行业领域的客户能更充分、全面地受益于云转型所带来的技术红利，分布式云是将云服务分布到不同物理位置，运营、治理和演进可由统一组织负责提供。"

刘如明
信通院云大所云计算部
高级业务主管

信通院云大所云计算部高级业务主管刘如明表示："企业应用分布式云原生带来了资源和管理成本的优化，在趋势方面，分布式云原生也是信通院非常关注的方向，尤其在东数西算的大战略背景下，异构算力特别是涉及到边缘算力单元的感知、回收和调度，目前来看比较好的解决方案就是通过分布式云原生的方式来解决。"

招商证券云原生转型项目调研负责人黄俊补充道，云原生跟信创的结合确实是一个很好的切入点，目前我们证券企业的实践来看，现阶段的信创软硬件虽然在单品性能与稳定性相比传统

黄俊
招商证券云原生
转型项目调研负责人

的IOE有一些差距，但利用云上规模化特点来弥补单品的劣势，发挥全栈软硬一体的联合优势。云原生可看成是第三代云计算技术，是为了更好的运用云计算的大规模调度能力、资源切分能力，其实本身也是有赖于云平台先把底层基础设施的差异性做了一层屏蔽，在基础资源标准化的基础上，再来做PaaS层的标准化，从而达到统一运维、统一交付、统一技术框架的三层目标，而这三层目标的实现，也才能真正助力业务敏捷，才有了DevOps落地的平台基础，这一块在历史包袱中的传统行业，价值更加突显，不过对研发人员、研发体系的要求确实也高了很多。

现阶段云原生技术发展阶段

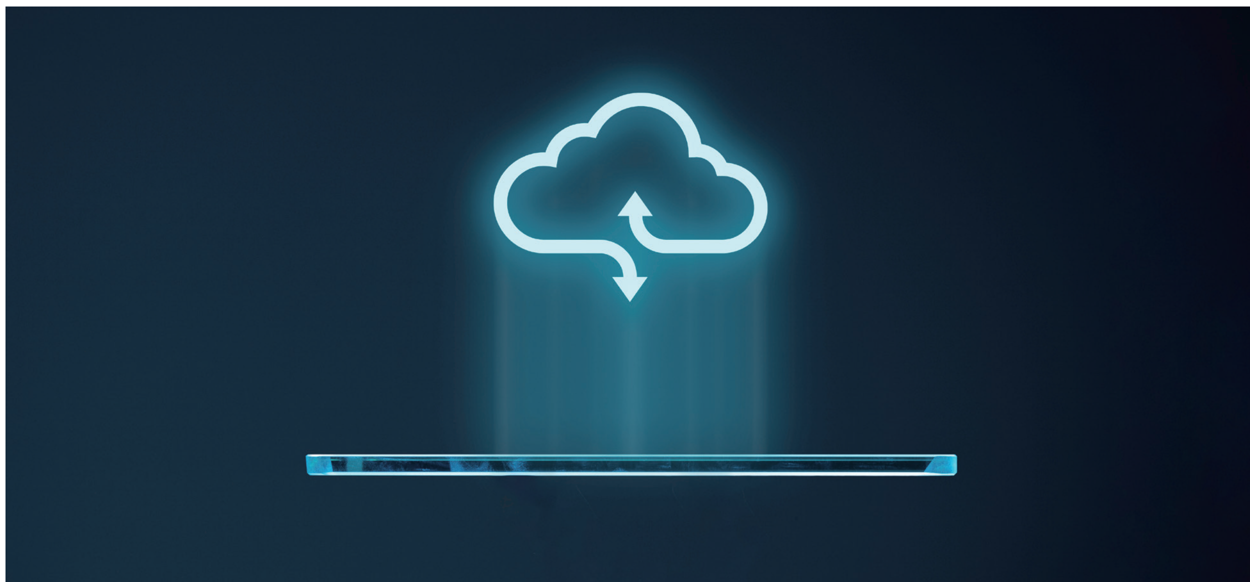
张琦先生表示云原生技术的特性之一就是开源开放。例如华为云主导开源的Kubeedge、Volcano、Karmada 三项技术在边缘计算、多云管理、高可用新、故障恢复和流量调度帮助企业夯实云原生基础设施建设，在实现上下游企业接口互通，解决行业共性问题。另外在今年，基于UCS的成功实践，华为云整合了几个开源项目，构建了开源的分布式云原生套件Kurator，该套件包括新增的开源分布式云原生服务中心 Kappital，可以帮助业界实现云原生应用服务化，并构建全生命周期管理平台，助力企业快速搭建分布式云原生平台，实现企业业务跨云、跨边分布式化升级开发。

沈剑
广联数科CTO

广联数科CTO沈剑认为，技术领域中心化和去中心化一直是走向对立统一。一方面中心化的云原生趋势已经越来越清晰，解决了资源集中化，研发运维集中化自动化，甚至未来随着低代码平台的大量应用以及和当前开发工具的融合，集中化趋势会更明显。另一方面，随着物联网终端的海量化，边缘计算带来的去中心化也会非常明显，整个网络体系的分层。

杜广源
昆仑数智科技技术总监

昆仑数智科技技术总监杜广源表示企业级应用向云原生架构转有很大难度，例如ERP、MES、CRM这些应用，不改变长事务、数据强一致性要求， K8S、Docker、DevOps还是服务化架构这些都还是流于表面，需要业务模式的变革，需要真正理解行业方向和信息化技术的专家引领。而央国企现在大范围在使用云原生技术，包括华为云，甚至在谋求云原生的应用重构，实际诉求有如下几种:



1.央国企信息化建设基于云对底层架构的变化对应用转型有驱动作用;
2. 国产化是大方向，而基础平台包括IaaS+PaaS的国产化是最简单的，也是能快速见效的;
3. 传统的企业级应用和基础设施技术架构的变化之间有Gap，例如，传统解决海量关系型数据的方法和手段是数据库调优、表分区、内存数据、缓存等等，云原生给了新的思路，服务拆分、业务和数据内聚，拆分多个独立的、无关联或者关联较小的库，这对解决传统关系型数据库的性能，甚至推进数据库国产化替代都是有巨大作用的;
4. 在云上的应用，如果不是云（原生）架构，很难发挥云的优势，属于技术架构和真实运行运维的不匹配;
5. 数字化转型需要传统业务对数据能力的接入和业务灵活变化，这对云原生的转型有促进作用。短期内，用了云原生的技术不代表就是云原生的架构，未来云原生在ERP、CRM、MES这样的大块头有了云原生的版本可能有逐步统一的趋势。

云原生技术未来极具应用潜力的行业和场景

张琦先生认为，未来云原生技术会渗透进更多行业，同时也会接受不同行业的不同需求进行更深入的进化。

柴思远
大搜车技术负责人

大搜车技术负责人柴思远表示从场景的角度看，我认为云原生数仓在国内的有极大发展空间。随着企业数字化的完善，Saas行业的爆发式增长，企业已经不缺数据了。以前都是大企业在建设数据中台，如今中小企业也开始意识到数据的价值。中小企业天然的研发资源不足，更多的业务跑在Saas上和平台上，如何更低门槛帮助这些中小企业构建自己的数据仓库，降低数据接入成本，存算分离实现更精准的安需收费，构建从接入到应用的全链路生态合作，是非常有价值的，云原生让云计算资源的使用更敏捷。

黄俊先生同意并指出，数据库的容器化，这个确实如大家所说，要视具体情况来看，数据库本身是一个性能敏感性产品，而盲目容器化，至少在运行层，相比物理机，多了一层容器的网络转发时延，如果处置不当，反而会造成容器化性能下降的假象。一般而言，我们更倾向于先裸金属上云，然后再此基础上，借助云的硬件卸载能力帮容器化造成的网络损耗、存储性能损耗Hold之后，再来考虑容器化，但上云是毋庸置疑的，这样才能更多地利用硬件能力为软件运行性能加把劲。

云原生未来发展趋势

近年来，云原生技术有了新的发展，其目标是更好地服务于全球范围内的政府和企业，推进他们的数字化、智能化改造，同时保障业务高效、可靠、安全，其用云的广度和深度也与之前的消费互联网时代明显不同，进入了全新的2.0阶段，这个阶段要解决的主要问题是以“应用为中心”，使各行各业更好地运用数字技术。在华为今年发布的迈向智能世界白皮书中，也阐述了云原生2.0十大关键技术趋势，包括：分布式云原生、以应用驱动的基础设施、业务混合部署与统一调度、存算分离、数智融合、多模态AI、可信及平民化的DevOps架构、无服务器（Serverless）架构、基于软总线的异构集成架构和全方位立体化安全架构模式。

张琦先生最后表示，云原生在重构整个软件生命周期，我们的判断是云原生底层系统将会迎来一波的技术创新高峰，相信不远的未来，云原生将在更多的行业、更多的场景迅速落地，助力企业在数字化转型浪潮中持续增长，引领时代变革。

4

云原生 Cloud Native

实践分享

陌陌: K8s 在容器资源超售方面的优化实践 ---------- 73-77

vivo: 基于 Karmada-Operator 多云容器编排平台的实践 ---------- 78-86

街电: 在华为云云原生实践 ---------- 87-89

探真科技: 云上原生的“右移”安全实践 ---------- 90-94

陌陌：K8s 在容器资源超售方面的优化实践

周峰 陌陌云基础设施团队虚拟研发负责人

业务场景描述

大部分业务在申请容器时，没办法准确预估服务要使用的CPU资源，所以在设置服务实例的配额时，业务都倾向于申请更多的CPU资源量，如下图所示：这个服务晚高峰期间最多使用0.6核CPU，但申请的时候却要了4个核

很多服务都这样设置造成集群的CPU资源已经被申请完了，但资源利用率却不高，如上图所示：集群的CPU资源已经被分配了94%，但集群的CPU峰值却只有33%。

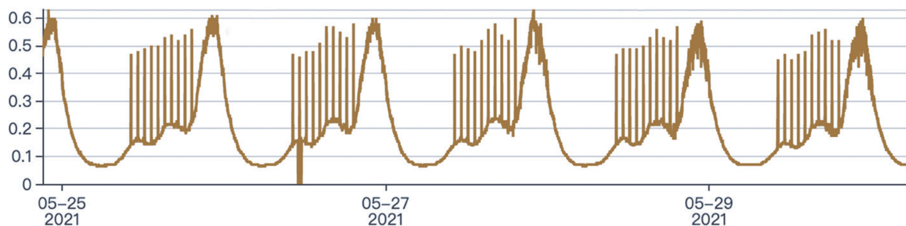
同样的问题在内存资源上也存在，不过内存资源没有CPU资源那么明显，见下图：

为了解决上面的CPU资源和内存资源业务申请和实际使用偏差非常大的问题，我们引入了针对性的资源超售插件。

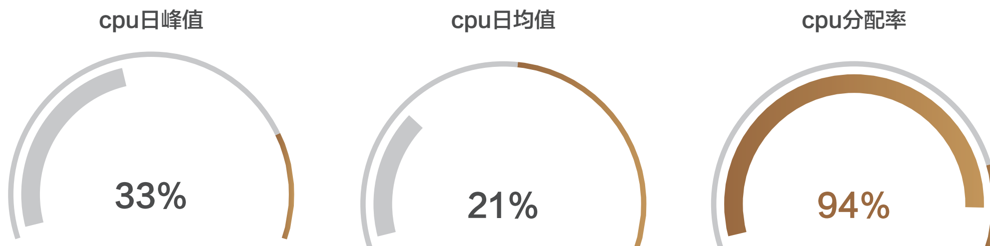
CPU 使用核数量

container_cpu_usage_seconds_sum(CPU使用核数)

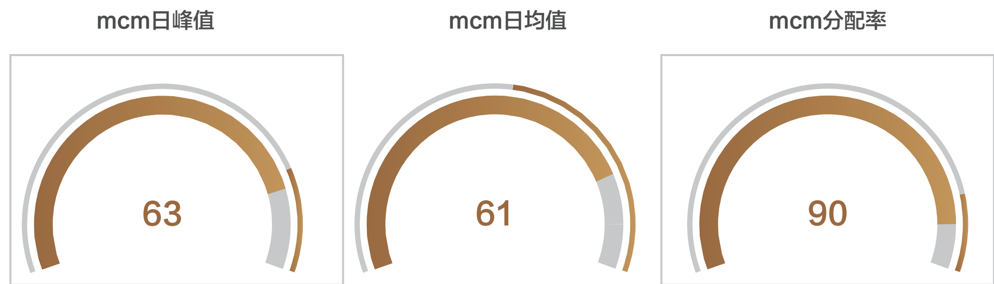
实例聚合：avg 时间聚合：1m 时间聚合函数：avg 绘制空值



CPU 资源利用率



内存
资源利用率

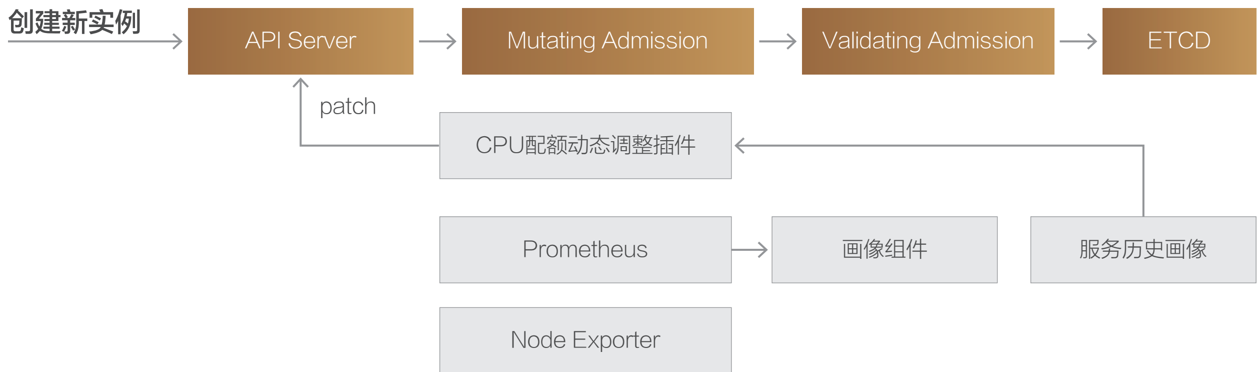


具体解决方案

服务粒度的CPU资源动态压缩机制

由于CPU属于可压缩资源，比如，如果容器的CPU使用量达到Limit值，容器不会被kill，但会被限制，也就是常说的CPU抑制；如果容器的CPU Request设置的比较小，当节点整体的资源利用率不高时，不会影响容器的正常运行，当节点整体的资源利用率比较高时，会影响容器的CPU时间片分配，也会出现CPU抑制。

考虑到上面CPU资源的特性，我们最终决定在服务粒度对业务实例的CPU资源进行校准。如下图所示：这是CPU资源动态压缩机制的大致流程。



从上图可以看出：

» 当业务新实例创建时，我们基于K8s原生的Pod Mutating Admission介入，API server会将新Pod信息转发给CPU资源动态压缩服务。

» CPU资源动态压缩服务从我们自研的画像组件中获取该服务上个晚高峰的CPU资源使用量最大值，然后通过patch方式更新该新实例的CPU Request值。

» API server会以修改后的值往下推进新实例的调度和创建。

通过这种方式，我们将业务实例的Request值从业务设置值修改成了改业务真实使用的最大量。这个机制在落地过程中，有些细节问题需要注意下：

对于新服务，是没有历史画像的，要如何处理？

我们的做法是采用了默认值设定，先以当前服务Limit值的50%作为初始值，等业务下次发布再进行延迟校准。这里没有考虑更复杂做法的主要原因是新服务的占比是非常少的，在整个容器集群中基本可以忽略不计。

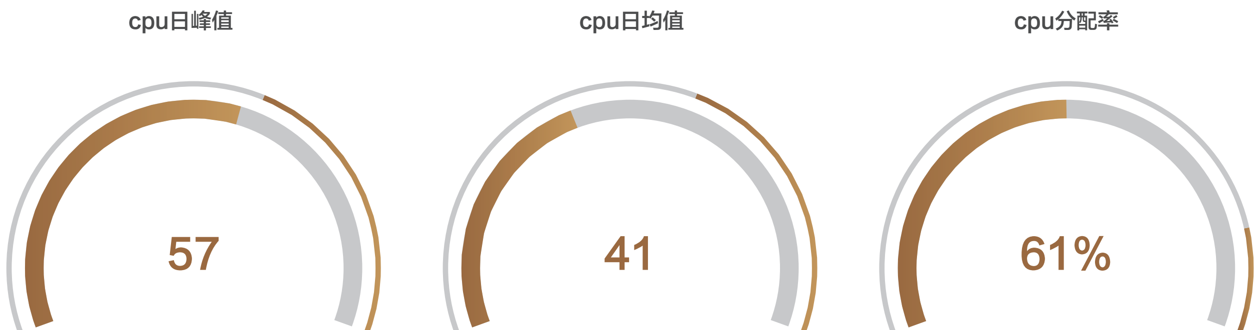
对于基础组件类容器，按常规值设置会有什么问题？

比如mesh容器，晚高峰期间的CPU资源使用量并不一定是最大值，如果设置值比实际使用值低很多（如果偏差不是很大，是没有什么影响的），可能会导致CPU抑制，影响基础组件的稳定性，为了保证mesh等基础容器的稳定性，我们设置了定制化策略：如果mesh容器的Request值比实际晚高峰的使用量更大，我们不会调整，而是保持原设置；如果mesh容器的Request值比实际晚高峰的使用量小，我们会调大相关值。

对于长期不发布的实例来讲，压缩值失准怎么办？

业务发布时，实例按照上个晚高峰实际使用量设置CPU Request后，如果过了很长一段时间，服务都没有再发布，但是对资源的实际使用量确在不断增加，这就会导致设置的Request值比当前使用量偏低。上面的问题里其实有描述过偏低的影响，我们为了解决这个问题，引入了巡检机制，会实时发现偏差比较大的情况，经验值设定是设置值比使用值低50%以上或少2个核以上时，会自动重建实例进行校准。

该插件落地后，对上面的CPU偏差问题有明显改善，如下图所示，CPU分配率和峰值基本持平。

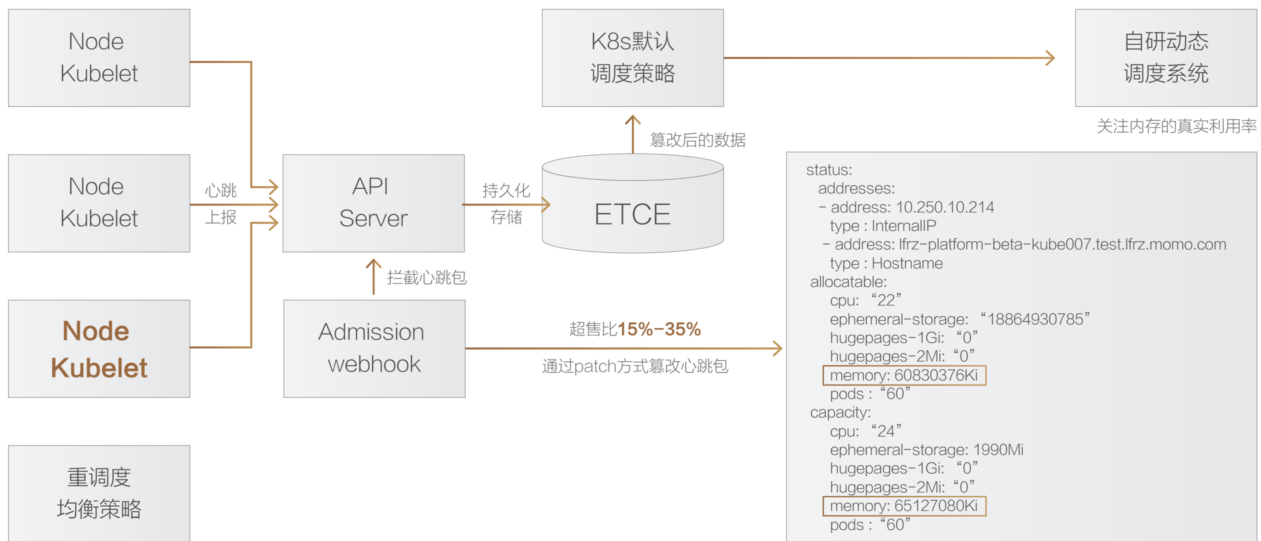


节点粒度的内存资源超售机制

由于内存是不可压缩资源，比如，如果容器的内存使用量达到Limit值时，容器会被内核因OOM而kill；如果容器的内存Request设置的比较小，容器会无法正常启动，直接OOM。考虑到内存设置是否恰当对业务实例的影响比较大，并且我们很难找到统一标准去给所有业务设置Request值，所以我们没有选择直接在容器维度上进行内存超售，而是让每个业务容器根据自己的实际情况都有一定冗余，而是在节点维度上进行了内存超售，我们的具体实现如下图所示：

从下图可以看出：

- Node Kubelet定时会向API Server进行心跳上报，心跳包里会把当前节点的已分配资源和总资源量上报给API Server。
- 内存超售插件会拦截心跳包，并根据不同集群的情况设置不同的超售比，对心跳包中的allocatable/memory和capacity/memory进行修改。
- API Server获取到篡改以后的数据并把它存到Etcd后，K8s默认调度器会以修改后数据为准进行分配实例，自研动态调度系统会以节点真实的内存利用率情况进行分配实例，两者结合最终把实例调度到最合适的节点上。
- 节点维度内存超售后，上面可以调度的实例会更多，难免有些情况节点的内存利用率会突增并超过阈值，这时会依靠重调度均衡机制发现和解决。



这套机制在实践过程中却有很多细节要关注下，下面重点描述下：

如何识别心跳包是原生的还是超售后的？

按正常情况，Node Kubelet每次上报的心跳包都应该是超售前的allocatable/memory和capacity/memory，但在实践过程中，我们发现心跳包会因为其它组件偶发修改也会上报超售后的allocatable/memory和capacity/memory数据，如果不正常识别，会导致对节点的这两个数据进行多次超售而引发严重的问题。所以超售组件中拿到心跳包以后，必须识别出是原生的还是超售后的，为了区分心跳包是否已经超售过，我们用了一个比较巧妙的方法，对超售后的数据的末尾3位全部取零，在拿到原始心跳包的时候会根据这个特征识别是正常心跳包还是偶发的超售后的心跳包。

如何处理超售组件挂掉以后的影响和风险？

超售组件挂掉以后，节点的allocatable/memory和capacity/memory会退化为初始值，而之前在节点超售状态下调度进来的实例可能会导致node的sum(pod_request) > 初始allocatable/memory，如果此时Node Kubelet发生重启，会对节点上的部分实例进行驱逐，直到node的sum(pod_request) < 初始allocatable/memory为止。极端情况下，如果有大批量的node出现上述情况，会导致大批量实例驱逐而发生pending。

由于上面的情况是K8s原生特性，无法修改，所以只能从机制上尽可能避免，我们的思路主要是两个，一方面会从超售组件的高可用入手，尽量简化超售组件的逻辑，同时多节点部署；另一方面是会通过巡检控制超售后sum(pod_request) > 初始allocatable的node数量。

如何处理集群版本不一致问题？

K8s集群总会因为各种原因出现多版本情况，如果你们公司没有这个问题，那自然是极好的，我们公司会有1.10.x到1.18.x的版本跨度，不同版本有些相关机制是存在差异的：

- 需要的依赖不同：MutatingWebhookConfiguration在v1.16之前对应的apiVersion是admissionregistration.k8s.io/v1beta1，从v1.16之后变成了admissionregistration.k8s.io/v1，所以超售组件需要在启动的时候缓存集群的版本信息，保证用对应的依赖包处理node的心跳数据。

» 心跳包上报的周期不同：在v1.13之前，节点的心跳只有nodestatus，默认上班周期是10s，在v1.13之后，NodeLease feature作为alpha特性引入，当启用NodeLease feature时，每个节点在kube-node-lease命名空间中都有一个关联的Lease对象,NodeLease 每10s更新一次，而只有在NodeStatus发生改变或者超过1分钟时，才会将NodeStatus上报。这个机制的不同对于超售组件异常时的影响是有差异的，即如果心跳包因为超售组件持续上报失败时会因为对应集群的版本不同而有不同影响。

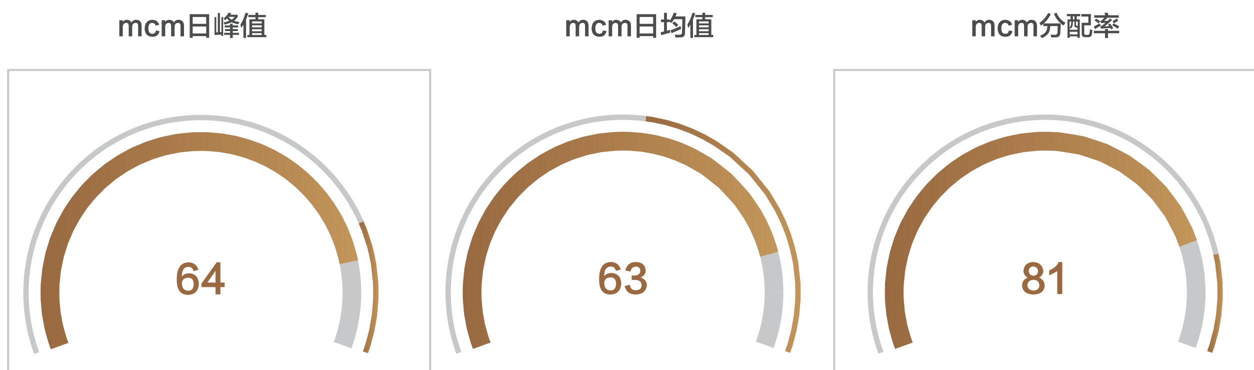
对节点内存进行超售后，监控机制需要联动适配吗？

我们是基于prometheus组件进行数据采集和监控的，超售后，kube_node_status_allocatable_memory_bytes的值会发生变化，对应的分配率数据也会对应变化，但这是符合预期，并不需要额外去做适配工作。

对节点进行超售时，节点的预留资源即capacity-allocatable的值如何确保？

我们采用的方法是预留内存大小保持不变，默认每个Kubelet会预留8Gi的内存空间，按照超售比算出新的allocatable/memory后，在这个基础上增加8Gi，得到新的capacity/memory。

该插件落地后，内存分配率和实际使用率偏差很大的问题得到明显环节，如下图所示：



总结和规划

其实服务粒度的CPU超售机制和节点粒度的内存超售机制在实现上并不复杂，最大的挑战在于很多细节问题的处理上要全面些。

目前这两套机制都已经在生产环境大面积应用，而且成功解决了对应问题。

后续我们还会考虑如何在不重启容器的情况下动态修改容器CPU Request等特性。

vivo：基于Karmada-Operator多云容器编排平台的实践

张荣 vivo互联网高级研发工程师

背景

随着vivo业务不断迁移到k8s上，集群规模和集群的数量快速增长，运维难度也急剧增加。为了构建多集群技术，我们也自研了多集群管理，但无法解决我们遇到的更多的问题。后来开始对社区相关项目做了细致的调研和测试，我们最终选择了Karmada。

主要原因如下：

- 具备对多套K8s集群的统一管理能力，业务通过服务维度去管理资源，降低容器平台的管理难度。
- 跨集群的弹性伸缩和调度能力，实现跨集群的资源合理利用，从而提升资源利用率并节约成本。
- Karmada完全使用了K8s原生的API，改造成本低。
- 容灾，Karmada控制平面与member集群解藕，集群异常时支持资源重新分配。
- 可扩展性，如可以添加自研的调度插件和添加自研Openkruise解释器插件等。

在我们探索怎么使用Karmada的同时，我们也遇到了Karmada自身运维的问题。

社区部署工具较多，需要用户自己选择。当前用户部署方式如下：

- Karmadactl
- Karmada charts
- 二进制部署
- hack目录下脚本

对于上面的几种工具，在Karmada的社区开展了问卷调研，并生成了统计报告。
主要总结如下：

- 社区部署工具较多，需要用户自己选择。
- 部署脚本也存在缺陷，需要用户自己解决，github上关于这方面的提问较多。
- 黑屏化操作，没有提供k8s api操作，用户难以产品化，我们主要期望对接我们的容器平台，实现可视化安装。
- 缺少CI测试和部署工具的发布计划。
- etcd集群缺少生产环境的关键功能点，如etcd的高可用、定期备份和恢复。
- 需要安装很多依赖插件，涉及到Karmada控制平面、Karmada的host集群和member集群。
- 缺少一键部署和配置繁琐等痛点。

针对以上问题，本文将分享Karmada-Operator的vivo实践，包括Operator的方案选择、API、架构设计和CI构建等。

Karmada-Operator的落地实践

Operator Framework 是一个开源工具包，用于以有效、自动化且可扩展的方式管理 Kubernetes 原生应用程序，即 Operator。Operator 利用 Kubernetes 的可扩展性来展现云服务的自动化优势，如置备、扩展以及备份和恢复，同时能够在 Kubernetes 可运行的任何地方运行。

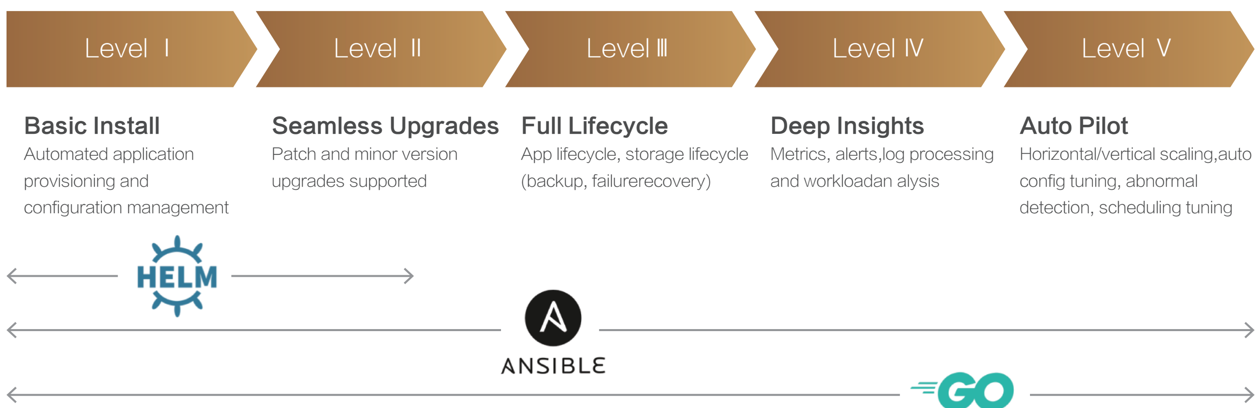
2.1 Operator SDK介绍

Operator 有助于简化对 Kubernetes 上的复杂、有状态的应用程序的管理。然而，现在编写 Operator 并不容易，会面临一些挑战，如使用低级别 API、编写样板文件以及缺乏模块化功能（这会导致重复工作）。

Operator SDK 是一个框架，通过提供以下内容来降低 Operator 的编写难度：

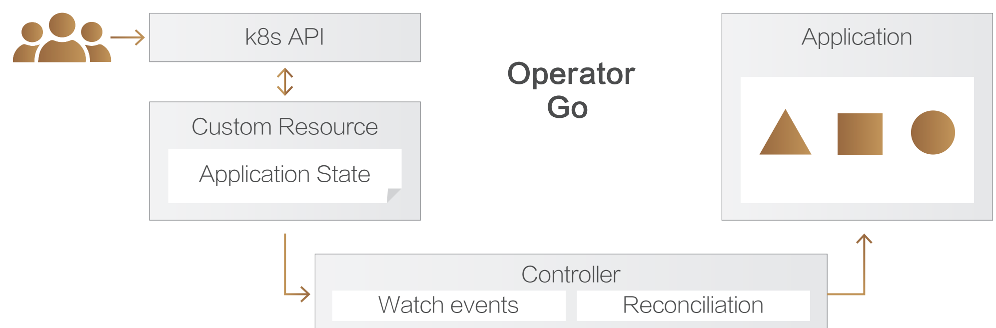
» 高级 API 和抽象，用于更直观地编写操作逻辑
» 支架和代码生成工具，用于快速引导新项目
» 扩展项，覆盖常见的 Operator 用例

如图所示，operator sdk可以基于helm、ansilbe和go构建operator，我们需根据当前的情况选择我们合适的operator框架。

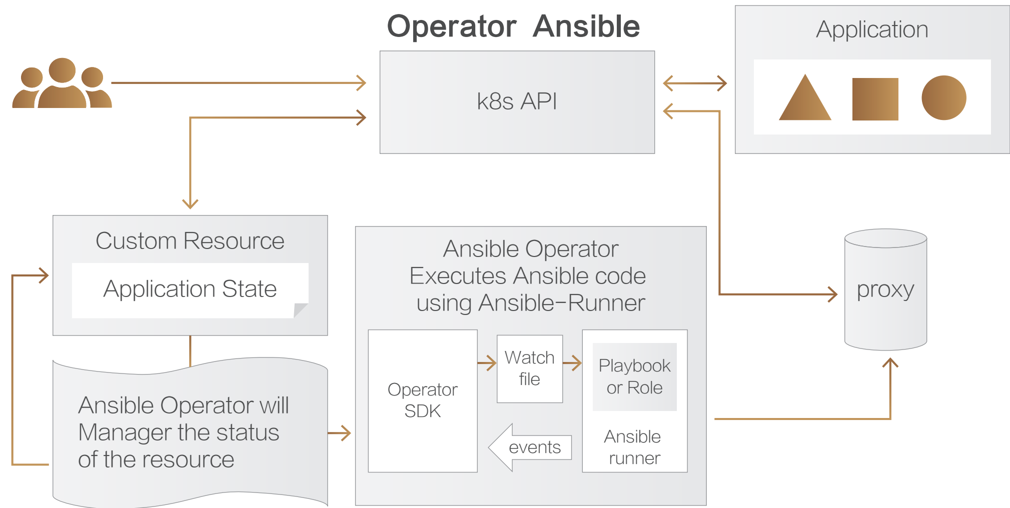


2.2 方案选择

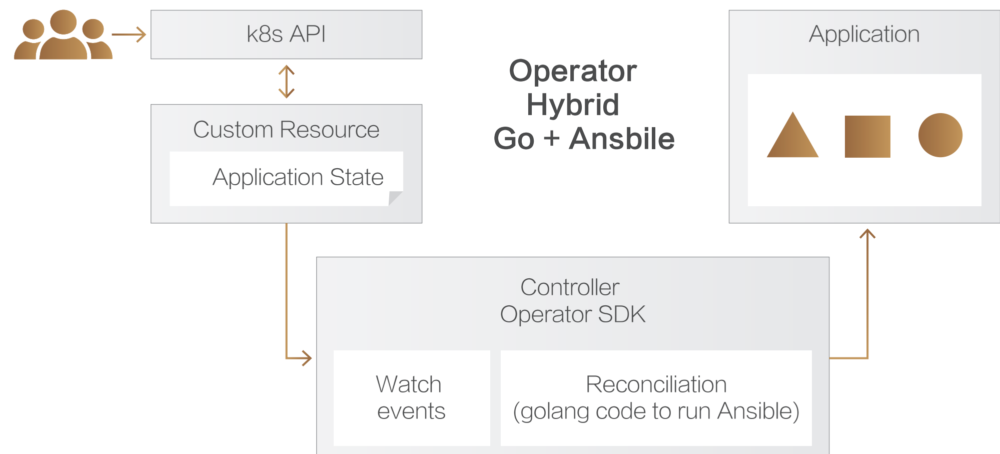
方案一：golang 开发 Operator



方案二：
ansible开发
Operator



方案三：
golang和ansible
混合开发Operator



根据Karmada的实际生产部署调研情况和vivo自身的实践，可以总结如下：

- 要支持在K8s集群和不依赖K8s集群二进制部署。
- 支持外部独立的etcd集群部署或者对接已有的etcd集群。
- Karmada集群具备迁移能力，如机房裁撤和机器故障等，就需要etcd集群管理有备份和恢复能力，如根据etcd备份数据快速在其它机房恢复集群。
- 需要支持第三方的vip给Karmada-apiserver提供负载均衡，目前vivo都是基于外部vip，并提供了域名。没有使用K8s的service给Karmada-apiserver提供负载均衡。
- Karmada控制平面一键部署和member集群的自动化注册和注销。也需要获取member集群的kubeconfig，pull模式也需要在member集群部署Karmada-agent。
- Karmada集群的addons插件安装，如istio、anp、第三方的crd等安装，需要在Karmada的控制平面、host主机集群，甚至需要在member集群上进行配置。
- 提供api能力，实现可视化部署。
- 针对Karmada单个组件的单独升级和全量升级。
- 支持在offline和离线模式。

面对Karmada如此复杂的条件限制，我们再来分析下上面3个方案谁可能比较合适。

方案一：

基于go开发的Operator，比较适合基于K8s集群的有状态服务管理，如etcd，数据库等，比较成熟的有etcd-Operator。Karmada涉及到不依赖K8s集群二进制部署、外部etcd、member集群的注册、注销和插件安装，不能很好的支持或者需要增加开发量。

方案二：

基于ansible开发的Operator，既可以基于K8s集群的对状态服务管理，也可以脱离K8s集群对如不依赖K8s集群二进制部署、外部etcd、member集群的注册、注销和插件安装。这里主要通过ansible 的ssh登录能力和K8s模块管理，通过调研我们也发现90%以上的用户可以提供ssh登录。

方案三：

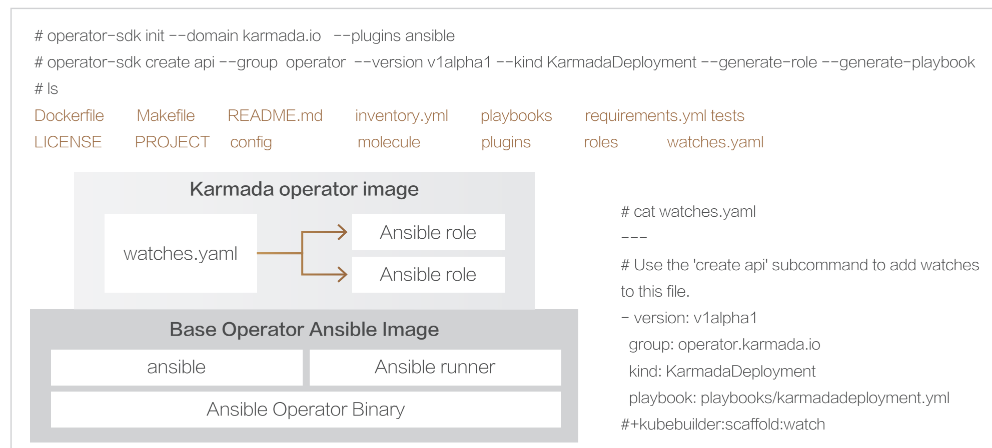
基于go+ansible的混合的Operator，读者可以阅读vivo开发的Kubernetes-Operator，就是基于这种方案。方案三具备方案二的所有能力，因为底层都是通过ansible去执行。

首先我们排除了方案一，对于方案二和方案三，本人也纠结了很长是时间，最终我们选择了方案二。主要原因如下：

- Operator SDK ansible已具备了和Operator SDK go相等的能力，并提供K8s、K8s_status模块、相似的webhook功能去对k8s的资源管理，以及reconciliation的能力。
- 符合目前Karmada实际生产部署的需求。
- 简单易学，只要知道ansbile的jinja模版、和K8s相同的yaml文件。你只需要编写ansible task，开箱即用，reconciliation由Operator SDK 解决。
- 对于常用ansible的人比较友好，不需要写golang代码。
- 扩展能力强，用户可自定义插件。管理端也支持local、ssh、zeromq三种方式连接。local模式可以直接对接K8s接口，ssh模式可以登录执行脚本。可以很好的混合使用，解决我们当前的需求。
- Karmada运维操作相对K8s要简单，不需要复杂的crd定义，ansible需要解析少量vars去执行playbook就行。golang+ansible模式比较适合复杂CRD定义和业务逻辑复杂的系统。

如下图所示，我们只需要执行Operator-SDK create api命令，就可以创建 KarmadaDeployment的CRD，然后就可以定义KarmadaDeployment的API。在watches.yaml里实现Reconcile的业务逻辑。

2.3 API设计



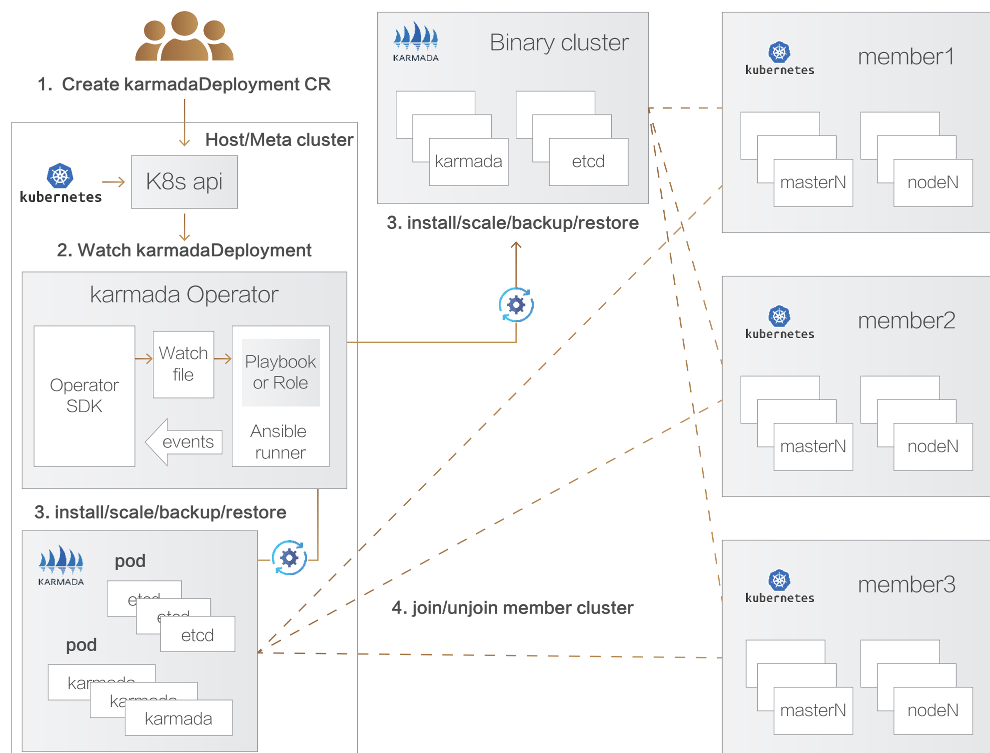
这里主要定义KarmadaDeployment、EtcdBackup和EtcdRestore个资源，分别用于Karmada的部署，和etcd的数据备份和恢复。ansible Operator会根据spec里定义解析成ansible的vars。status将通过 ansible runner 输出为用户自定义的状态。也可以通过ansible的k8s_status更新KarmadaDeployment的状态。当前主要考虑的是在K8s运行Karmada，后面会添加二进制部署模式，当前的CR没有涉及。

```
apiVersion: operator.karmada.io/v1alpha1
kind: KarmadaDeployment
metadata:
  name: karmadadeployment-sample
  namespace: karmada-system
spec:
  imageRegistry: ""
  clusterDomain: "cluster.local"
  etcd:
    size: 3
    version: "3.4.9"
    pvc:
        storageClass: "local-path"
        size: "1Gi"
        storageType: "pvc"
  apiServer:
     size: 1
     version: "v1.2.0"
     serviceType: "ClusterIP"
     loadBalancerApiserverIp:  “”
   …………
  members:
    - name: "member1"
      syncMode: "push"
      kubeConfigSecretName: "member1-config"
    - name: "member2"
      syncMode: "pull"
      kubeConfigSecretName: "member2-config"
```

```
apiVersion: operator.karmada.io/v1alpha1
kind: EtcdBackup
metadata:
     name: ” etcd-1”
spec:
    etcdEndpoints: <etcd-cluster-endpoints>
    storageType: <store_type>
    <store_type>:
            <key>: <vaule>
            <key>: <vaule>
```

```
apiVersion: operator.karmada.io/v1alpha1
kind: EtcdRestore
metadata:
     name: ” etcd-1”
spec:
    etcdCluster:
         name: “etcd-test”
     storageType: <store_type>
    <store_type>:
            <key>: <vaule>
            <key>: <vaule>
```

2.4 架构设计

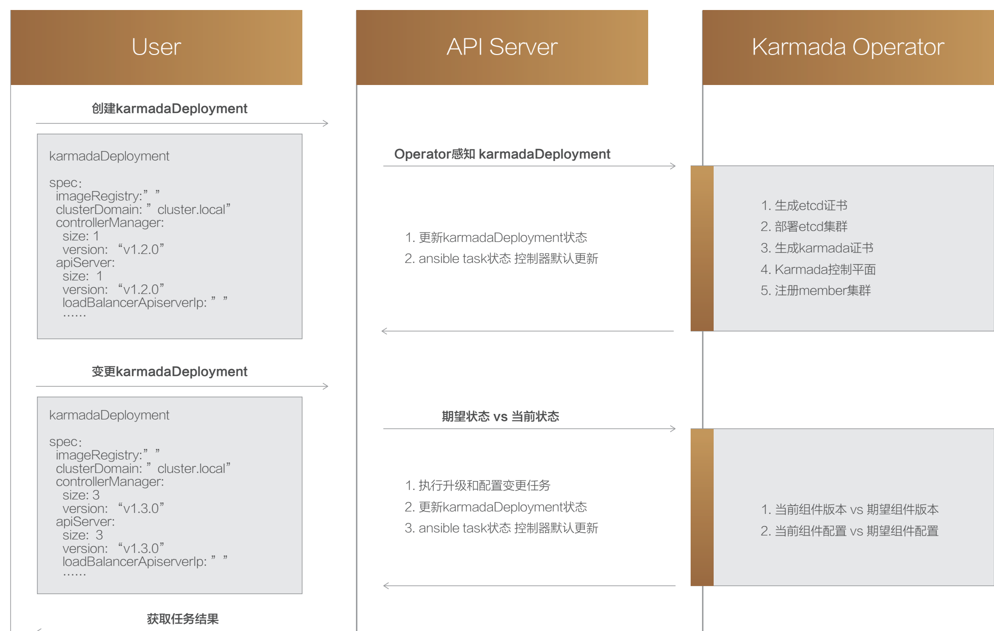


如上图所示Karmada Operator提供了容器化和二进制集群部署设计，其中Karmada的容器化部署不需要执行ssh登录，只需通过K8s和k8s_status就可以完成karmada控制面的管控。Karmada的二进制部署主要通过ssh登录完成Karmada控制平面的管控。member集群的join和unjoin需要提前提供member集群的kubeconfig文件，也可以设置member的登录权限操作，需要在CR里定义member集群的用户和密钥。

执行流程如下。

» 用户通过KarmadaDeployment定义Karmada操作
» Karmada Operator感知KarmadaDeployment的CR变化，开始进入控制器逻辑
» 根据用户的定义，选择是容器化部署或者二进制部署，开始执行安装、扩所容和备份等操作
» 执行join/unjoin操作，将member集群注册到Karmada集群或者注销member集群

2.5 Karmada 控制平面管理



如上图所示，主要是karmada控制平面生命周期管理，对比当前社区的部署工具我们如下优化：

» 标准化证书管理，主要是用openssl生成证书。其中etcd和Karmada证书单独分开维护，和k8s集群证书命名相同，方便接入我们的监控。
» Karmada-apiserver支持外部负载均衡，不限于当前的k8s service提供的负载均衡。
» 更灵活的升级策略，支持单独组件升级和全量升级。
» 更丰富的全局变量定义，计划支持组件配置变更等。

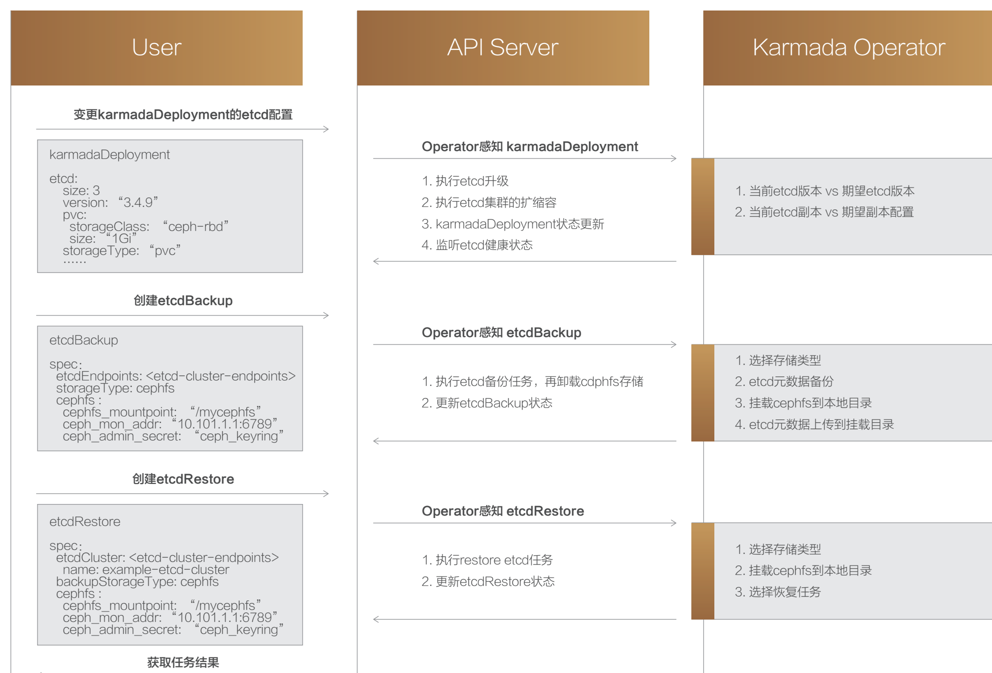
2.6 etcd集群管理

etcd集群是Karmada的元数据集群,生产中需要保证etcd集群高可用和故障恢复等。如上图展示了etcd集群必要的生产要素，如自动扩缩容、升级、备份和etcd集群的故障恢复。自研了基于ansible的plugins和library，实现etcd集群管理能力如下：

» 添加member到存在的etcd集群。
» etcd集群删除member。
» etcd集群的备份，比如支持cephfs的数据备份。
» etcd集群故障恢复。
» etcd集群健康状态查询。

这里定义了etcdBackup和etcdRestore的CR，没有合并到KarmadaDeployment里。主要考虑到etcd集群本身操作的安全性和简化KarmadaDeployment的ansible任务。其中etcdRestore功能，可以根据etcd集群备份数据，实现导入到新的etcd集群，从而恢复Karmada集群所有的业务状态。当前主要场景如下:

» Karmada集群所在的机房裁撤，需要备份etcd数据，迁移到新的Karmada集群。
» 期望通过Karmada-Operator管理Karmada集群，只需备份etcd数据，实现etcdRestore功能即可。
» Karmada集群故障，可以通过etcd备份数据，结合etcdRestroe实现故障恢复。



2.7 member 集群管理

member集群的生命周期管理主要有注册和注销，上图是执行的流程。为了处理member集群的注册和注销，这里会动态的生成inventory。Ansible Inventory 是包含静态 Inventory 和动态 Inventory 两部分的，静态 Inventory 指的是在文件中指定的主机和组，动态 Inventory 指通过外部脚本获取主机列表，并按照 ansible 所要求的格式返回给 ansilbe 命令的。

这里Karmada-Operator基于k8s的CR实现了动态inventory plugins，主要通过解析KarmadaDeployment的members定义去动态的生成inventory。这里添加了add-member和

del-member 2个角色， add-member里集群会被注册到Karmada控制平面，del-member里的集群会被从Karmada控制平面注销，这样就可以并发的注册和注销多个member集群。同时也可以提供ssh登录模式，方便后期扩展。

Karmada-Operator 的 CI 介绍

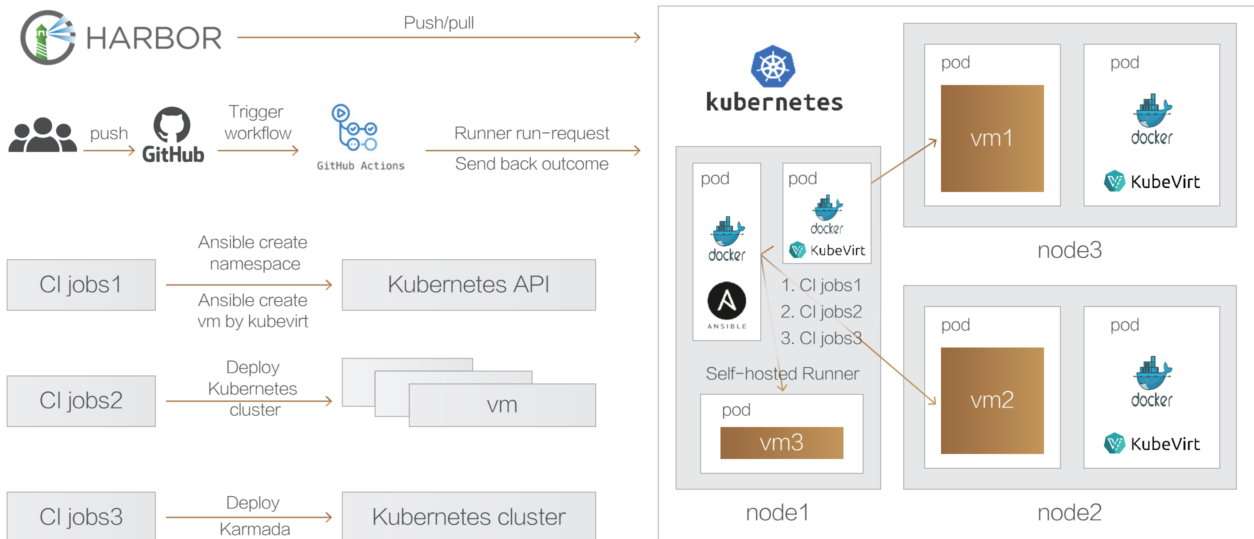
为了更好的提高开发人员的体验，计划提供Karmada-Operator的CI构建能力。这里在K8s集群里部署github的self-hosted Runner和kubevirt。

- 用户在github上提交PR
- 触发github Actions，我们在self-hosted里定义的流程
- 执行语法和单元测试
- 通过kubevirt创建vm
- 在多个vm里部署1个host和2个member集群
- 部署Karmada和添加member集群
- 执行Karmada e2e和bookfinfo案例测试

计划添加的CI矩阵测试如下:

语法测试

- ansible-lint
- shellcheck
- yamllint
- syntax-check
- pep8



集群部署测试

- Karmadactl、charts、yaml和二进制部署和所有配置项安装测试
- join/ unjoin member 集群
- 升级Karmada
- etcd集群的备份和恢复

功能测试

- Karmada e2e测试
- 创建bookinfo案例

性能测试

我们主要通过kubemark组件模拟了多个2000节点的member集群进行了性能测试，其中一个测试案例是集群故障转移，结论是4w个无状态pod能够在15分钟完成故障迁移，有机会可以分享我们的性能测试。

总结

通过社区的调研和vivo的实践，最终确定了Karmada-Operator方案设计。Karmada-Operator具有高度可扩展性、可靠性、更直观地编写操作逻辑和开箱即用等特点，我们相信通过这种高度可扩展的声明式、自我修复云原生系统管理Karmada，为我们全面切换到Karmada去管理业务提供了强有力可靠保障。

基于ansible的operator也存在如下缺点。第一点没有提供webhook的能力，需要自己添加或者在ansible的task添加相关的校验；第二点是自动生成了通用的CRD模版，没有详细可定义的脚手架工具去自动生成CRD。

当前Karmada-operator还在初始阶段，提供了方案和部分实践，具体功能还需不断的完善和改进。具体可以查看vivo的Karmada-Operator仓库,欢迎大家试用和提建议。当前代码提供的能力矩阵可以查看项目规划。

街电：在华为云云原生实践

右右 竹芒科技高级容器运维工程师

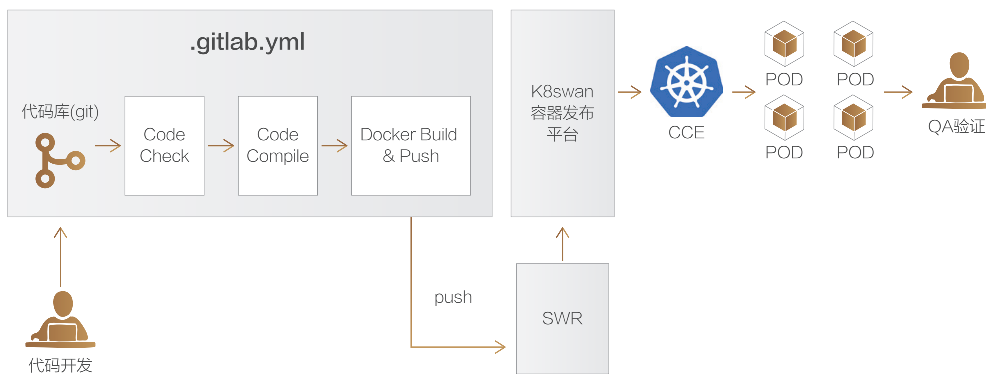
前言

云原生的概念最早在2010年被提及，当时人们只是想用一种方式描述应用程序与中间件在云环境中良好的运行状态；在2015年CNCF基金组织成立，云原生进入了高速发展轨道，随着各大公司加入，逐渐构建出围绕云原生的具体工具。随着时间的发展云原生逐渐从概念层面演变得更加实体化，从最初的容器化封装、自动化管理、面向微服务，到构建一个统一的、开源的云技术生态，能和云厂商服务解耦的开源技术栈。

云原生随着新技术出现发展而演化，从最初的容器化封装、自动化管理、面向维服务到现在Devops、持续交互、容器、服务网格、微服务和声明式API，这个生态越来越多样化。街电业务从2012年上线至今，也是经历了以上几个场景，从大单体构建到现在微服务、从人工构建到持续交互、从虚拟服务到现在的容器和服务网格稳定运行。下面谈谈街电业务目前在华为云Devops实现和基于CCE与ASM构建的云原生实践。

自动化构建与发布

Devops是将公司开发和运维，从组织到流程打通，如何合作、如何划分边界提升组织效率。街电devops工具链依赖gitlab，结合了华为云的SWR，从代码的静态检查、编译、打包、构建到SWR，实现代码级别的持续交付，提高交付效率。通过gitlab CI实现设计、开发、测试、发布、运维各个阶段的重复性工作，通过工具实现标准化、规范化，降低出错概率。

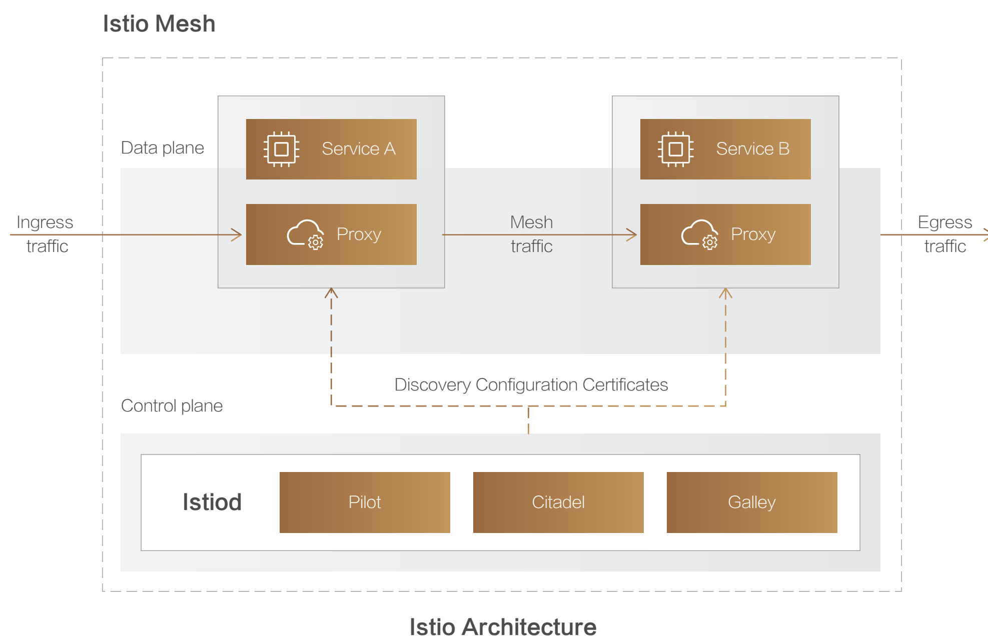


对于开发过程来说，少交流、少沟通、少开会就是最高效的；对于运维来说，少参与、少配置、少动手也是最高效的。SWR是华为云产品化的容器镜像服务，通过创建组织，在CI阶段自动化推送容器镜像。结合SWR我们提高构建与发布稳定性，减少自建镜像仓库带来的宕机风险，大大降低运维人员的运维压力。下面来看看街电结合SWR与CCE实现的自动化构建和发布流程：

在代码开发阶段，开发人员将代码推送到代码仓库，通过自动触发的方式实现代码级别代码检查、自动化测试、代码编译打包、构建成docker镜像，最终推送到容器镜像服务。另外，k8swan是我们自研的一套容器发布平台，高度整合helm chart发布，具备自动化的审核流程、一键回滚等等功能，完美整合SWR与CCE集群。

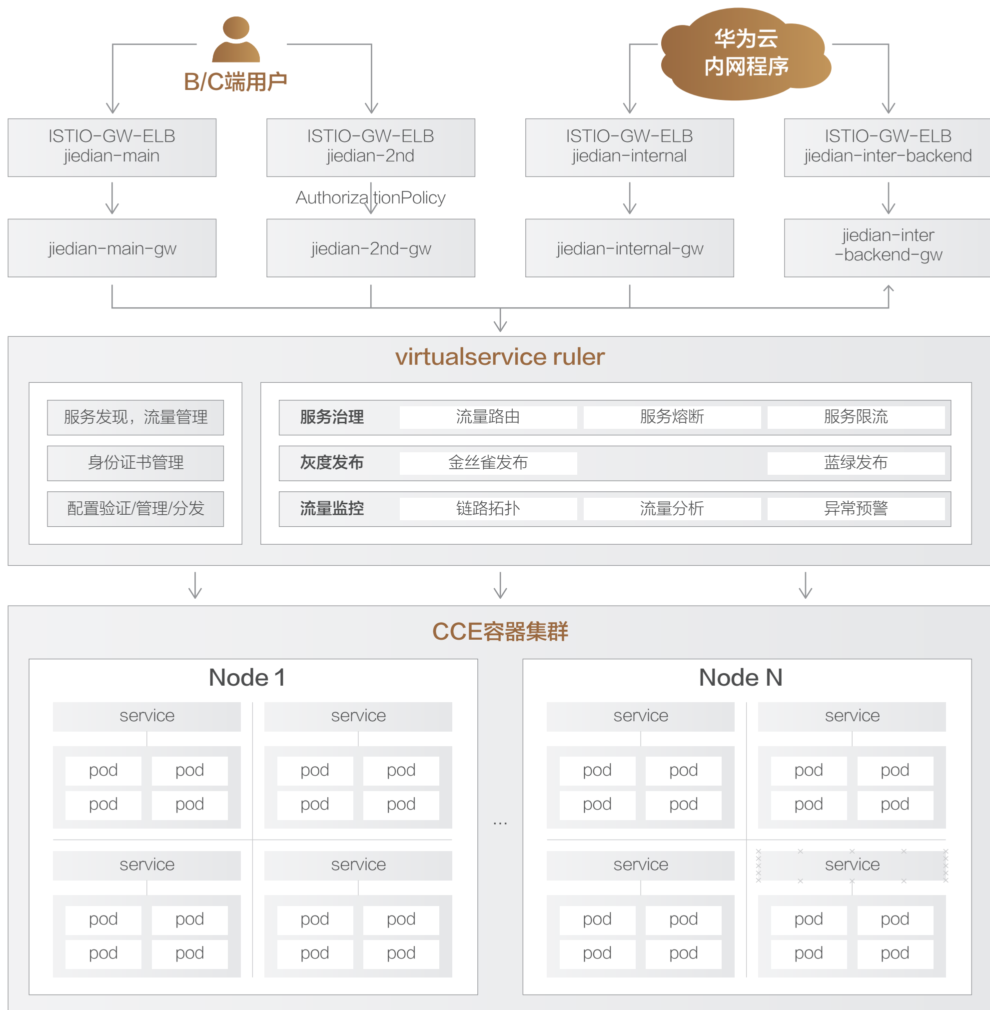
服务部署与访问

服务网格是分布式应用在微服务软件架构之上发展起来的新技术，目的在于将微服务间的连接、安全、流量控制和可观测等通用功能转化为平台基础设施，实现应用与平台基础设施的解耦。意味着开发者无需关注微服务相关治理问题，聚焦于业务逻辑本身，提升应用开发效率。换句话说，因为大量非功能性从业务进程剥离到另外进程中，服务网格以无侵入的方式实现了应用轻量化，在开源架构中服务网格典型流程如下：



Istio Architecture

通过华为云ASM我们了解到，应用服务网格是一种高性能、高可靠和易用性的全托管的服务网格，支持虚拟机、容器等多种基础设施，支持跨区域多集群服务的统一治理。以基础设施的方式为用户提供服务流量管理、服务运行监控、服务访问安全以及服务发布能力。ASM控制面和数据面均和开源ISTIO完全兼容，无缝对接华为云的企业级kubermetes集群服务容器集群引擎CCE。由于完全兼容开源的ISTIO，街电服务从腾讯云TKE和ACM迁移到华为云的CCE和ASM基本是实现零修改，可以说是平移。街电业务在ASM和CCE中运行架构如下：



通过上面的流程可以了解到，街电服务在ASM上配置了Header、Cookie等请求信息实现部分服务基于内容灰度，基于流量比例灰度规则，结合权重比例进行灰度流量分配，最大限度地实现服务优雅升级和切换。街电服务通过ASM的七层和四层连接池管理，配置传输层最大连接、重试、响应和等待，实现服务熔断、限流、降级、超时，一定程度上加强了服务稳定性和健壮性。

另一方面，通过istio的AuthorizationPolicy 结合ASM，从网关层面实现了用户白名单限制，确保内部项目只对内开放。

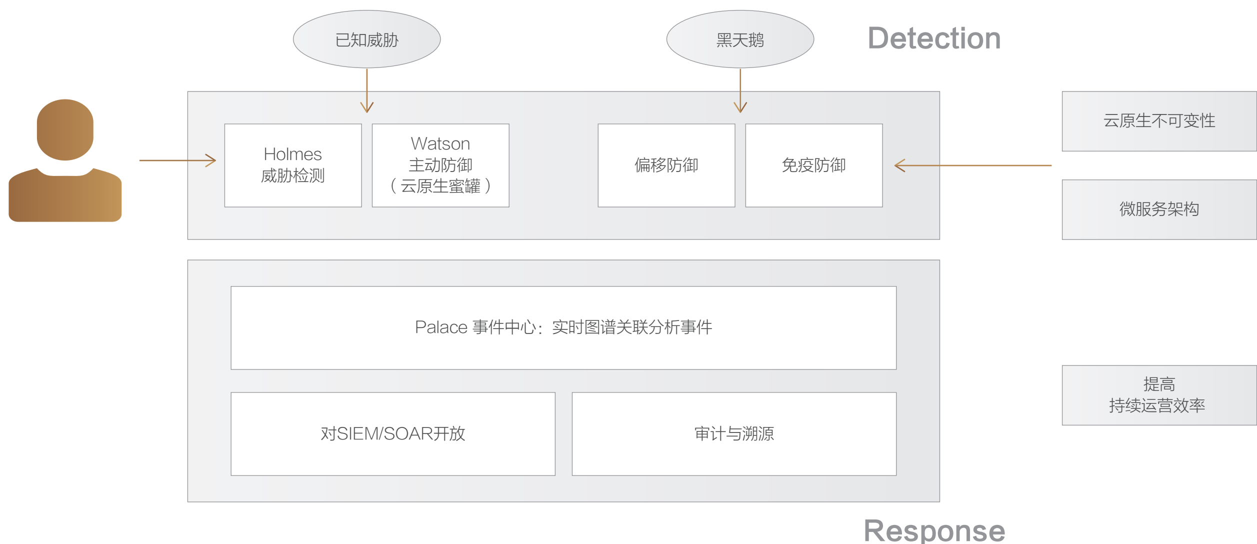
四、总结

能够提高交付速度和效率这是衡量一个团队能力的重要指标，我们选择devops并紧紧结合华为云SWR和CCE，进一步整合云能力；能够实现业务级别故障自愈和服务化能力是衡量一个项目稳定的重要手段，我们选择业务微服务化并紧密结合华为云ASM和CCE，进一步实现云健壮。未来的软件一定是生长于云上的，这是云原生理念的最核心假设。相信在华为云云原生技术的发展下，街电在云原生的实践将会越来越细化、越来越得心应手。

探真科技：云上原生的“右移”安全实践

李祥乾 探真科技研发副总裁

前文提到，左移安全能够在应用被交付到运行时环境之前，通过系统的检测与修复/改进来提高运行时工作负载整体的安全水位，降低运行时风险。然而，其并无法完全避免运行时安全风险的产生。探真科技的检测响应正是针对已知威胁与未知威胁提供了完整的检测与响应体系，旨在在事中与事后及时检测与阻断攻击威胁，并提供完善的时候复盘与溯源机制，整体提高安全运营效率。



探真科技的运行时检测响应具有以下独有的优势与特点：

1. 基于eBPF高效的检测机制，构建了基于专家规则的覆盖了从主机到容器的全面的Holmes攻击威胁检测机制。包括：逃逸，提权，挖矿，数据泄露等，与ATT&CK框架对应。
2. 基于云原生架构国内独家的云原生蜜罐系统：Watson主动防御。
3. 基于云原生架构的两个特性：不可变性与微服务架构，构建了针对未知威胁的有效检测与针对高危行为的实时阻断机制。
4. Palace事件中心基于广泛丰富的安全威胁信号检测，结合实时的大数据流图谱分析，降低信号的噪音，突出真正的安全威胁，提高安全运营效率。

Watson 主动防御：全新的云原生蜜罐

相比传统的防火墙技术和入侵检测技术，蜜罐技术更加主动和隐蔽，蜜罐的主要优势在于能诱导和记录网络攻击行为，阻止或延缓其对真正目标的攻击，而且能记录攻击日志，便于审计和回溯。 实际环境中通过蜜罐可以优先体现产品检测能力，结合产品其他能力提前于真实环境对攻击进行分析和拦截，以及对攻击者进行溯源，阻拦其对真实环境的后续攻击。

相比于传统蜜罐，云原生蜜罐具有更好的诱导效果与隐蔽性，体现在：

1. 诱饵容器按需弹性部署，隐藏于集群中，可以被弹性调度到任意节点。
2. 诱饵容器的pod可以以独立的工作负载存在，或者以sidecar容器注入到业务正常的工作负载之中。

云原生的新特性使得探真科技的Watson系统具备了主动与被动的诱捕能力：

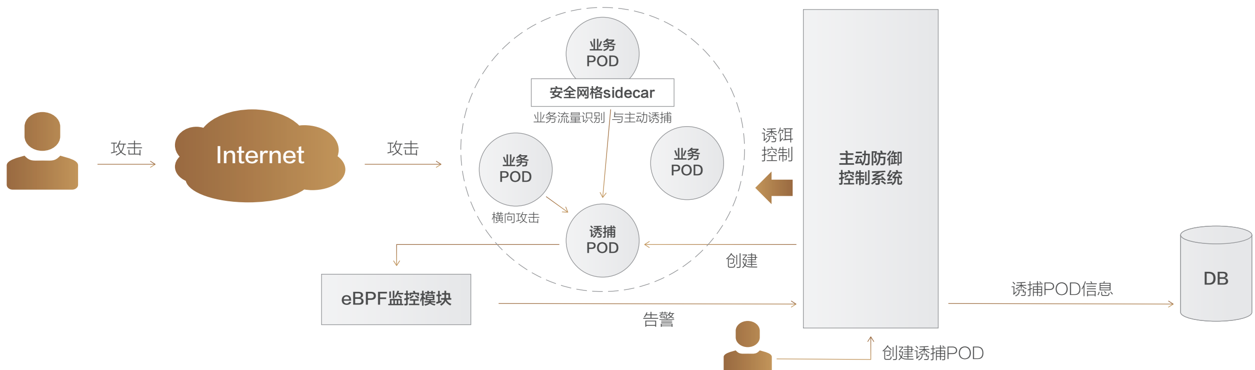
1. **被动诱捕：**系统具备多种类型的包含了不同类型漏洞的丰富的诱饵镜像。其具备被动诱捕的能力，攻击者主动发现了潜在的可被利用的攻击点，吸引攻击者横向移动与利用攻击。
2. **主动诱捕：**探真科技独有的安全网格，可以在透明、无感知的情况下，识别具备攻击威胁特征的东西向流量，并主动将其导流到特定的诱饵容器，达到主动诱捕的效果。

探真科技基于eBPF的检测能力构建了针对诱饵容器有效的攻击者行为检测能力，并提供全面的针对容器内的系统调用，文件读写，命令执行，网络调用等全面的行为捕捉能力，针对诱饵容器进行高敏感度的全面监控，提供事中的决策判断与事后的溯源分析。

为了降低诱饵容器可能对攻击者带来的可利用的攻击威胁并继续横向移动，为用户的集群带来风险，探真科技提供了以下安全保障：

1. 诱饵镜像本身经过了严格筛选，且业务单一，经过安全专家评估，尽可能避免了其它可能造成风险的组件的引入。
2. 蜜罐存在于内网，攻击者触发蜜罐报警是存在一定权限基础上，不对公网暴露，也不会引入外部攻击风险。
3. 提供了“只进不出”的网络隔离配置，可开启。攻击者无法通过网络横向移动到其它容器。

探真科技提供了丰富的诱饵镜像类型，并且具备高度灵活与高度隐蔽性的诱饵容器部署机制，结合主动与被动结合的诱捕机制，构建了整体的主动防御体系。



云原生不变性保证：有效检测和阻断未知风险行为的防御机制

传统的攻击威胁检测，依赖由安全专家持续跟进新的攻击威胁，并分析产出新的检测规则，通过在线或者离线更新的方式更新到检测系统。传统的方式有如下的弊端：

1. 无法针对零日威胁，未知威胁进行有效的检测。
2. 无法针对确定性危害的行为进行阻断。只能依赖安全人员及时响应安全事件并做对应的人工处置。

探真科技利用了云原生的两个显著特性：

1. 不可变性
2. 微服务架构

设计了一整套结合了DevOps变动流程的自适应的未知威胁防御机制：偏移防御与免疫防御。那么，具体来说，如何利用不可变性的特性来做到针对未知威胁的检测，以及阻断呢？

云原生不可变性与微服务架构带来的价值

我们针对软件漏洞修复这个场景举一个例子。我们上面也有提到，传统安全的漏扫中，通常会通过批量执行脚本的方式针对众多主机上的软件漏洞进行批量的修复和版本的升级。所以我们一般意义上认为，传统架构下的基础架构是可变的，任何一个时间点，即使是同一个服务的不同工作负载，它们的运行环境，版本，配置，无法保证是一致的。即使版本是一致的，运行环境里的软件版本也可能存在非显著的差别，这些差别可能不影响业务的正常运转，但是却产生了不确定性。

在云原生架构下，对于漏洞修复的最佳实践是，非常不鼓励运行时在进入容器去更新软件版本，引入新的软件，原因如下：

1. 严重破坏了不可变性。每个工作负载的状态变得无法预测。同样定义的工作负载的行为无法预测。
2. 容器的规模指数级高于传统主机规模。主机是物理存在的，或者是需要较大创建成本去创建的虚拟机/云主机。其创建和修改成本较大。而容器是轻量级的隔离与虚拟化技术，创建成本极低。所以一个主机上理论上就可以部署几百个容器，并且可以频繁的销毁与创建。去通过批量脚本方式进入容器修改软件版本是极其低效的方式。云原生更主张修改镜像的Dockerfile，来批量对基于同样镜像构建的工作负载进行批量升级，来维护不可变性的同时以最高效率、最合理的方式完成任务。这也是Infrastructure as Code实践的最基础落地实践。

云原生的不可变性为云原生安全提供了新的可能性。同样的，微服务架构也是如此。为什么这么说呢？

因为微服务架构下，传统的单体架构按照业务与团队分工拆解成为了多个微服务。每个微服务是一组工作负载和一组镜像。因为业务功能更加专注于聚合，其容器运行时的行为会更加的内聚，降低耦合性，其行为具备更好的可预测性。

而因为有了云原生不可变性，同一个服务的不同的工作负载之间的差异被最小化了，运行时的不确定性也最小化了，进一步提升了容器行为的可预测性。

基于以上的两种特性，我们在加速的DevOps流程中，引入了偏移防御与免疫防御。

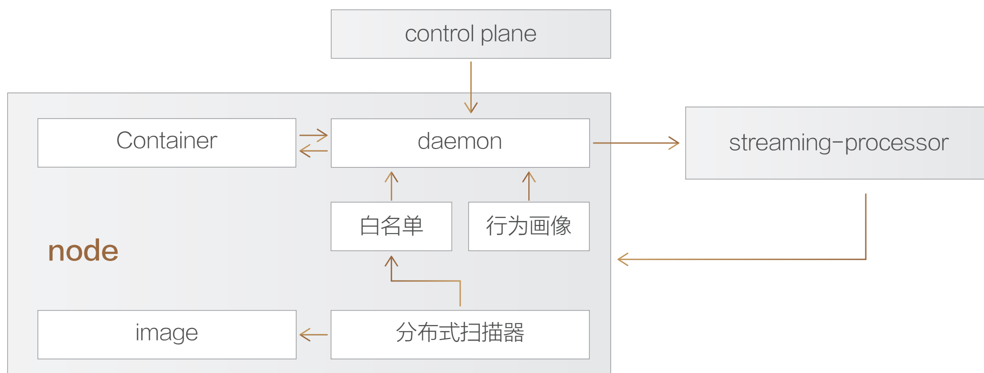
» 偏移防御基于我们专利的分布式扫描技术，在本地对容器所基于的镜像完成扫描，基于工作负载的Yaml配置，生成其可执行文件的白名单。基于白名单，在运行时对偏移白名单的可执行文件的执行进行检测，根据策略配置进行预警，或者进行阻断。
» 基于不可变性构建的偏移防御在达到对未知威胁的极强的检测能力的同时，还达到了非常好的准确率和可大规模落地的优势。

然而，偏移防御存在以下的缺陷：

» 偏移防御对于已经包含在镜像中的可被利用的风险存在检测与阻断的缺陷，同时对于执行可执行文件的参数列表无法进行固化。左移安全一定程度上可以降低这个问题的风险，但是无法避免。
» 偏移防御只能对可执行文件的执行进行预警和阻断。

基于这样的缺陷，我们提供了近一步的免疫防御能力：

» 免疫防御提供了运行时基于镜像与配置的变化进行在线的持续的学习，以及基于统计算法与无监督模型的算法进行行为画像的学习。基于产出的行为画像进行对不满足画像的未知行为的检测与阻断。行为画像支持3种类型：文件读写，带参数的命令执行，网络调用。
» 运行时的地持续学习基于镜像版本与运行配置的变化自动触发，增量学习。



针对不变性保证带来的防御能力显著提升

上述的能力对运行时整体的检测响应能力带来了质的飞跃，具体体现在：

1. 针对运行时威胁的直接阻断的能力。
2. 极好的准确率与对攻击威胁的检测，使其具备大规模的落地可能性。
3. 对于攻击者的攻击链路的必经步骤极好的阻断能力以及非常难以绕过的特性，具备极好的实战性。

上述结论在客户的专业攻防团队与我们的攻击防御的对比测试中，达到了极好的检测率，与少有的阻断响应能力。并且已经在多个客户的复杂场景下大规模落地。

检测能力强大是不够的，需要高效率的安全运营方案

安全产品提供了高可复制性的安全解决方案，然而这些解决方案需要以高效率、可行、低摩擦的方式正确的运营，才能给一个企业带来真正的价值。

对于甲方安全人员通常面临的困境就是，产品过多地关注高召回，高效与强大的检测能力，却忽视了，安全团队如何处理这些安全事件，如何构建出一个长期可持续的运营模式，能够长期高效率地应对攻击威胁。

对于检测事件来说，降低噪音，提高效率是重中之重。否则，爆出几千个告警，就会把真正的攻击威胁淹没，虽然可以说是可以召回，但是在实际的运营中，却无法真正的解决问题。

探真科技采取了一种与众不同的路线：

1. 在每个节点的DaemonSet，我们使用了eBPF高效率地检测出了众多系统事件与可疑行为，经过了较为宽松目标是高召回的规则初筛之后，统一收集到后端进行实时数据流的数据分析。通过我们的数据分析能力进行实时图谱的关联分析。
2. 通过实时的关联分析能力，降低安全事件流的噪音，提升安全事件的准确率，并且提供了一定程度实时的攻击链路复原能力。

基于这样的路线，探真科技具备了多种类型的关联分析能力，并且随着产品版本迭代，其能力将会持续加强与优化。

开放能力与生态

探真科技的产品永远不会是封闭的产品，构建篱笆围起来，去大包大揽。探真科技希望基于我们的技术优势与特色解决客户的安全痛点与运营问题。同时，具备集成与被集成的能力，以期望与合作伙伴一起为客户提供最大的真实价值，更好地解决客户的痛点和问题。

具体来说，我们通过几种方式来实现这样的能力：

1. 探真科技提供了面向Developer的Open API，产品具备的能力，通过Open API也可以编排与调度；产品不具备的能力，Open API也能提供更加灵活与丰富的可编排调度能力。
2. 探真科技提供了可配置的webhook，可以回调客户的其他平台的开放能力，形成更好的继承能力。
3. 探真科技与探真的合作伙伴共同营造生态能力，为客户提供更高的价值。

探真科技不仅仅提供更好的产品，也会提供更好的面向开发者（To Developer）的产品。

没有最佳实践，只有更佳实践

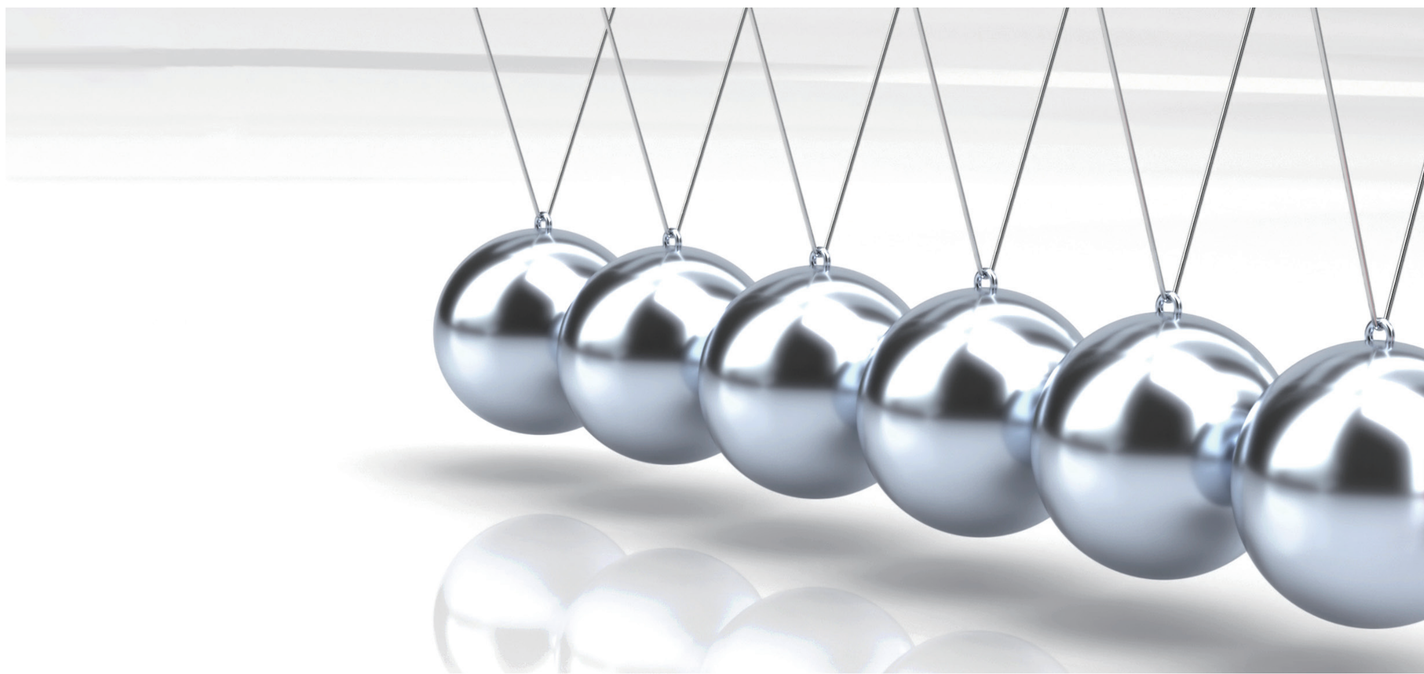
我们为云原生场景下的DevOps提供全生命周期、全流程的安全保证，同时，这种安全能力能够为用户带来低摩擦的真实的价值与体验。这些体验不会为伙伴团队，比如开发团队，运维团队带来痛苦，带来阻力，而是为整体的DevOps带来真实的、可落地的价值。这是探真科技一直贯彻的理念与价值观。探真科技秉承着这种理念，持续迭代产品，持续创造更佳的实践。

5

云原生 Cloud Native

人物专访

云原生底座之上，顺丰智慧供应链领跑的秘密 ------------------------------ 97-100



云原生底座之上，顺丰智慧供应链领跑的秘密

假设你是一个大闸蟹的商家，在即将到来的旺季，肯定是既喜又忧，既希望能利用机会打一个销售的漂亮仗，又担心物流的瓶颈，影响用户体验。大闸蟹是一个特殊的商品，既要在运输过程中保活，又要确保用户对时效性的需求，这一切对物流的仓储和配送都极具挑战。如何解决这个挑战？在创原会 “千行百业共话云原生”栏目采访顺丰科技大数据总监蔡适择的过程中，我们找到了答案，那就是：数字化。

上面提到的大闸蟹商家所遇到的挑战，比起顺丰来小得多。顺丰这么大规模的线路和车辆，如何进行规划和运营？每年的618和双11大促，业务高峰超过平时几倍甚至几十倍，如何进行高效的资源保障？数据分散于各个独立的组织中，彼此共享数据困难，如何打破数据孤岛？这些都是看似不能解

决的难题。正是在这样的背景下，顺丰科技聚焦用科技改变物流体验，将顺丰带向数字化转型的深水区。

“顺丰高速发展的背后，各领域的数据问题成为精细化运营的困境，主要体现在几大核心业务线跨领域数据共享困难、数据服务时效性不足、数据质量问题、分析口径不统一和缺乏对数据的深度分析和挖掘。” 顺丰科技大数据总监蔡适择这样说。

数据是数字时代的石油，也是数字化、智能化的基础，因此，这样的数据孤岛是顺丰实现数字化转型首先要解决的问题。由此，顺丰科技启动了数据中台项目，以打通内部数据的使用效率。这是一个强大的数据中台。据蔡适择透露，顺丰科技的数据中台全景图“1+1+N+X”体系，其中一个物流数据底盘即基于第四代大数据框架的云原生实时数据湖，产品整体包括：弹性资源层、融合计算层、数据洞察层和安全中心，可以提供更简单、更安全、更高效的数据湖服务。

可以说，云原生实时数据湖在顺丰科技的大数据平台中扮演了重要作用。顺丰科技的云原生实时数据湖充分展现出云原生的核心特性，具有存算分离、实时数仓、湖仓一体三大核心能力。例如，在面对618和双11大促业务高峰时，存算分离就展示出了其巨大优势，传统大数据平台是存算融合，无法进行资源的定向伸缩，会带来很大的资源浪费，而云原生数据湖的存算分离，从根本上解决了存和算独立伸缩的问题，业务高峰时可以按需获取公有云算力等资源，高峰过后，资源就能释放，资源利用率从平时的50%提升到了80-90%！顺丰科技找到了用云原生破解物流供应链难题的钥匙。

智慧物流供应链进化论

现在有了云原生大数据平台，上面提到的大闸蟹物流难题也迎刃而解了。在物流配送过程中，顺丰科技云原生大数据平台充分发挥“提前预测、实时规划”的能力，可以对每个环节中收派件的时间和空间维度精准记录分析，帮助快递员建设起精细化、智能化排班，以及实时调度分配的信息管理体系。同时，通过件量预测、分仓管理、路线规划等数据分析，为配送商品精准匹配物流小哥、运输车、飞机等，预测判断哪条线路的运输效能最优，实现物流领域的全民数字化管理和智慧决策过程。

其中的路径规划，体现得更为明显。上面提到，顺丰拥有庞大的干支线线路、车辆，要实现人、车、货的精准匹配和智能调度，单靠人工规划极其困难。而基于云原生大数据平台，加上大数据算法，顺丰的车辆调度系统实现路径规划和车辆调配的智能化。据顺丰科技相关人士透露，这种智能规划比人工规划可以提升约10-30%的车辆利用率。

上面这些还只是顺丰智慧供应链的冰山一角，我们不妨化身一个化妆品商家，看看一瓶化妆品从用户下单到最终收到货的过程，就能更清晰地看到其中的痛点以及解决过程。当用户下单购买一瓶化妆品时，一定希望能有好的购物体验：尽快收到货，而且货的准确性、完整性都足够好。站在商家的角度，为了实现这一点，就需要进行销量预测，根据不同区域消费者行为习惯，预测不同区域的销量，将合适的商品调配到距离消费者最近的仓库。而且，化妆品很大程度也属于快消品，商家还需要规划不同仓库中的货品数量，以免商品过期或者跟不上消费需求。而且，商家还需要有好的补货策略等等。以上行为，对于传统商家来说都是痛点，因为靠人工规划不仅仅是效率低，实际上已经行不通了。这时候，顺丰的智慧物流供应链就有了用武之地，无论是消费者画像分析，还是机器学习销量预测，或是智能库存补货，以及智能仓库选址、仓库之间的库存调拨策略、线路规划等，大数据技术都能扮演重要角色。

具体来说，有了智慧物流供应链，一瓶化妆品的旅程是这样的：当你下单前，大数据平台已经根据区域的消费习惯，洞察到了区域可能购买的行为，提前将化妆品调拨到了你附近的仓库，当你下单后，配送员会迅速取货、极速送达，在配送过程中，路线规划也是智能的，既保证配送时效，又保障综合配送成本做到最低。

更重要的是，有了大数据平台，整个物流供应链就变成了会思考、能进化的智能生命体，可以不断生长。据顺丰科技相关人士透露，顺丰的智慧供应链是基于运筹优化技术的应用，在持续不断的优化。为了更好满足市场变化和消费者对履约时效的需求，顺丰的供应链在持续优化中转场和网点的选址以及班次规划，优化前置网点布局、运输时效，为客户提供更优质的供应链物流服务。

后记：云原生的底盘

在调研顺丰科技的云原生大数据平台过程中，我们深刻感受到了云原生所扮演的底盘作用。顺丰，用数据化实现人、车、货的智能调配，大数据平台起到了重要的支撑作用。这时候，“上云”不是指把底盘直接搬到云上，而是要让底盘云化、对应用透明，即云原生化。信通院调查数据显示，超过四成用户已经将云原生用于核心生产业务，顺丰就是其中的领跑者。用蔡适择的话来说，“我们的技术会升级换代，每次升级，不可能将整个底盘重新刷一遍。云原生化后，可以理解为体系变成了操作系统+软件的模式，操作系统不用随着技术升级换代而改变，我们只需要做软件更新就可以了。”正是这种云原生的稳固底盘，不仅大幅降低了顺丰的IT开发和运维成本，也让顺丰科技可以将精力集中在真正助力核心业务的工作上，实现资源高效、应用敏捷、业务智能。

同样，顺丰科技的案例也体现出云原生的另一个精髓：“一切皆服务”。据了解，在云原生领域，顺丰科技和华为云开展了深度的技术合作和联合创新，推动应用的云原生改造。蔡适择表示，“华为云面向全行业提出了云原生2.0的理念，为顺丰的大数据平台提供了很贴心的服务，让我们能够轻松获取云原生能力，可以像操作系统一样，在上面开发出任何自己想要的能力。” 换句话说，顺丰科技云原生大数据平台的背后，有着华为云“基础设施即服务”的助力。

同样，借助云原生，顺丰科技也在加速实现“经验即服务”，将自己智慧供应链的能力和数字化转型的经验开放出来，以服务的方式赋能千行百业。华为云Marketing部长董理斌在采访过程中表示：云原生技术与大数据的结合，带来了物流行业的诸多的变化；华为云将秉持一切皆服务的宗旨，与顺丰一起，将云上的能力、云上的技术、云上的服务赋能给更多物流企业。

今年6月28日，顺丰云原生实时数据湖通过信通院数据平台建设服务商能力成熟度评估，专家集中评审认为顺丰在资源评估、规划设计、实施方案、实施执行、实施验证、交付、资源运维、运维管理等能力都合乎企业级大数据服务要求，这是对顺丰云原生实时数据湖的权威认可，也是顺丰大数据服务市场化过程的里程碑。目前，顺丰科技大数据平台已经在零售、冷链、医疗、金融、农业、食品等多个领域得到应用。

据了解，顺丰科技可以提供从采购到消费者末端交付的一体化智慧供应链解决方案，包括传统的仓储管理、运输管理软件，以及基于企业经营数据的大数据平台等。基于这些大数据及数据处理能力，一方面能够帮助企业建立智慧化经营的底层平台，另一方面可以提供基于物流供应链的驾驶舱，能够在物流供应链方面做到一体化的管理、有计划的管理、可视化管理。

在顺丰，云原生不仅掀起了一场技术革命，更是掀起了一场生产力革命与公司模式的革命。顺丰，以云原生为底盘将智慧供应链推进到新高度，跨越了数字鸿沟。同样，自身也在基于云原生加速成为千行百业数字化转型的底盘，迈入增长的新范式。

聚八方领航者 论云原生之道

与行业技术精英

共创云原生的无限可能