阿里云 | 云原生应用平台 CLOUD NATIVE APPLICATION PLATFORM | 阿里云开发者社区 ALIBABA CLOUD DEVELOPER COMMUNITY

智算时代的容器技术演进与实践

2023云栖大会容器服务 ACK 分享实录合辑

容器服务 ACK 如何加速现代化应用平台构建

- 容器技术与产品最新趋势
- 容器 AI 工程化探索与实战
- 企业大规模应用实践案例

容器服务 Kubernetes 版 ACK

容器镜像服务 ACR

分布式云容器台 ACK One

服务网格 ASM

阿里云云原生公众号

阿里云云原生视频号

ACK 产品免费试用

阿里云开发者“藏经阁”
海量电子手册免费下载

导论

今天，能想到的或是想不到的领域，对容器和 Kubernetes 的需求都居高不减，使这项技术正在真正走向无处不在。

从 2015 年正式对外提供服务至今，阿里云容器服务产品家族已经成长为企业的云原生应用操作系统，帮助越来越多的客户实现智能化、数字化创新，包括自动驾驶、智能科研、金融科技等众多新兴领域。其覆盖了从公共云、边缘云、到本地数据中心的各个场景。让所有需要云能力的地方，都有统一的容器基础设施。

2023 年，阿里云容器产品能力持续受到业界的广泛认可。2023 年 9 月，在权威咨询机构 Gartner 发布的容器管理魔力象限中，由于在公共云、专有云、混合云等环境完善的产品体系，阿里云成为全球领导者，亚洲唯一。在 2022 年 4 季度，Forrester 公共云开发与基础设施平台 Q4/22 评测中，阿里云是中国云原生开发者的最佳选择。

智算时代已来。正如一个文明社会的科技水平取决于其对能源的利用能力，企业的智能化水平取决于其对算力的利用能力。云计算为智算时代带来无限可能，2023 年云栖大会上，阿里云容器服务宣布了以加速企业构筑现代化应用平台、最大化利用阿里云强大弹性算力为使命，在高效云原生算力、高性能智算应用、智能化运维管理、可信基础设施、分布式云架构 5 大核心方向带来的产品能力全新升级。

本书精选 2023 云栖大会中“容器技术与服务”专题分享精华，集合容器服务产品家族最新发布、容器 AI 工程化创新、容器前沿技术与大规模生产实践、典型场景企业案例等方向内容，希望能够帮助您了解如何基于容器技术与服务，拥抱智算时代，为现代化应用构建加速！

目录页

第一章

容器产品最新发布

阿里云 ACK 新升级，打造智算时代的现代化应用平台

作者：易立，阿里云研究员&容器服务负责人

今天，能想到的或是想不到的领域，对容器和 Kubernetes 的需求都居高不减，使这项技术正在真正走向无处不在。

在 2023 云栖大会上，阿里云云原生产品线容器服务负责人易立关于容器服务 ACK 在本届亚运会上应用的介绍，让现场观众眼前一亮，“以杭州亚运会为例，作为云原生技术底座，为亚运一站通、亚运钉等众多核心应用提供了高弹性、高可用、异地多中心的架构支持，确保了赛事系统万无一失。”

阿里云容器服务 ACK 已经成长为企业的云原生应用操作系统，帮助越来越多的客户实现智能化、数字化创新，包括自动驾驶、智能科研、金融科技等众多新兴领域。其覆盖了从公共云、边缘云、到本地数据中心的各个场景。让所有需要云能力的地方，都有统一的容器基础设施。

在过去一年，阿里云容器产品能力持续受到业界的广泛认可。2023 年 9 月，在权威咨询机构 Gartner 发布的容器管理魔力象限中，由于在公共云、专有云、混合云等环境完善的产品体系，阿里云成为全球领导者，亚洲唯一。在 2022 年 4 季度，Forrester 公共云开发与基础设施平台 Q4/22 评测中，阿里云是中国云原生开发者的最佳选择。

智算时代已来，易立介绍了为助力企业构建现代化应用平台，阿里云容器服务在高效云原生算力、高性能智算应用、智能化运维管理、可信基础设施、分布式云架构 5 大核心方向带来的产品能力全新升级。

1. 新一代云原生算力，提升企业计算效能

更大规模：弹性算力池新突破

阿里云提供了丰富的弹性算力，包括 Intel/Amd/倚天 Arm 等多 CPU 架构，GPU/RDMA 等多种异构加速器，以及按量、Spot、节省计划等多样化的售卖形态。使用 ACK，客户能够最大化利用阿里云整体弹性算力池能力，根据自己的需求灵活选择，增效降本。

ACK 集群支持托管节点池、虚拟节点两种不同的数据面形态：

托管节点池，支持任何 ECS 裸金属和虚拟机实例作为 K8s 工作节点，一个工作节点可以运行多个 Pod，全兼容 K8s 语义，兼具灵活性与易用性。

虚拟节点，每个 Pod 运行在独立的弹性容器实例 ECI 之中。每个 ECI 实例是一个独立安全沙箱，具备高弹性、强隔离，免运维等特点。阿里云弹性计算基于 CIPU 可以统一生产 ECS 裸金属实例、虚拟机实例和弹性容器实例。这意味这 ECI 支持弹性计算丰富的算力类型，具备充足的库存保障。

今年 ACK 集群通过与弹性计算调度相互感知，可以更好调度 ECI 实例，支持将 K8s 对集群资源调度能力扩展到整个弹性算力池，确保了 ECS 节点池与虚拟节点的调度统一和能力一致，用户无需修改现有 K8s 应用定义即可最大化使用云资源。

越来越多的客户基于 ACK 集群，构建大规模微服务架构应用和大规模数据计算任务。同时为了满足对集群规模增长的诉求，ACK 单集群最大支撑的节点从 10000 提升至 15000，ECI 实例从 20000 提升至 50000 实例。我们的控制面组件会根据数据面规模按需伸缩，保障稳定性。

更优性价比：倚天架构专属优化

越来越多的 ACK 客户选择倚天芯片作为新算力选择。客户选择倚天架构实例主要有如下三个原因：

- 高性价比：相比 G7 实例族，Web 应用提升 50%，视频编解码提升 80%，Spark 任务提升 28%。
- 高吞吐：采用 Arm V9 架构，提供独立物理核心，提供更确定性的性能；相比 G7 实例族，Web 应用吞吐提升 22%；Spark TPC-DS Benchmark 速度提升 15%。
- 专属优化：容器镜像服务 ACR 联合基础软件团队、龙蜥社区在制品中心，提供了面向倚天芯片专属优化的基础软件及应用软件镜像。通过基于 AI 和专家知识库的 KeenTune 为倚天架构提供专项参数调优。在主流场景中，优化后相比优化前性能提升 30%。

为了支持容器应用向倚天架构平滑切换，ACR 提供了多架构镜像构建能力，支持一份源码构建出包含 x86、Arm 架构的应用镜像，同时 ACK 集群可以同时包含 Arm/x86 节点池

或虚拟节点，让客户 K8s 应用在不同 CPU 架构下按需调度，逐步切换。

更高弹性：全新发布节点池即时弹性能力

最大化利用云的弹性能力是客户对容器产品的重要诉求，易立也带来了 ACK 的一项全新发布：“在阿里云上，容器服务每天有数百万核的算力资源按需扩缩容，帮助客户优化计算成本。今天，我们正式发布 ACK 节点池即时弹性能力”。

ACK 节点池即时弹性 Scaler 拥有以下特点：

- 更快的弹性速度：在 100 节点池的规模上，保持平均 45s 的端到端扩容速度，相比社区 Cluster Autoscaler 提升 60%。
- 支持用户定义灵活的规格匹配策略：在社区的 Cluster Autoscaler 中，每个节点池中节点 CPU/Memory 规格是固定的，如需满足不同需求需要创建多个节点池，会带来配置管理复杂性、资源碎片引入的可能，并增加由于库存不足导致弹性稳定性降低的风险。即时弹性 Scaler 支持用户定义灵活的规格匹配策略，不同机型节点规格匹配条件下，系统会根据待调度的 Pending Pod 集合的资源请求和调度约束，及对 ECS 的库存感知，生成优化的装箱结果。这样，只需一个节点池就可以完成对多规格、多可用区

的节点弹性。在降低节点池配置复杂度的同时，减少了资源碎片，提升了弹性的成功率。

即时弹性完全兼容现有节点池能力和使用习惯，可以配合托管节点池实现节点的自动化运维。

更简运维：ContainerOS 与全托管节点池结合

对于 K8s 集群，节点运维是保障系统稳定性与安全的重要日常工作，但是手工操作非常复杂繁琐。ACK 托管节点池支持节点的全生命周期自动运维，包括 CVE 高危漏洞自动修复、节点故障自愈、OS/ 节点组件自动升级，其中节点自愈成功率 98%；集群节点运维时间减少 90%。

ContainerOS 是龙蜥社区发布的面向容器优化的操作系统，采用不可变基础设施理念构建，具备精简、安全、可编程等特点。千节点弹性时间 P90 55s，相比 CentOS 等节点弹性时间降低 50%。ContainerOS 与全托管节点池可以完美结合，进一步优化了节点池的弹性和可运维性，让企业聚焦在自己的自身业务，而非 K8s 基础设施维护。

更丰富场景：Serverless 容器为 AI 场景增效降本

对 Serverless Container 的支持是 K8s 演进的重要方向，基于 ECI 的 ACK Serverless 在客户场景中得到了广泛的应用。ACK、ECI 不但帮助微博热搜，钉钉会议等在线应用的弹性伸缩，也在助力众多 AI 和大数据客户降本增效。

深势科技基于基于 ACK 与 ECI 实现多地域部署 AI 科研平台，免运维，按需创建实验环境 ，支持大规模 AI 镜像秒级拉取，资源利用率提升 30%。

米哈游基于 ACK 与 ECI，统一全球各区服大数据平台架构，单日创建 200 万 以上 ECI 实例执行 Spark 计算任务。通过高效利用 ECI Spot 实例，整体资源成本下降 50%。
今年 ECI 弹性容器实例有四个重要发布：

- 普惠降本：新增「经济型」规格，相比当前通用型价格下降 40%，面向成本敏感的 Web 应用、计算任务、开发测试等工作负载。此外现有通用型实例价格也将在近期下调，最高下降 15 %。

- 极致性能：计划新增「性能增强型」规格，面对计算密集型业务场景，如科研、高性能计算、游戏，相比现有通用型实例，提供更高性能的算力、更具确定性的性能。

- 弹性加速：ECI 通过对用户负载特征自学习和预测，实现底层资源的预调度，扩容速度

提升至 7000 Pod/min，非常适于大规模数据任务处理场景。此外业界首家支持 GPU 驱动版本选择，为 AI 应用提供更多灵活性的同时，冷启动提速 60%。

- 灵活提效：ECI 今年发布了对倚天 Arm、AMD 架构的支持，ACK 也在近期上线了 Windows 容器支持，支持更加丰富的企业应用场景。并且发布对细粒度内存规格支持，帮助用户精细化资源适配，消除空闲资源开销。

2. 云原生智算基础设施，构筑高效现代应用平台

全面支持灵骏集群，为大模型训练提效

过去一年，AIGC/ 大语言模型无疑是 AI 领域最重要的进展。随着大模型参数规模、训练数据和上下文长度的增长，训练大模型所消耗的计算量呈现指数级增长。ACK 全面支持阿里云灵骏智算集群，为大规模分布式 AI 应用提供高性能、高效率的 Kubernetes 集群。

ACK 提供了对灵骏高性能算力的全面支持，以及批量 AI 任务调度，数据集加速，GPU 可观测与自愈等能力。通过软硬件协同设计与云原生架构优化，ACK 助力 PAI 灵骏智算方案高效利用强大的算力，为 AIGC、自动驾驶、金融、科研等众多智算业务场景提效。

ACK 云原生 AI 套件增强，构筑企业专属 AI 工程化平台。

ACK 去年推出云原生 AI 套件，帮助用户基于 Kubernetes 充分利用阿里云上弹性算力，支持弹性训练与推理等场景。在此之上既服务了阿里云 PAI、灵骏智算、通义千问等 AI 平台与服务，也提供对开源 AI 框架和模型的容器化支持。

今年，针对大模型场景，AI 套件新增了对开源大模型框架 DeepSpeed，Megatron-LM，TGI 的容器化支持与优化。通过云原生 AI 套件的调度优化与数据访问加速，AI 训练速度提升 20%；大模型推理冷启动速度提升 80%，数据访问效率提升 30%。ACK AI 套件已被广泛应用于众多海内外企业，帮助客户构建自己专属的 AI 平台，显著提升 GPU 资源效率和 AI 工程效率。

- 国产 AI 绘画工具「海艺 AI」：基于 Fluid 数据集加速和 AIACC 模型优化方案，推理性能提升 2 倍。
- 任意门 Soul：基于 ACK 构建近千卡规模 AI PaaS 平台，开发迭代效率提升 2-5 倍。

ACK 集群调度器， 面向 AI /大数据负载优化扩展

ACK 集群调度器基于 Koordinator 项目。它是基于阿里巴巴大规模混部实践孵化出的开源 Kubernetes 调度器实现，可以统一、高效地支持微服务、大数据、AI 应用等多样化的工作负载。其中我们针对 AI、大数据负载进行了如下优化和扩展：

- 在全面兼容 Kubernetes 现有调度能力基础上提供批量任务的调度元语，如 Gang Scheduling，弹性配额、优先级调度等，可以与 Kubeflow，KubeDL 等社区项目无缝集成。
- 支持拓扑感知性能优化，根据 PCIe、NVSwitch，以及 RDMA 网卡等互联链路的拓扑信息，自动选择能够提供最大通信带宽的 GPU 卡组合，有效提升模型训练效率。
- 支持对 GPU 的细粒度资源共享调度，有效提升模型推理场景 GPU 资源利用率。

近期我们与小红书在社区合作，将发布 Hadoop Yarn 任务与 Kubernetes 负载混部的能力，进一步提升 Kubernetes 集群的资源效率。相关工作帮助小红书 ACK 集群资源效率提升 10%。

我们也在推进 Koordinator 捐赠到 CNCF 基金会，保持项目长期健康的发展，也欢迎大家在社区共建。

3. 智能自治体系，降低容器运维管理成本

ACK AIOps 智能产品助手，加速 K8s 问题定位与解决

Kubernetes 自身技术复杂性是阻碍企业客户采用的一个重要因素。一旦 K8s 集群发生故障，对应用、集群、OS、云资源的问题排查，即使对经验丰富的工程师也充满挑战。

ACK 全新升级容器 AIOps 套件，通过大模型结合专家系统的方式，让管理员可以通过智能产品助手，使用自然语言与系统进行交互，加速 Kubernetes 问题定位与解决。当问题发生时，AIOps 套件会采集上下文相关的 Kubernetes 对象与云资源的定义，状态与拓扑信息。比如 Deployment，Pod 和关联的节点等。以及相关的可观测信息，如日志，监控，告警等。然后会基于大模型进行数据分析与归集，给出当前问题的可能原因与修复方案。ACK 背后的大模型方案面对云原生开发和运维知识库进行了调优，提升了问题分析的准确度。用户可以进一步利用智能诊断中的专家经验系统，进行根因定位。

现有 AIOps 套件包含 200+ 诊断项，覆盖 Pod，节点，网络等问题场景，可以对网络抖动，内核死锁、资源争抢等问题进行深入排查。除了用户驱动的问题诊断，AIOps 套件也在加强对自动化巡检和异常事件自动化实时处理，为集群稳定性、安全提供更加全面的防

护，防患于未然。

ACK FinOps 套件全面升级，精细场景化分析与分摊策略

ACK 去年发布了 FinOps 成本管理套件，为企业管理员对 K8s 集群现了成本的“可见，可控，可优化”。在过去的一年中，FinOps 套件支持了不同行业的上百家客户，其中：

乾象投资利用 FinOps 套件，优化应用配置，集群资源利用率提升 20% 成本节省超过 10 万元/月。

极氪汽车通过 FinOps 套件实现混合云弹性降本，一年节省了数百万 IT 成本。

今年，FinOps 套件全面升级，增加了更多场景化的分析与分摊策略，例如：在 AI 场景，可以基于 GPU 卡、显存等维度进行成本可视化。此外，FinOps 套件还发布了一键资源浪费检查功能，可以快速发现集群中空置的云盘、SLB 等未被使用的资源，让集群的整体资源利用率进一步提升。

4. 端到端容器安全，为构建可信 AI 应用护航

可信化应用交付增强，ACK 与 ACR 提供 DevSecOps 软件供应链

软件供应链安全是企业落地云原生技术的最大关切，Gartner 预计到 2025 年，全球 45% 的组织都会遭受过软件供应链攻击。

阿里云 ACK 和 ACR 服务提供 DevSecOps 最佳实践，实现了从镜像构建、分发到运行的自动化风险识别、阻断与预防能力。帮助企业构建安全可信的软件供应链。

DevSecOps 的实践依赖研发、运维、安全团队的深入协同，今年，我们推出了集群容器安全概览，帮助企业安全管理员更好感知集群配置、应用镜像、容器运行时的安全风险，让供应链流程更加透明高效。

通过使用我们的 DevSecOps 供应链安全能力：著名的汽车制造商路特斯每月实现千次安全配置巡检，预防高危风险配置上线；招联金融基于供应链策略治理能力，在每日 CI/CD 流程中实现千次风险镜的拦截阻断，保障金融业务安全。

两全其美：Sidecarless 与 Sidecar 模式融合的服务网格新形态

服务网格已经成为云原生应用的网络基础设施。阿里云服务网格 ASM 产品进行了全新的升级，成为业界首个发布托管式 Istio Ambient Mesh 的产品，提供对 Sidecarless 模式与 Sidecar 模式的融合支持。

经典服务网格架构采用 Sidecar 模式，需要为每个 Pod 注入 Envoy Proxy Sidecar，实现流量拦截与转发。具备极高的灵活性，然而引入了额外的资源开源，增加了运维复杂性和与建联时延。在 Sidecarless 模式下，L4 代理的能力被移到节点上 CNI 组件中，可选 L7 代理独立于应用程序运行。应用程序无需重新部署即可享受服务网格带来的安全加密，流量控制和可观察性等功能。

在典型客户场景中，采用 Sidecarless 模型服务网格，可以减少资源开销 60%，简化运维成本 50%，降低时延 40%。

托管式 Istio Ambient Mesh 有效地降低服务网格技术复杂度，推动零信任网络技术落地。

新推隐私增强型算力，护航可信 AI 应用构建

为解决企业对数据隐私日益关切，阿里云、达摩院操作系统实验室与 Intel 和龙蜥社区一起，推出基于可信执行环境（TEE）的机密计算容器（Confidential Containers，简称 CoCo）在云上的参考架构，结合可信软件供应链、可信数据存储，实现端到端安全可信容器运行环境，帮助企业抵御来自外部应用、云平台，甚至企业内部的安全攻击。

ACK 基于阿里云八代 Intel 实例所提供的 Trust Domain Extension TDX 技术，全新推出对机密容器以及机密虚拟机节点池支持。使用 TDX 技术，业务应用无需更改，即可部署到 TEE 之中，极大降低了技术门槛，为金融、医疗、大模型等数据应用，提供隐私增强型算力。

在 AI 时代，模型和数据成为企业核心业务资产。基于机密计算容器，阿里云基础软件、容器、以及英特尔团队提供了可信 AI 应用一个演示方案。在这个示例架构中。应用、AI 模型和微调数据集都被加密存储在云端服务中，在运行时由机密容器在 TEE 中对其进行解密后执行。

- 模型推理与微调过程安全可信，保障数据的机密性与完整性。
- 高性价比，基于 AMX 指令集优化，32 核 CPU 可以实现秒级 Stable Diffusion 出图。
- 低损耗，TDX 带来的性能给损耗可以控制在 3% 以内。

5. 更简单的跨云协同，让业务管理更高效

ACK One Fleet 为不同地域的多个 K8s 集群提供了统一的控制平面，我们可以对公共云集群、边缘云集群和本地数据中心集群，实现统一的集群管理，资源调度、应用交付以及备份恢复能力。

智联招聘使用 ACK One 实现混合云负载感知弹性，使用 ECI 5 分钟实现业务数万核扩容。

极氪汽车使用 ACK One 统一管理数十个混合云 K8s 集群，提升安全水位和业务连续性，减少 25% 的资源用量，运维效率提高 80%。

在模拟仿真、科学计算等大规模数据计算工作流场景中，一个批次的计算可能需要数万，甚至数十万核算力，超出单地域的弹性供给能力，需要依赖跨地域的计算供给。在 IoT 以及医疗等场景中，海量数据分散在不同地域，需要具备就近计算能力。为此，ACK 推出全托管 Argo 工作流集群，具备事件驱动，大规模、免运维、低成本、跨地域等特点。

- Argo 工作流集群充分利用多 AZ、多地域的弹性算力，自动化利用 ECI Spot，有效降低资源成本。相比自建 Argo 工作流系统，可实现 30% 的资源成本节省。
- 集群内建分布式数据缓存，提供更大的聚合读取带宽，数据吞吐相比直接访问提高 15 倍。
- 集群提供优化 Argo 引擎，并行计算规模提升 10 倍。

泛生子使用全托管 Argo 工作流集群在 12 小时内完成处理数千例肿瘤基因样本的处理，速度提升 50%，成本下降 30%。

6. 阿里云容器服务 ACK，智算时代云原生基础平台

正如一个文明社会的科技水平取决于其对能源的利用能力，企业的智能化水平取决于其对算力的利用能力。云计算为智算时代带来无限可能，阿里云容器服务以为企业构筑现代化应用平台，最大化利用阿里云强大弹性算力为使命：

- 通过对多样化算力的场景化高效利用，提升计算效能
- 通过弹性与调度，提升资源利用率；
- 通过智能自治，降低运维成本
- 通过最佳实践与技术创新，提供端到端安全、可信运行环境

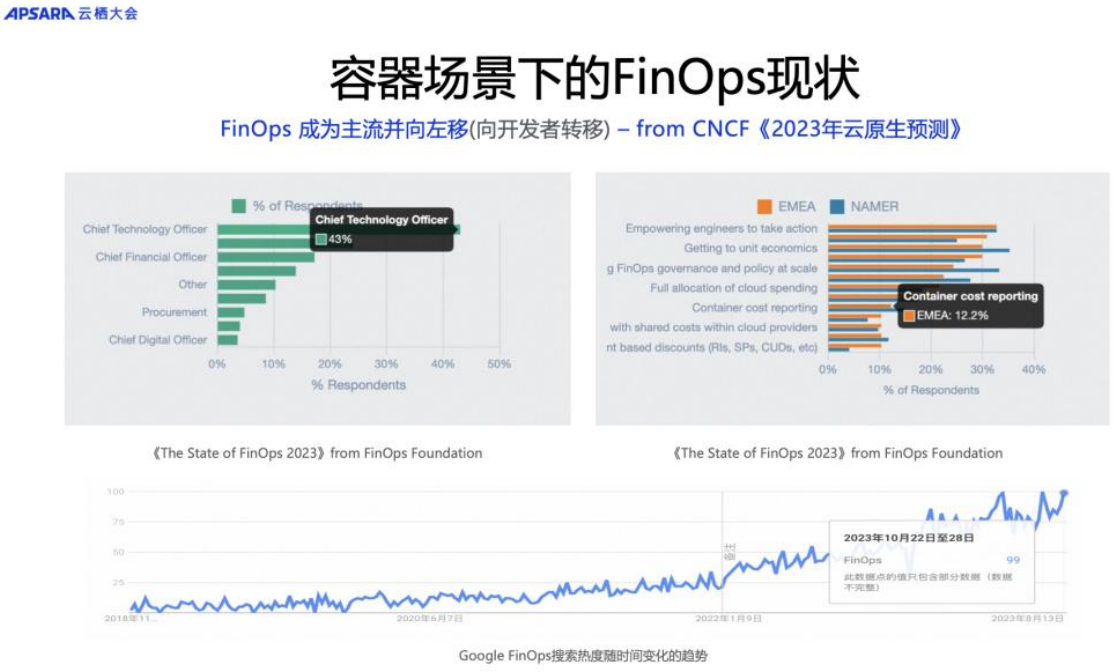
第二章

容器服务典型企业案例

云原生场景下月省 10 万元资源成本，这家企业做对了什么

作者：冯诗淳，阿里云技术专家&ACK 成本套件研发负责人

相信近期从事基础设施工作的各位，对 IT 成本治理，以及 FinOps 体系的概念已经有了一些认知。在 Google 近 5 年的热度趋势中，FinOps 的趋势也在持续上升。在阿里云的同学与客户实际工作协同中，我们发现成本治理是几乎每位客户都存在的普适需求，特别是各位技术管理者重要的关注点之一。据 FinOps 基金会 2023 年的报告，有 43%、24%、17% 的公司，是由 CTO、CIO、CFO 直接指派 FinOps 团队向他汇报，只有 14% 的公司处于还未建立体系化的降本增效的 KPI。



根据 FinOps 基金会的报告，建设 FinOps 体系 Top 的痛点非常复杂，包括技术方面问题、如何驱动工程师进行优化、如何减少浪费的资源、如何在容器场景做成本报告分析；同时也存在管理等问题，比如如何让团队组织适应 FinOps 体系等等。我们希望阿里云在提供产品功能的同时，也能正确真正地帮助我们的客户落地自己的 FinOps 体系，真正让客户降本增效。

在 2023 年云栖大会现场，我们有幸邀请到某头部科技型量化投资公司的云基础设施负责人，为我们提供基于阿里云容器服务成本套件 ACK FinOps 落地的云原生场景成本治理案例，帮助大家了解在容器场景下的企业成本治理现状、挑战，以及如何结合 ACK 成本套件产品功能构建云原生用户自己的 FinOps 体系。

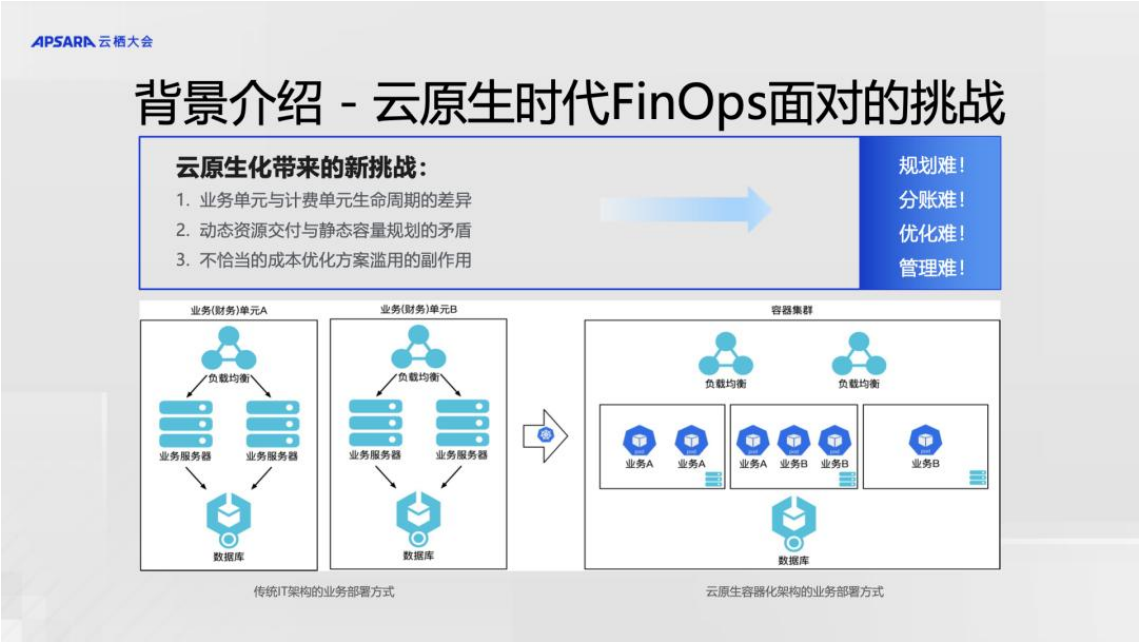
1. 容器场景成本治理挑战与实践

本次分享的企业是中国领先的以人工智能和机器学习为基础的科技型量化投资公司，使用了大量的 AI、大数据作业来辅助量化交易决策，需要大量弹性的算力的同时，也需要更好的实现成本的控制，通过 Kubernetes 将 AI、大数据、工作流等作业放在一个集群中分时、弹性运行。

以该企业为例，业务系统大致分为几类应用部署形态：

- 稳定的系统应用
- 不特定时间的按需任务
- 测试开发环境的应用

这几类应用都会消耗基础计算资源，并产生成本。目前该企业部分业务在使用阿里云容器服务 ACK 集群做容器化部署，通过 Kubernetes 进行量化交易的数据执行与决策，及阿里云 ACK FinOps 套件实现成本的洞察与分摊，经过治理后实现了近 30% 资源水位的提升。在企业成本治理的实践过程中，该企业主要遇到规划难、分账难、管理难、优化难这 4 方面的挑战。



- 规划难

在进行成本治理方面工作时，首先遇到的挑战是按需任务、测试开发环境的容量规划问题。开发、测试应用在容器化部署架构下，实现快速迭代的同时，难以较准确地给出分配的资源量。过度分配资源会导致资源浪费，资源超售过度则会导致稳定性问题。

- 分账难

该企业的云基础设施每天为很多的上层应用提供服务，多个容器应用共享一个 K8s 集群。一个计算节点上可能运行多个 Pod，而且 Pod 可以弹性伸缩，在节点间动态迁移。多个业务应用混部在同一个池化的 K8s 集群中，难以把整个集群的账单分摊到应用和人。应用层与资源层计量计费在空间、时间等多个维度都无法做到一一对应，成本治理的复杂性业因此而来。

- 管理难

另外，由于各个应用的使用场景存在很大差别，每当找出闲置浪费的资源后，往往难以“爽快地”马上缩容下线资源，如何在优化资源成本浪费的同时保障业务的稳定性，一直是一个难以回答的问题。

- 优化难

容器化后是拥有各种丰富的成本优化手段，但“这样调低 request 资源分配水位后，是否影响业务？”，“现有的 HPA 弹性伸缩策略，是否能在业务真正需求资源时正确工作”，甚至于“我现在要下线的网络、存储资源是不是真的没人使用？”

云原生技术中例如弹性、混部、Serverless、超卖等技术都有各自适合的典型场景。如果使用不当，比如弹性配置错误，可能带来意想不到的资源浪费甚至稳定性问题。

如何解决分账难题

首先要面对分账难问题，理清花费在哪儿是最重要的工作。站在 Infra 团队的视角，一直以来和上层业务、应用层的部门同事的协作工作方式都是：

当新业务需要上线、或老业务需要扩容时，业务部门会申请告诉我们他们“期望”使用多少的容量，为了保证业务稳定性，资源需求往往拍脑袋定义，且业务团队都希望申请冗余远远超过实际预期的资源量。

长此以往，集群的水位就会出现大量闲置。由于业务是容器化混合部署的应用在同一集群中，应用的水位分布也往往呈现长尾效应，稳定的大规模应用往往经过重点优化已经有较高的资源利用率，但大量小规模应用使用大量闲置资源。传统部署模型下的资源成本统计方式，是按业务使用的节点维度分析成本，但是在 K8s 场景下，业务使用的资源统一从资源池中调度，业务对资源浪费也隐藏在整个集群、节点的水位中难以发现。要算清这本糊涂账，一定要把成本归因到具体某个业务应用，甚至是具体到某个人，才能推动真正地降本。怎么把成本归因到具体业务，首先需要精细化的监控数据，来看清业务对资源的使用情况。阿里云 ACK 团队可以为企业提供详细的成本、资源观测数据，包括：

- 每天每笔云上资源的真实花销成本账单
- 每个容器部署的资源使用量、使用水位

部门、业务、个人这些业务层层级关系，该企业通过按集群的 namespace、不同工作负载、任务通过打特定 label 的方式，最终与具体 K8s 集群中的花费资源成本的 Pod 进行映射。最终通过结合阿里云 ACK 成本洞察数据的方式，可构建多个不同视角的成本资源监控大盘，包括：

- 每天每笔不同云资源账单维度的监控大盘
- 归因到业务应用/个人的监控大盘

由此，便于分析发现应用维度的浪费，如形成 Top 浪费的应用报表，进行数据驱动地成本优化推进。

面对成本管理难题

Infra 团队在推动降本增效时往往是无力的，更多需要推动跨团队的协作。站在一个业务应用的上线过程来看协同关系，Infra 团队往往职责是接受上层业务层同事的需求，以及保证提供资源，这里的需求关系是从业务层到 Infra 层是至顶向下的。

然而 Infra 团队与成本资源花销的距离是最近的，感知是最深切的，所以往往需要由 Infra 团队来推动成本治理，构建 FinOps 体系的建设。

这里的路径在跨部门的协同关系上反而是至下而上反方向的。Infra 团队就算找到对应的业务团队，推动他们缩容、下架掉闲置的云资源，往往由于没有数据驱动或对降本增效清晰的认识而难以开展工作，最终会导致极其低效的降本增效，白白浪费 Infra 团队工程师们宝贵的时间。

我们不妨换个思路拆解一下解决方案。首先需要明确，所有人都需要对降本增效负责，且需要划分清晰的责任范围。以该企业为例，业务协同主要分为三大类角色：

- 业务应用团队：负责业务应用的具体研发
- 业务平台团队：负责为业务应用提供通用业务能力
- Infra 团队：为以上团队提供基础设施

顺着成本治理的至下而上的路径，该企业划分了成本治理清晰的权责范围，以及通过构建不同视角的成本监控大盘构建统一的数据驱动成本洞察体系。

首先对成本资源感知距离最近的 Infra 团队：

拿数据说话，驱动业务团队优化。

通过集群的 overview 整体视角的监控大盘，从集群、各项云资源、节点等视角，界定确定性的浪费资源，以及通过对各集群资源使用的 breakdown 分析，找到成本问题的症结所在。

对于业务平台团队，从业务预算、Quota 层面驱动业务成本优化。每个业务也需要从财务层面做成本治理，这里业务平台团队通过成本洞察的数据，结合财务的预算，形成统一的报表、监控。如预算超标，需要透传分配 Infra 团队根据 breakdown 数据，进行成本分析。

业务应用团队，需要选择科学可靠的成本优化手段。作为应用的研发，使用业务平台、Infra 平台，他们是对业务、代码最了解的专家，也是需要平衡资源浪费与应用稳定性的最终负责人。

在 FinOps 体系中，ACK 成本套件为他们提供应用视角的监控大盘，清晰观测自己应用资源、成本水位的同时，判断收敛后的资源水位是否合理，以及对自身业务变化规律来制定科学的弹性策略以满足动态资源的需求。

如何规划资源 & 成本

有了以上的分账、跨团队协作的解决方案后，我们来看规划难的问题。新业务上线需要规范流程，制定合理的容量规划。而新业务、跑批任务等，经过上线前压测，通过经验值或成本套件资源画像等只能推荐出科学的资源规格配置。

针对这个问题，在上线过程可以使用 ACK AHPA 等智能弹性策略来做到动态业务趋势的智能资源调整。

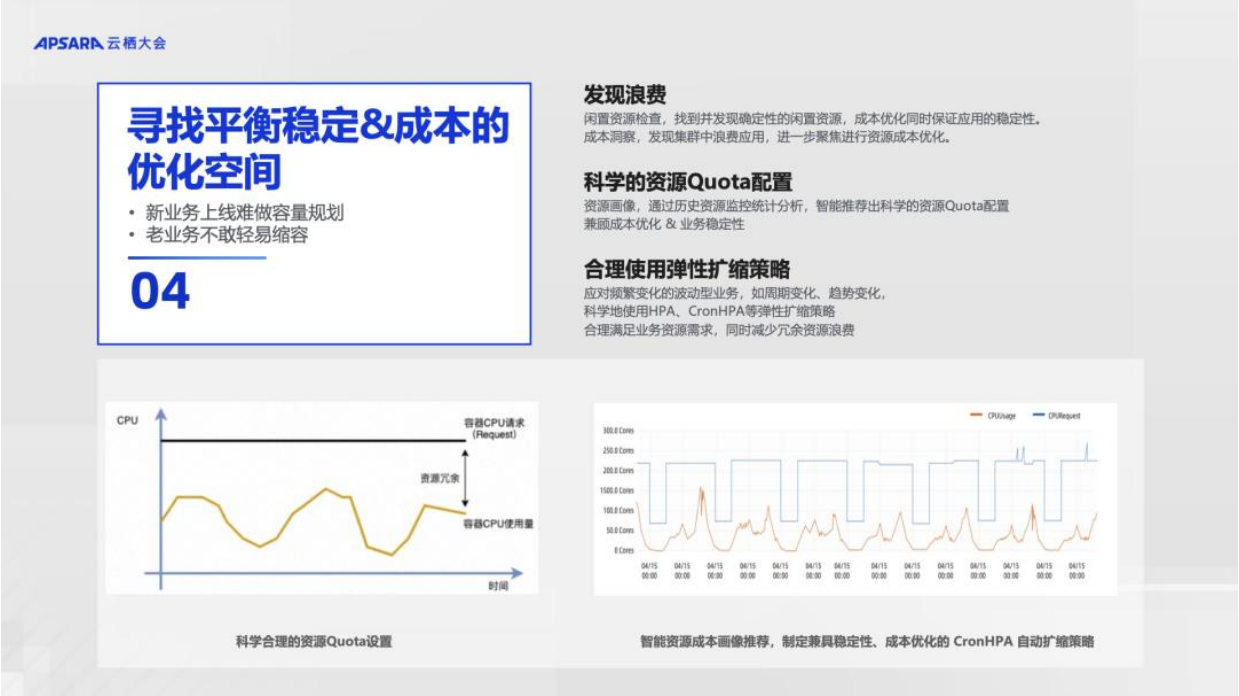
每个业务都不应该无限申请成本。把成本、资源归因到个人，同时也需要根据业务量、资源趋势制定财务预算，以及成本 Quota 计划。合理地进行成本控制。

部门、业务、个人的成本预算，应按应用使用比例分摊到集群中的应用部署、Pod。该企业的做法主要是通过 namespace、给容器副本打业务 label 的方式进行映射。最终预算与归因到对应业务后的实际成本花费进行比对。

成本控制方面也是通过 API 集成 ACK 成本洞察的成本数据后，细粒度到业务应用、个人

来配置的成本超预算报警。

寻找稳定和成本间的平衡



最后，在真正进行资源优化过程中。平衡稳定性和成本浪费是非常重要的。

首先对于浪费发现，存在两部分浪费：

- 首先需要发现确定性的浪费。完全没有使用的网络 SLB、EIP 等资源，长期空闲的节点等，这些可以通过 ACK 限制资源检查找到这些确定性的浪费。

- 第二部分是非常普遍的，应用的资源浪费。虽然平均集群资源利用率经过优化达到了约 50%。由于是容器化混合部署的应用在同一集群中，应用的水位分布也往往呈现长尾效应，稳定的大规模应用往往经过重点优化已经有较高的资源利用率，但大量小规模应用使用大量闲置资源，隐藏在整个集群中难以发现。这里可以通过拉取 ACK 成本洞察的 Pod 维度的成本资源数据，归因浪费到具体应用/个人后，会使这些应用的碎片化浪费逐个暴露出来。

科学合理的 Quota 设置

对 K8s 有经验的使用者，对 K8s 资源分配量(Request)、资源限制(Limit)两个值应该会有深刻的理解。科学地配置工作负载的 Request 量可以帮助进行容量规划控制资源成本，Limit 资源限制则可以实现混部的超卖和保证应用的稳定性。

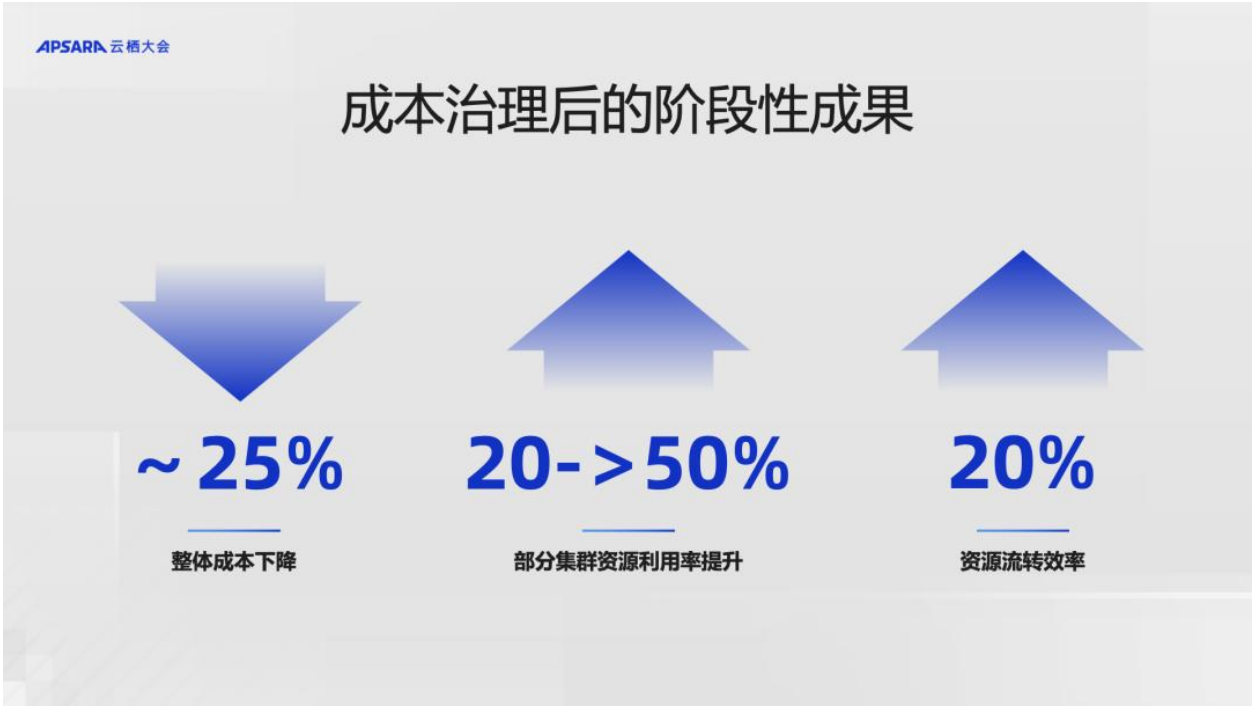
通过统一 K8s 集群上应用的 request、limit 设置规范，通过业务量压测、预估经验值，结合根据历史资源使用量的 ACK 资源画像智能推荐的 request、limit 值，该企业可以做到科学地为各个应用设置合理 Quota，平衡业务稳定性和成本浪费。

合理地使用弹性策略

HPA 很先进，但激进的 HPA 配置会导致应用不符合预期地扩缩、甚至导致业务稳定性；保守的 HPA 配置可能会导致还是会有大量闲置资源，起不到太多成本节省的效果。

云原生技术中例如通过业务指标进行 HPA、CronHPA 等都有各自适合的典型场景。在该企业中也有部分业务应用使用 HPA 策略。首先比较确定性的场景如周期性的业务，使用 CronHPA；同时，参考成本、资源监控数据优化阈值，通过 HPA 的历史数据，保证资源的流转效率。

在决定 HPA 的指标的选择上，该企业会先区分 CPU 密集型的业务还是内存密集型的业务，根据调度的关键资源指标作为 HPA 的决定值。在一些新的业务，没有能参考的资源指标场景，也在使用 ACK AHPA 智能 HPA 策略，形成动态智能的弹性扩缩。

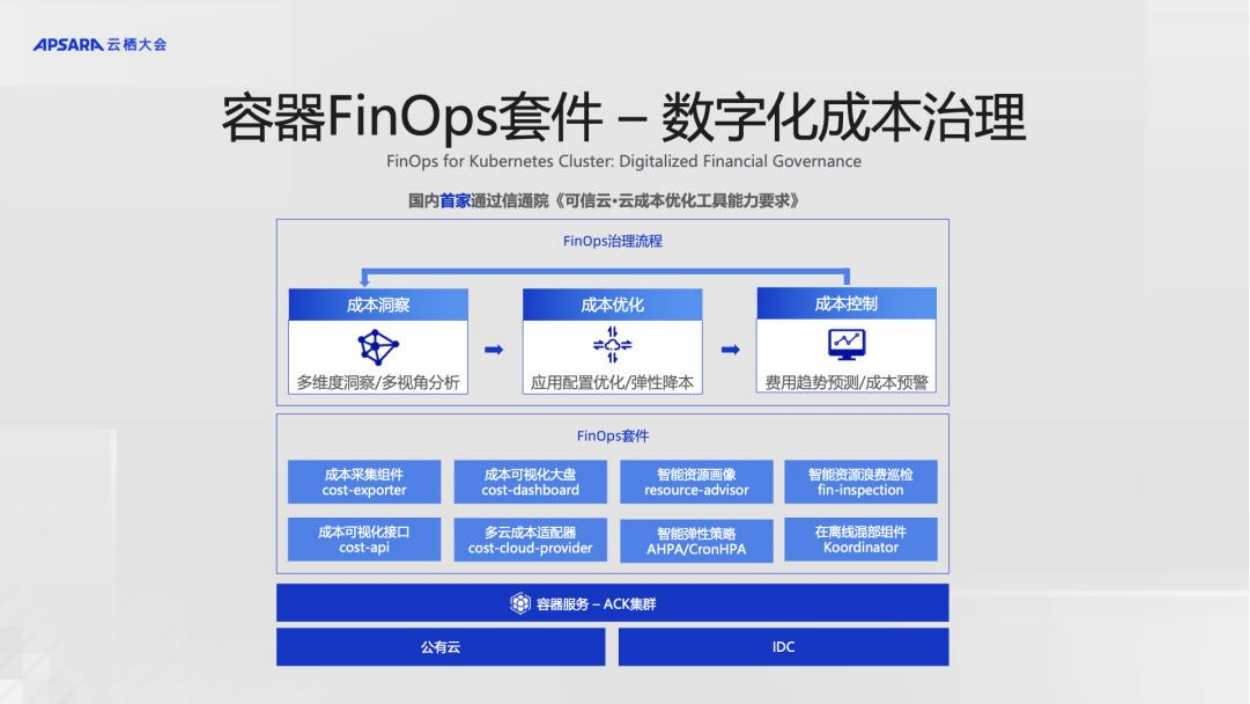


整个成本治理工作是一个复杂且综合性的事务。经过近一年多，目前在 IT 成本上节省约 25% 的成本，超过月 10w+ 的成本节省，部分集群资源利用率从 20% 提升至 50%。

在整个实践的过程中，该企业也定义了资源流转效率指标，一个业务应用通过弹性扩缩对新资源的使用率，来反映一个应用对资源的浪费程度，资源流转效率越大代表越节约。目前经过 IT 成本治理，资源流转效率有了 20% 的提升。“我们也希望通过本次分享我们在 IT 成本治理方面的工作经验，帮助其他互联网金融客户等云上客户更好地建设 FinOps 体系。”

2. 阿里云 ACK FinOps 套件助力容器成本数字化治理

阿里云 ACK 团队希望提供真正能帮助用户在容器场景构建 FinOps 体系的产品能力。在深入沟通、了解企业对于容器成本治理的需求和问题后，我们总结出通用的三大 FinOps 治理流程：成本洞察、成本优化、以及成本控制。



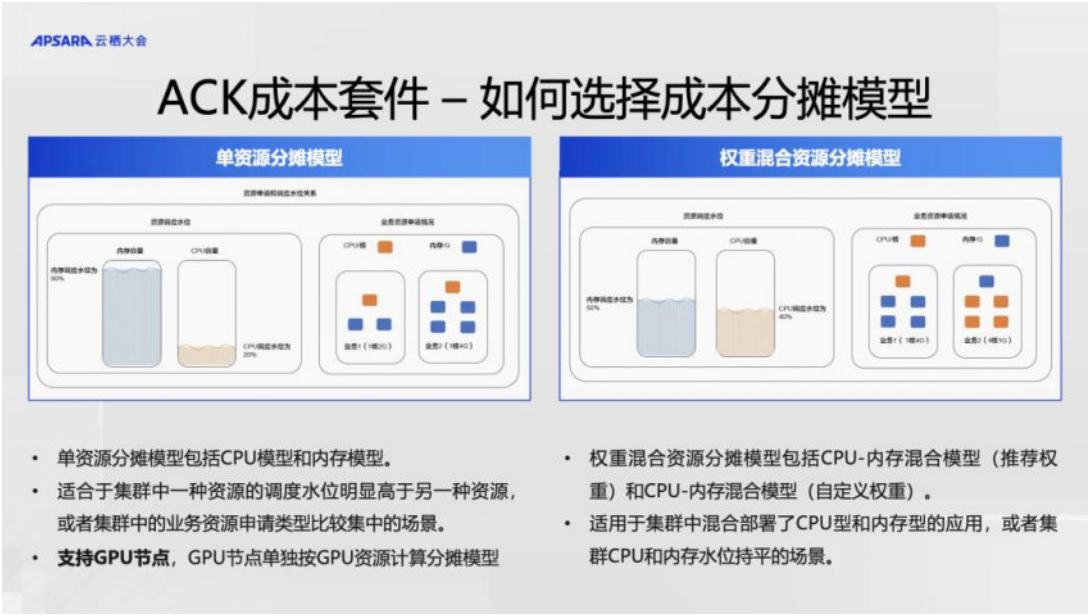
在成本洞察中，ACK FinOp 套件提供多维度视角，帮助用户把集群中业务成本归因到组织和个人。成本洞察能力经过更多客户场景的打磨，推出更科学的分账算法，同时目前支持通过 API 让客户进行二次开发，以及如极氪汽车等多云场景我们支持多云成本适配器，帮助多云、IDC 机器混合等场景下成本治理保持统一。

在成本优化中，我们提供资源画像功能，智能推荐应用优化配置，并通过 Koordinator 在离线混部组件进一步提升资源利用率。以及提供 CronHPA、 AHPA 等丰富的自动弹性扩缩容策略。并提供智能资源浪费巡检。

在成本控制阶段，ACK 将提供成本洞察大盘周报功能，直接抄送成本周报至对应业务团队更能推进团队进行成本优化，并树立 FinOps 建设意识，提供费用趋势预测，帮助刚好地指定业务预算，最终进行成本控制。

看清成本、找出浪费，永远是成本治理的第一步。ACK 成本洞察功能帮助用户构建数据驱动的成本观测能力。

ACK 成本洞察功能提供开箱即用的集群成本大盘，实时计算出多维视角的集群应用成本账单，以及提供不同资源配型，如包年包月、抢占式节点的横向比较，以及推荐不同的节省策略。同时，下钻到应用层的应用视角成本大盘，提供对应用浪费资源程度的 Top 排序，清晰明了发现混部隐藏在集群中的应用浪费问题。



Infra 团队和上层应用团队需要按统一的口径进行成本分账。

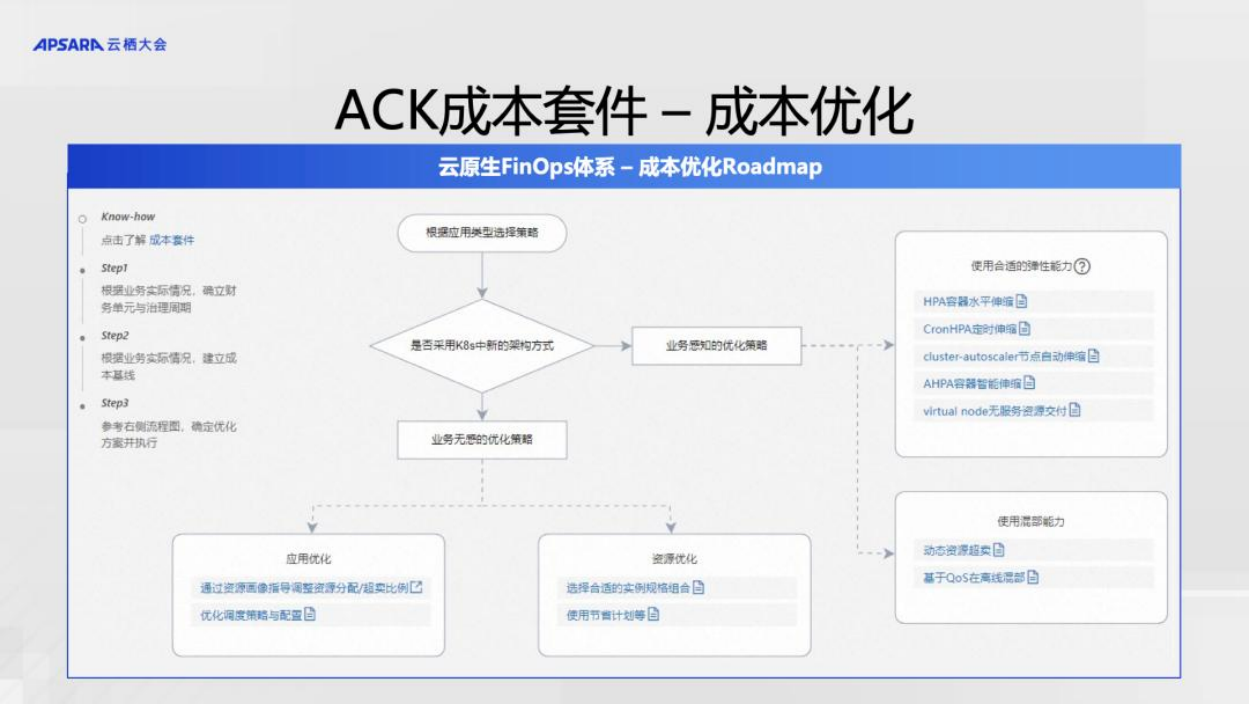
云原生场景架构复杂，不同应用形态也会在混部场景中对调度资源产生影响。ACK FinOps 提供独有的云原生容器场景成本分摊与估算模型，通过衡量应用对调度的影响大小，更科学合理地对应用成本进行拆分。

多数用户的应用可分为两种场景，如 JAVA、J2EE 部署的应用，多为内存密集型场景，如跑批的分布式计算任务，多为 CPU 密集型场景，此类场景，CPU 或内存、甚至 GPU，会作为集群调度的关键资源，决定应用是否能被调度。此场景我们推荐单资源分摊模型，按关键资源进行分账。

如一个典型场景，用户在 ACK 集群的 GPU 节点跑 spark 的跑批任务，GPU 资源是当前最影响调度的资源，所以该应用的成本，应该按当前应用运行时间内占用的 GPU 资源来拆分整个节点的成本。

混部场景的用户，作为云原生的深度用户，一个集群中会有内存密集型、CPU 密集型等多种应用混部，此时每种资源都决定调度策略，此时我们推出按资源调度水位计算的权重混合资源分摊模型，此模型计算一个应用应该分摊的成本，是由他所申请资源影响可调度资源的部分决定。

ACK 成本洞察可通过混合资源分摊模型，按每个应用对调度的影响，自动计算出合理的应用分账成本。



看清了成本浪费后，云原生容器场景下有复杂的架构体系，如何进行优化往往无从下手。ACK FinOPs 成本套件梳理出 ACK 成本优化路径落地的最佳实践，帮助用户在同步场景选型不同的成本优化方案。如业务应用经常波动的场景，可以通过感知业务的波动，选择自动弹性策略、或通过混部场景的动态资源超卖等提高资源利用率。在不感知业务的情况下，可以检查是否已经是最优的资源配型等。

确定性的浪费是我们首要需要找出来并收敛掉的。真实生产环境中，Infra 团队不敢轻易删除集群里的资源，在此 ACK 成本套件推出闲置资源巡检功能，帮助用户找出集群里确定性的闲置资源。这里通过找到处于未使用状态的资源，但在出账时却被计入本集群的成本的资源。包括 无业务应用使用的 云服务器 ECS、块存储、负载均衡 CLB 和弹性公网 IP 的闲置检查。

根据 FinOps 基金会的 2023 年报告，对云资源的更大利用率使用，以及如何驱动工程师团队采取优化措施是现在 FinOps 体系中最令人头疼的问题。

在容器场景混部、动态的应用环境下，ACK 资源画像功能可以提供基于应用历史数据的智能资源配置推荐功能。

千人千面，在容器场景下是一应用一画像。为每个应用智能推荐升降配策略。解决应用刚上线、或应用业务波动大，无法正确容量规划问题。资源画像的核心技术点在于提供同时平衡过冗余时的浪费且保证过度超卖的稳定性的推荐算法。我们的推荐算法主要考虑了以下 3 方面：

- 使用多种资源维度进行统计，并使用类似分位数的统计方法区分应用突发峰值需求和日常资源需求。
- 使用半衰期华东窗口模型，确保新的数据对算法模型的影响越大，越旧的数据对算法的影响越小。
- 以及考虑了容器运行时状态，参考容器的 OOM 等运行状态，进一步提高推荐值的准确性以及保证稳定性。

用户为什么要云原生容器化，除了使用统一标准化的配置方式规范地使用云资源，更大程度也是为了享受集群池化的资源带来的资源利用率提升与系统稳定性的平衡。

HPA 弹性伸缩策略是 Kubernetes 技术生态对这一平衡的重要体现。ACK 容器服务提供丰富的 HPA 弹性策略，针对不同的场景。

- 标准的 HPA，主要解决的业务负载压力波动比较大时，需要人工根据监控来不断调整副本数的问题，实现自动扩缩容。

- VPA 垂直扩容 Pod 所在的节点，主要解决资源配额（Pod 的 CPU、内存的 limit/request）评估不准的问题。

- 非常实用的如上文企业面对的周期性波动的应用场景，使用 CronHPA，定时水平扩缩容。基于可预期的时间，提前规划扩缩容 Pod 数。主要解决在可预期的时间，资源不足的问题。

同时我们也提供一些领域垂直的弹性伸缩解决方案，如业务事件驱动的 Keda、以及 Serverless 场景支持如灰度发布等场景的 knative 等领域弹性伸缩解决方案。

HPA 的配置确实需要根据应用具体场景，如是否波动，来决定具体选择哪种 HPA 解决方案，以及关键指标应该如何选择等。很多同学看到这里就知难而退了，这里有太多的 HPA 策略，复杂的场景，难以规划的阈值参数，需要丰富的 K8s 经验。现在 ACK 推出 AHPA 智能弹性策略功能，解决这个问题，也让我们窥见了下一个阶段 HPA 弹性策略的新形态。

AHPA 通过收集应用 Pod 的历史数据，通过智能的周期检查+预测算法，结合 ACK 专业的 K8s 应用部署经验。

- 通过资源提前预热，解决 HPA 弹出滞后性的问题。

- AHPA 自动配置阈值，智能识别业务指标曲线，无需人工干预，自动弹性规划。
- 支持配置弹性降级保护，快速兜底容错。

AHPA 使用资源提前预热的方法，根据智能算法提前预测将要发生的弹性对资源的需求，实时调整资源容量。正常客户的 HPA 拉起新的 Pod，需要经历如资源调度、拉取镜像、等待容器启动等耗时过程，AHPA 预热后解决客户弹性之后性的问题。

通过历史数据的智能预测，无需人工干预，自动规划弹性策略、阈值，解决阈值配错、不好配的问题；以及 AHPA 与标准 HPA 相比，更合理阈值的配置也解决了弹性的滞后性问题。

对突发的业务流量收缩，支持配置弹性降级保护措施，避免过保守的 HPA 策略不能降本增效，过激进的 HPA 策略面对突发情况时会影响业务稳定性。在提升资源使用率的同时，保障业务的稳定性。

希望通过我们的分享，能够帮助更多企业了解在容器场景下的企业成本治理现状、挑战，以及如何结合阿里云容器服务 ACK 成本套件产品功能，构建企业自己的云原生 FinOps 体系。

米哈游大数据云原生实践

作者：米哈游大数据开发

近年来，容器、微服务、Kubernetes 等各项云原生技术的日渐成熟，越来越多的公司开始选择拥抱云原生，并开始将 AI、大数据等类型的企业应用部署运行在云原生之上。以 Spark 为例，在云上运行 Spark 可以充分享有公共云的弹性资源、运维管控和存储服务等，并且业界也涌现了不少 Spark on Kubernetes 的优秀实践。

在刚刚结束的 2023 云栖大会上，米哈游数据平台组大数据技术专家杜安明分享了米哈游大数据架构向云原生化升级过程中的目标、探索和实践，以及如何通过以阿里云容器服务 ACK 为底座的 Spark on K8s 架构，获得在弹性计算、成本节约以及存算分离方面的价值。

1. 背景简介

随着米哈游业务的高速发展，大数据离线数据存储量和计算任务量增长迅速，早期的大数据离线架构已不再满足新场景和需求。

为了解决原有架构缺乏弹性、运维复杂、资源利用率低等问题，2022 年下半年，我们着手调研将大数据基础架构云原生化，并最终在阿里云上落地了 Spark on K8s + OSS-HDFS 方案，目前在生产环境上已稳定运行了一年左右的时间，并获得了弹性计算、成本节约以及存算分离这三大收益。

1） 弹性计算

由于游戏业务会进行周期版本更新、开启活动以及新游戏的上线等，对离线计算资源的需求与消耗波动巨大，可能是平时水位的几十上百倍。利用 K8s 集群天然的弹性能力，将 Spark 计算任务调度到 K8s 上运行，可以比较轻松的解决这类场景下资源消耗洪峰问题。

2） 成本节约

依托阿里云容器服务 Kubernetes 版 ACK 集群自身强大的弹性能力，所有计算资源按量申请、用完释放，再加上我们对 Spark 组件的定制改造，以及充分利用 ECI Spot 实例，在承载同等计算任务和资源消耗下，成本节约达 50%。

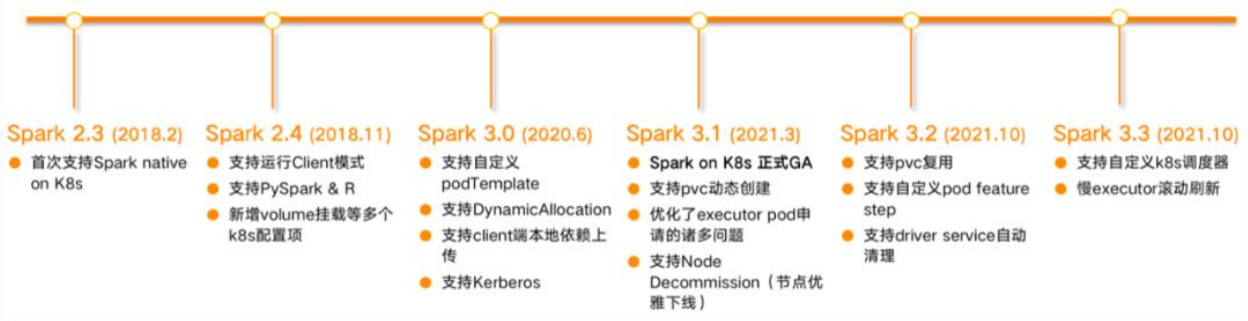
3） 存算分离

Spark 运行在 K8s 之上，完全使用 K8s 集群的计算资源，而访问的则数据也由 HDFS、OSS 逐步切换到 OSS-HDFS 上，中间 Shuffle 数据的读写采用 Celeborn，整套架构实现了计算和存储的解耦，易于维护和扩展。

2. Spark on K8s 架构演进

众所周知，Spark 引擎可以支持并运行在多种资源管理器之上，比如 Yarn、K8s、Mesos 等。在大数据场景下，目前国内大多公司的 Spark 任务还是运行在 Yarn 集群之上的，Spark 在 2.3 版本首次支持 K8s,并于 2021 年 3 月发布的 Spark3.1 版本才正式 GA。

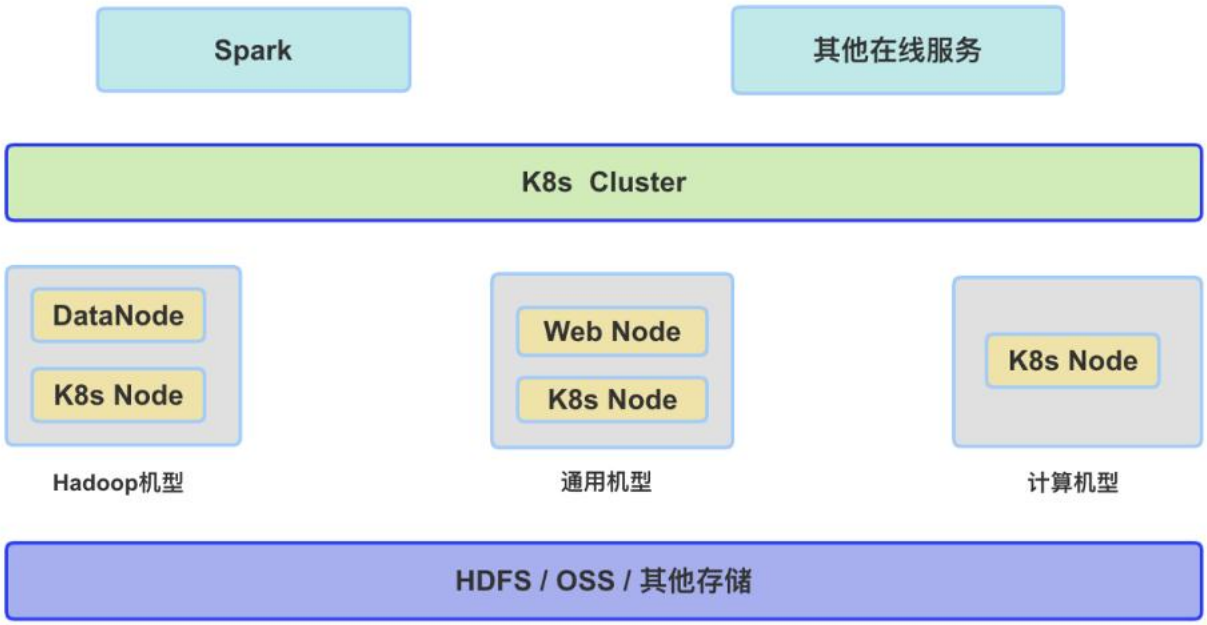
相较于 Yarn，Spark 在 K8s 上起步较晚，尽管在成熟度、稳定性等方面还存在一定的欠缺，但是 Spark on K8s 能够实现弹性计算以及成本节约等非常突出的收益，所以各大公司也都在不断进行尝试和探索，在此过程中，Spark on K8s 的运行架构也在不断的向前迭代演进。



1） 在离线混部

目前，将 Spark 任务运行在 K8s 上，大多公司采用的方案依旧是在线与离线混合部署的方式。架构设计依据的原理是，不同的业务系统会有不同的业务高峰时间。大数据离线业

务系统典型任务高峰期间会是凌晨的 0 点到 9 点钟，而像是各种应用微服务、Web 提供的 BI 系统等，常见的业务高峰期是白天时间，在这个时间以外的其它时间中，可以将业务系统的机器 Node 加入到 Spark 所使用的 K8s NameSpace 中。如下图所示，将 Spark 与其他在线应用服务等都部署在一套 K8s 集群之上。



该架构的优点是可以通过在离线业务的混合部署和错峰运行，来提升机器资源利用率并降低成本，但是缺点也比较明显，即架构实施起来复杂，维护成本比较高，而且难以做到严格的资源隔离，尤其是网络层面的隔离，业务之间不可避免的会产生一定的相互影响，此外，我们认为该方式也不符合云原生的理念和未来发展趋势。

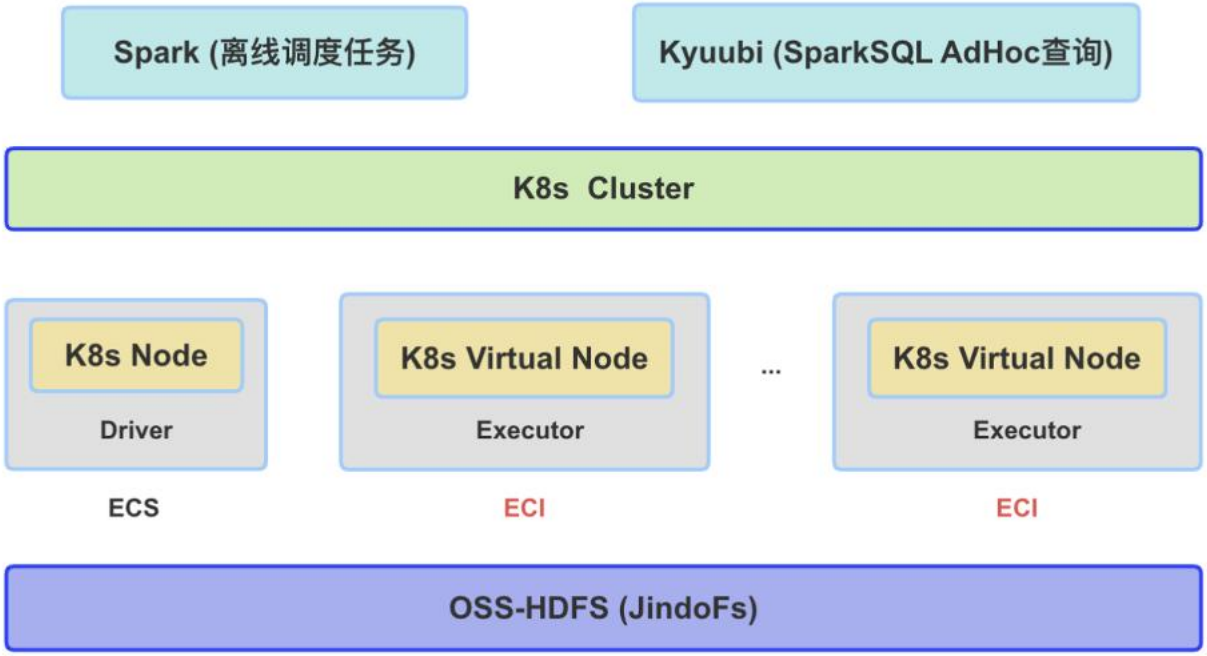
2） Spark on K8s + OSS-HDFS

考虑到在离线混合部署的弊端，我们设计采用了一种新的、也更加符合云原生的实现架构：底层存储采用 OSS-HDFS(JindoFs)，计算集群采用阿里云的容器服务 ACK，Spark 选择功能相对丰富且比较稳定的 3.2.3 版本。

OSS-HDFS 完全兼容了 HDFS 协议，除了具备 OSS 无限容量、支持数据冷热存储等优点以外，还支持了目录原子性、毫秒级 rename 操作，非常适用于离线数仓，可以很好的平替现有 HDFS 和 OSS。

阿里云 ACK 集群提供了高性能、可伸缩的容器应用管理服务，可以支持企业级 Kubernetes 容器化应用的生命周期管理，ECS 是大家所熟知的阿里云服务器，而弹性容器实例 ECI 是一种 Serverless 容器运行服务，可以按量秒级申请与释放。

该架构简单易维护，底层利用 ECI 的弹性能力，Spark 任务可以较为轻松的应对高峰流量，将 Spark 的 Executor 调度在 ECI 节点上运行，可最大程度的实现计算任务弹性与最佳的降本效果，整体架构的示意图如下所示。



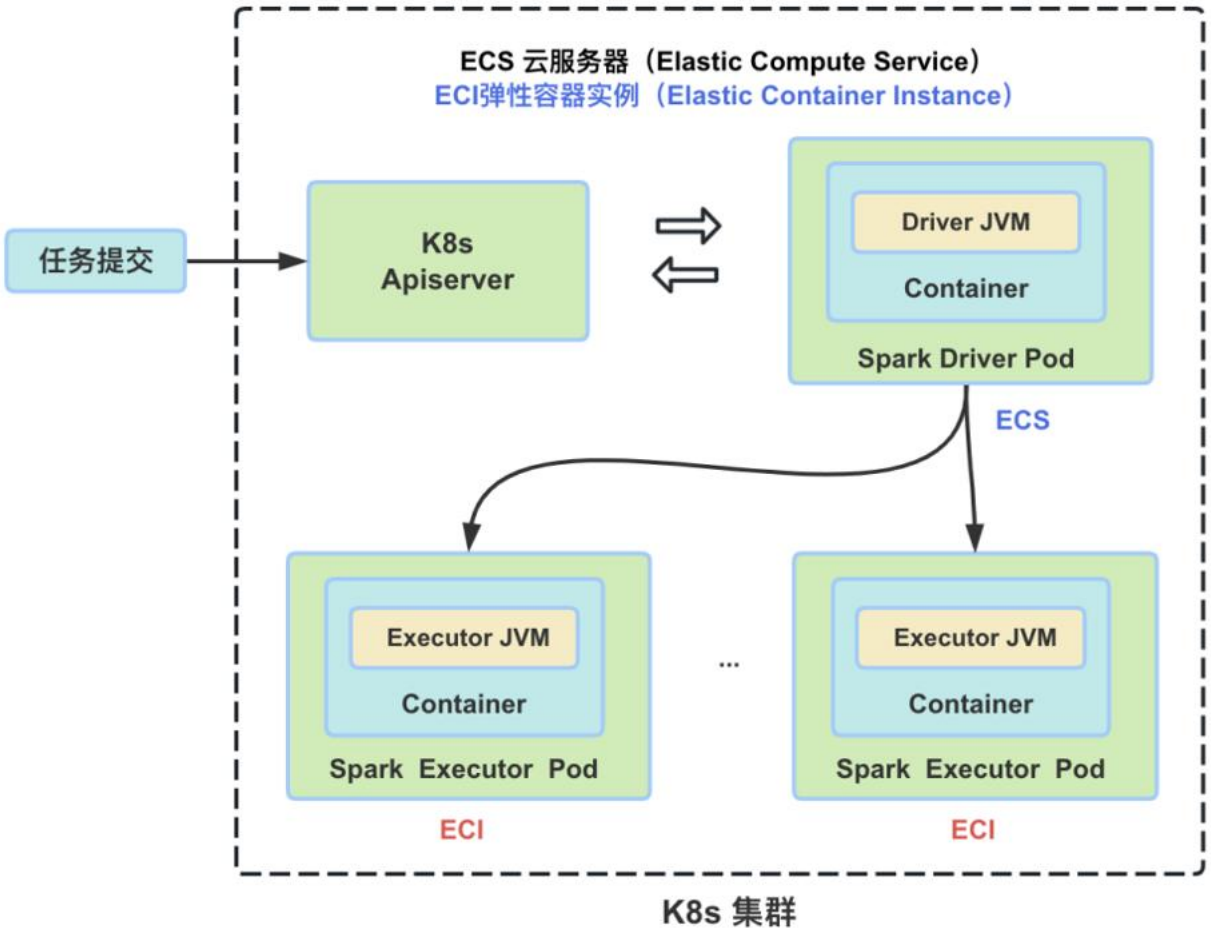
3. 云原生架构设计与实现

1）基本原理

在阐述具体实现之前，先简要介绍一下 Spark 在 K8s 上运行的基本原理。Pod 在 K8s 中是最小的调度单元，Spark 任务的 Driver 和 Executor 都是一个单独 Pod，每个 Pod

都分配了唯一的 IP 地址，Pod 可以包含一个或多个 Container，无论是 Driver 还是 Executor 的 JVM 进程，都是在 Container 中进行启动、运行与销毁的。

一个 Spark 任务被提交到 K8s 集群之后，首先启动的是 Driver Pod，而后 Driver 会向 Apiserver 按需申请 Executor，并由 Executor 去执行具体的 Task，作业完成之后由 Driver 负责清理所有的 Executor Pod，以下是这几者关系的简要示意图。

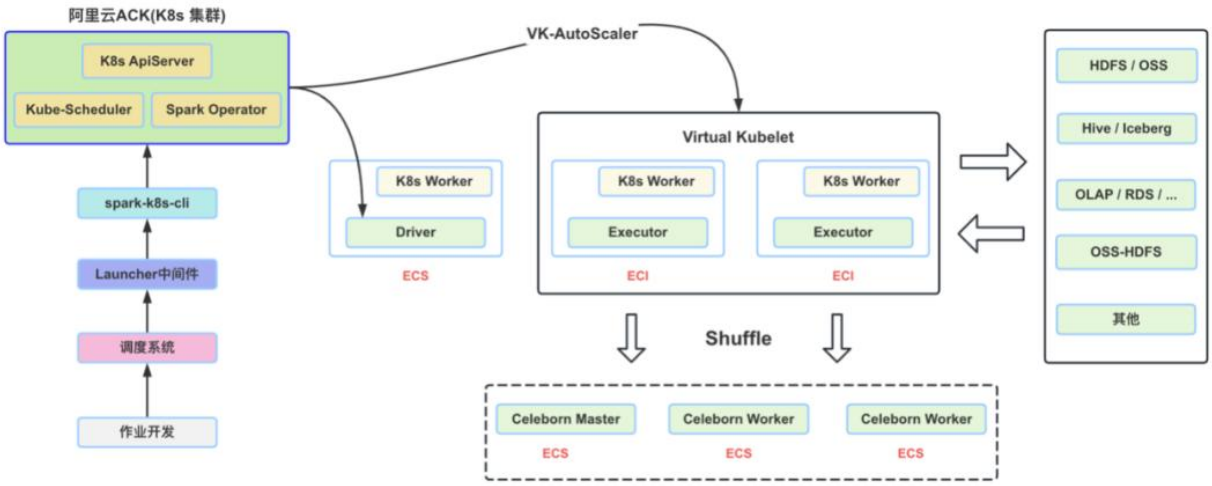


2） 执行流程

下图展示了完整的作业执行流程，用户在完成 Spark 作业开发后，会将任务发布到调度系统上并进行相关运行参数的配置，调度系统定时将任务提交到自研的 Launcher 中间件，并由中间件来调用 spark-k8s-cli，最终由 Cli 将任务提交至 K8s 集群上。

任务提交成功之后，Spark Driver Pod 最先启动，并向集群申请分配 Executor Pod，

Executor 在运行具体的 Task 时，会与外部 Hive、Iceberg、OLAP 数据库、OSS-HDFS 等诸多大数据组件进行数据的访问与交互，而 Spark Executor 之间的数据 Shuffle 则由 CeleBorn 来实现。



3） 任务提交

关于如何将 Spark 任务提交到 K8s 集群上，各个公司的做法不尽相同，下面先简要描述下目前比较常规的做法，然后再介绍目前我们线上所使用的任务提交和管理方式。

3.1 使用原生 spark-submit

通过 spark-submit 命令直接提交，Spark 原生就支持这种方式，集成起来比较简单，也符合用户的习惯，但是不方便进行作业状态跟踪和管理，无法自动配置 Spark UI 的 Service 和 Ingress，任务结束后也无法自动清理资源等，在生产环境中并不适合。

3.2 使用 spark-on-k8s-operator

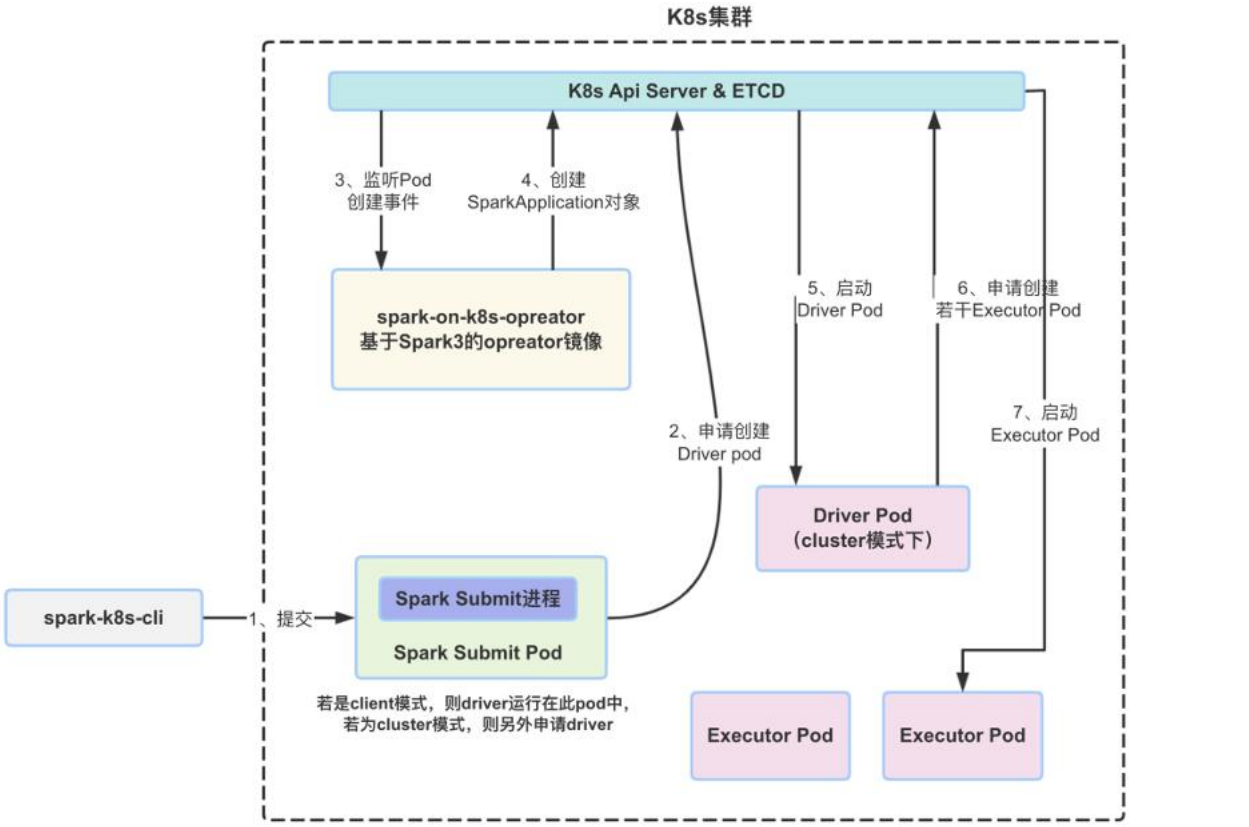
这是目前较常用的一种提交作业方式，K8s 集群需要事先安装 spark-operator，客户端通过 kubectl 提交 yaml 文件来运行 Spark 作业。本质上这是对原生方式的扩展，最终提交作业依然是使用 spark-submit 方式，扩展的功能包括：作业管理，Service/Ingress 创建与清理，任务监控，Pod 增强等。此种方式可在生产环境中使用，但与大数据调度平台集成性不太好，对于不熟悉 K8s 的用户来说，使用起来复杂度和上手门槛相对较高。

3.3 使用 spark-k8s-cli

在生产环境上，我们采用 spark-k8s-cli 的方式进行任务的提交。spark-k8s-cli 本质上是一个可执行的文件，基于阿里云 emr-spark-ack 提交工具我们进行了重构、功能增强和深度的定制。

spark-k8s-cli 融合 spark-submit 和 spark-operator 两种作业提交方式的优点，使得所有作业都能通过 spark-operator 管理，支持运行交互式 spark-shell 和本地依赖的提交，并且在使用方式上与原生 spark-submit 语法完全一致。

在上线使用初期，我们所有任务的 Spark Submit JVM 进程都启动在 Gateway Pod 中，在使用一段时间后，发现该方式稳定性不足，一旦 Gateway Pod 异常，其上的所有正在 Spark 任务都将失败，另外 Spark 任务的日志输出也不好管理。鉴于此种情况，我们将 spark-k8s-cli 改成了每个任务使用单独一个 Submit Pod 的方式，由 Submit Pod 来申请启动任务的 Driver，Submit Pod 和 Driver Pod 一样都运行在固定的 ECS 节点之上，Submit Pod 之间完全独立，任务结束后 Submit Pod 也会自动释放。spark-k8s-cli 的提交和运行原理如下图所示。



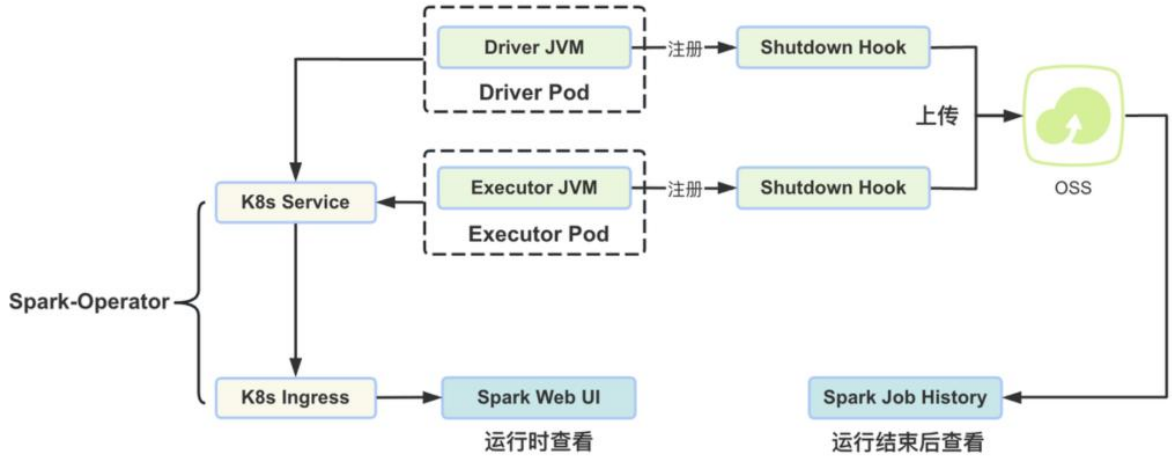
关于 spark-k8s-cli，除了上述基本的任务提交以外，我们还做了其他一些增强和定制化的功能。

- 支持提交任务到同地域多个不同的 K8s 集群上，实现集群之间的负载均衡和故障转移切换
- 实现类似 Yarn 资源不足时的自动排队等待功能 （K8s 如果设置了资源 Quota，当 Quota 达到上限后，任务会直接失败）
- 增加与 K8s 网络通信等异常处理、创建或启动失败重试等，对偶发的集群抖动、网络异常进行容错
- 支持按照不同部门或业务线，对大规模补数任务进行限流和管控功能
- 内嵌任务提交失败、容器创建或启动失败以及运行超时等告警功能

4） 日志采集与展示

K8s 集群本身并没有像 Yarn 那样提供日志自动聚合和展示的功能，Driver 和 Executor 的日志收集需要用户自己来完成。目前比较常见的方案是在各个 K8s Node 上部署 Agent，通过 Agent 把日志采集并落在第三方存储上，比如 ES、SLS 等，但这些方式对于习惯了在 Yarn 页面上点击查看日志的用户和开发者来说，使用起来很不方便，用户不得不跳转到第三方系统上捞取查看日志。

为实现 K8s Spark 任务日志的便捷查看，我们对 Spark 代码进行了改造，使 Driver 和 Executor 日志最终都输出到 OSS 上，用户可以在 Spark UI 和 Spark Jobhistory 上，直接点击查看日志文件。



上图所示为日志的收集和展示原理，Spark 任务在启动时，Driver 和 Executor 都会首先注册一个 Shutdown Hook，当任务结束 JVM 退出时，调用 Hook 方法把完整的日志上传到 OSS 上。

此外，想要完整查看日志，还需要对 Spark 的 Job History 相关代码做下改造，需要在 History 页面显示 stdout 和 stderr，并在点击日志时，从 OSS 上拉取对应 Driver 或 Executor 的日志文件，最终由浏览器渲染查看。

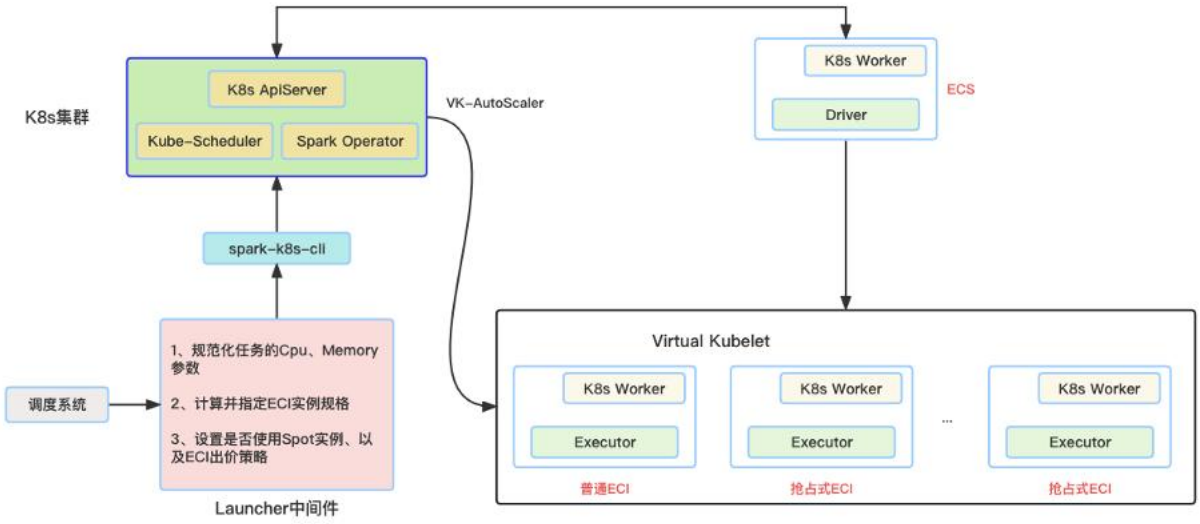
另外，对于正在运行中的任务，我们会提供一个 Spark Running Web UI 给用户，任务提

交成功后，spark-operator 会自动生成的 Service 和 Ingress 供用户查看运行详情，此时日志的获取通过访问 K8s 的 api 拉取对应 Pod 的运行日志即可。

5）弹性与降本

基于 ACK 集群提供的弹性伸缩能力，再加上对 ECI 的充分利用，同等规模量级下的 Spark 任务，运行在 K8s 的总成本要明显低于在 Yarn 固定集群上，同时也大大提高了资源利用率。

弹性容器实例 ECI 是一种 Serverless 容器运行服务，ECI 和 ECS 最大的不同就在于 ECI 是按量秒级计费的，申请与释放速度也是秒级的，所以 ECI 很适合 Spark 这一类负载峰谷明显的计算场景。

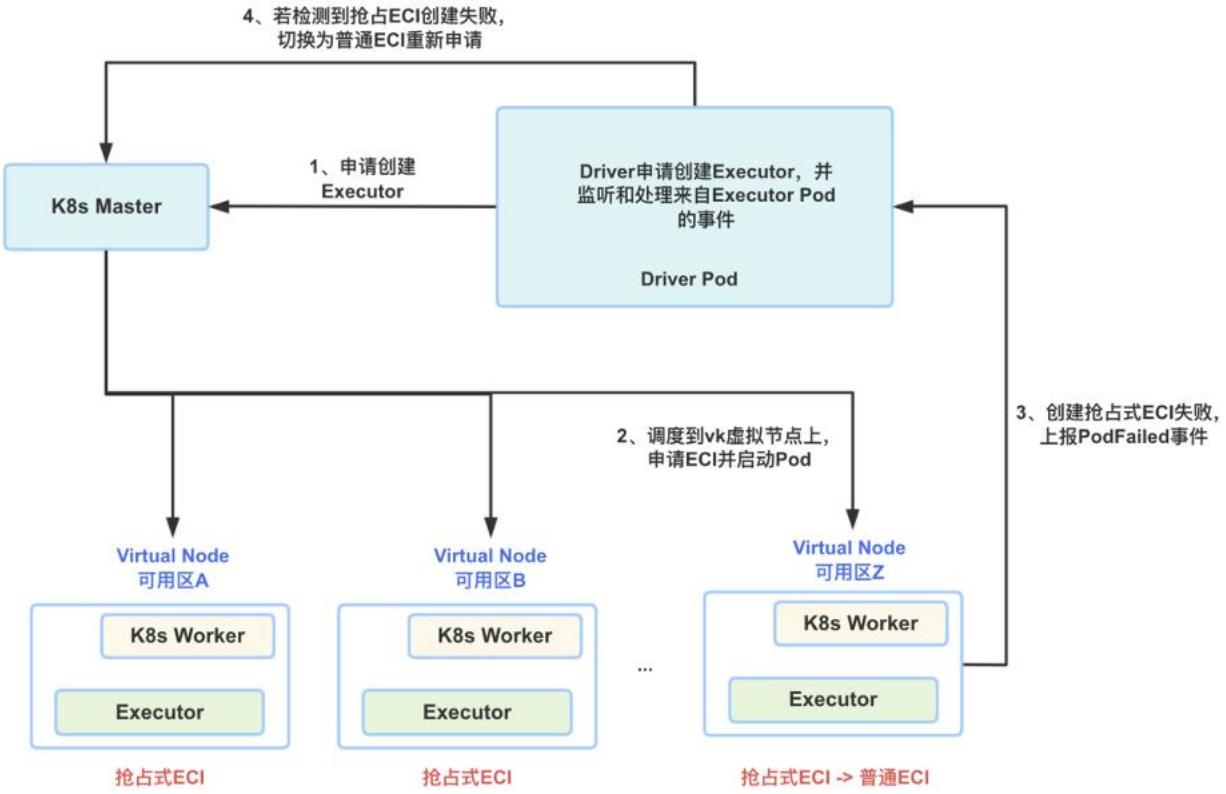


上图示意了 Spark 任务在 ACK 集群上如何申请和使用 ECI，使用前提是在集群中安装 ack-virtual-node 组件，并配置好 Vswitch 等信息，在任务运行时，Executor 被调度到虚拟节点上，并由虚拟节点申请创建和管理 ECI。

ECI 分为普通实例和抢占式实例，抢占式实例是一种低成本竞价型实例，默认有 1 小时的保护期，适用于大部分 Spark 批处理场景，超出保护期后，抢占式实例可能被强制回收。

为进一步提升降本效果，充分利用抢占式实例的价格优势，我们对 Spark 进行改造，实现了 ECI 实例类型自动转换的功能。

Spark 任务的 Executor Pod 都优先运行在抢占式 ECI 实例上，当发生库存不足或其他原因无法申请创建抢占式实例，则自动切换为使用普通 ECI 实例，保证任务的正常运行。具体实现原理和转换逻辑如下图所示。



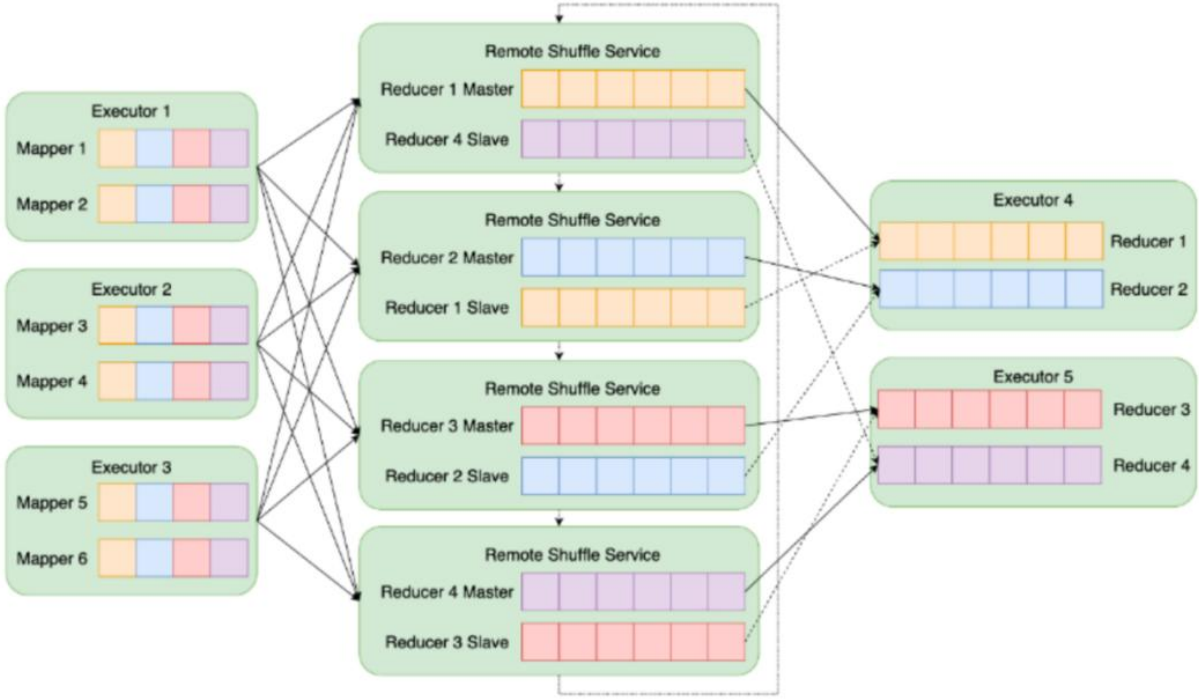
6） Celeborn

由于 K8s 节点的磁盘容量很小，而且节点都是用时申请、用完释放的，无法保存大量的 Spark Shuffle 数据。如果对 Executor Pod 挂载云盘，挂载盘的大小难以确定，考虑到数据倾斜等因素，磁盘的使用率也会比较低，使用起来比较复杂。此外，虽然 Spark 社区在 3.2 提供了 Reuse PVC 等功能，但是调研下来觉得功能尚不完备且稳定性不足。

为解决 Spark 在 K8s 上数据 Shuffle 的问题，在充分调研和对比多家开源产品后，最终采用了阿里开源的 Celeborn 方案。Celeborn 是一个独立的服务，专门用于保存 Spark 的中间 Shuffle 数据，让 Executor 不再依赖本地盘，该服务 K8s 和 Yarn 均可以使用。

Celeborn 采用了 Push Shuffle 的模式，Shuffle 过程为追加写、顺序读，提升数据读写性能和效率。

基于开源的 Celeborn 项目，我们内部也做了一些数据网络传输方面的功能增强、Metrics 丰富、监控告警完善、Bug 修复等工作，目前已形成了内部稳定版本。

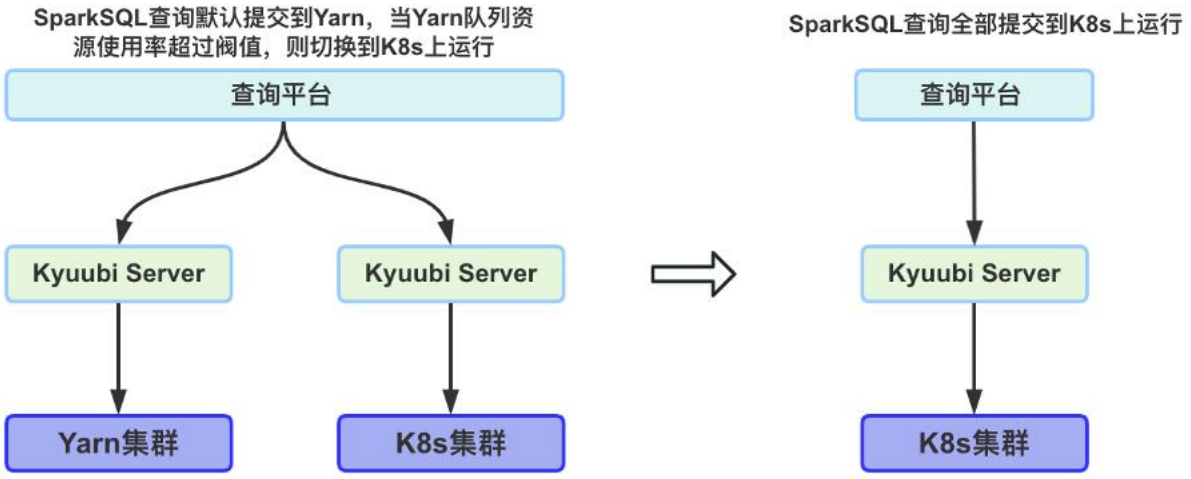


7） Kyuubi on K8s

Kyuubi 是一个分布式和多租户的网关，可以为 Spark、Flink 或 Trino 等提供 SQL 等查询服务。在早期，我们的 Spark Adhoc 查询是发送到 Kyuubi 上执行的。为了解决 Yarn 队列资源不足，用户的查询 SQL 无法提交和运行的问题，在 K8s 上我们也支持了 Kyuubi Server 的部署运行，当 Yarn 资源不足时，Spark 查询自动切换到 K8s 上运行。鉴于 Yarn 集群规模逐渐缩减，查询资源无法保证，以及保障相同的用户查询体验，目前我们已将所有的 SparkSQL Adhoc 查询提交到 K8s 上执行。

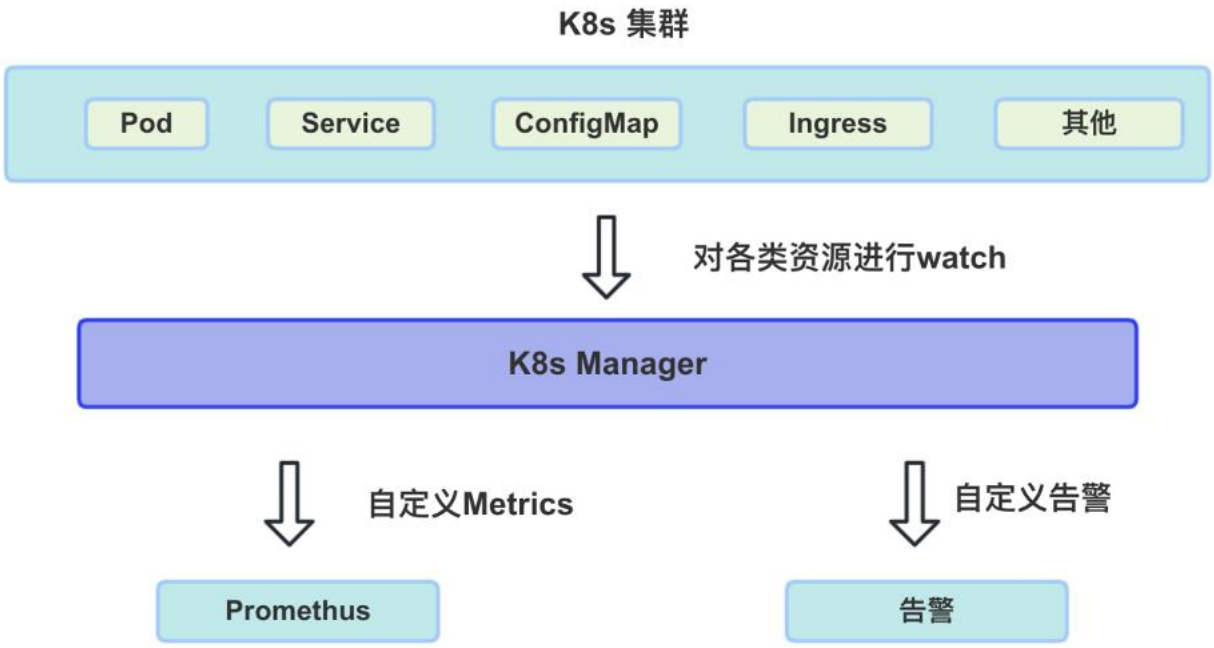
为了让用户的 Adhoc 查询也能在 K8s 上畅快运行，我们对 Kyuubi 也做了一些源码改造，包括对 Kyuubi 项目中 docker-image-tool.sh、Deployment.yaml、Dockfile 文件的改写，重定向 Log 到 OSS 上，Spark Operator 管理支持、权限控制、便捷查看任

务运行 UI 等。



8） K8s Manager

在 Spark on K8s 场景下，尽管 K8s 有集群层面的监控告警，但是还不能完全满足我们的需求。在生产环境中，我们更加关注的是在集群上的 Spark 任务、Pod 状态、资源消耗以及 ECI 等运行情况。利用 K8s 的 Watch 机制，我们实现了自己的监控告警服务 K8s Manager，下图所示为该服务的示意图。



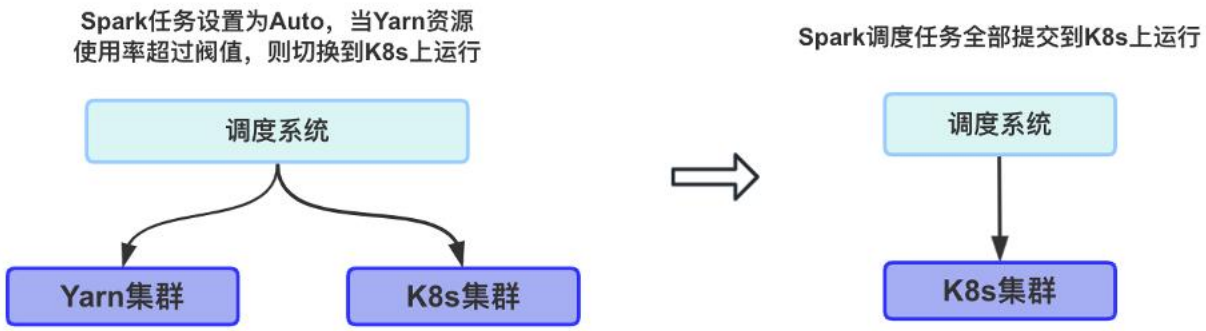
K8sManager 是内部实现的一个比较轻量的 Spring Boot 服务，实现的功能就是对各个

K8s 集群上的 Pod、Quota、Service、ConfigMap、Ingress、Role 等各类资源信息监听和汇总处理，从而生成自定义的 Metrics 指标，并对指标进行展示和异常告警，其中包括集群 CPU 与 Memory 总使用量、当前运行的 Spark 任务数、Spark 任务内存资源消耗与运行时长 Top 统计、单日 Spark 任务量汇总、集群 Pod 总数、Pod 状态统计、ECI 机器型号与可用区分布统计、过期资源监控等等，这里就不一一列举了。

9） 其他工作

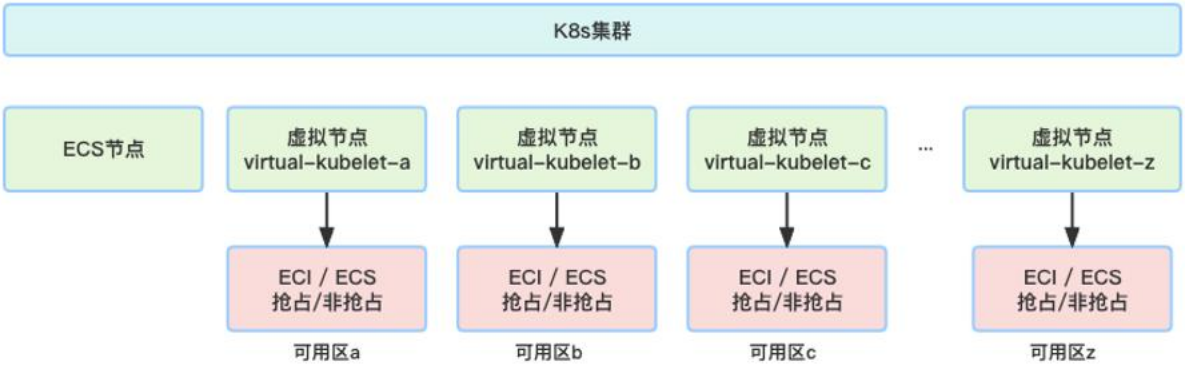
9.1 调度任务自动切换

在我们的调度系统中，Spark 任务支持配置 Yarn、K8s、Auto 三种执行策略。如果用户任务指明了需要运行使用的资源管理器，则任务只会在 Yarn 或 K8s 上运行，若用户选择了 Auto，则任务具体在哪里执行，取决于当前 Yarn 队列的资源使用率，如下图所示。由于总任务量较大，且 Hive 任务也在不断迁移至 Spark，目前仍然有部分任务运行在 Yarn 集群上，但最终的形态所有任务将由 K8s 来托管。



9.2 多可用区、多交换机支持

Spark 任务运行过程中大量使用 ECI，ECI 创建成功有两个前提条件：1、能够申请到 IP 地址；2、当前可用区有库存。 实际上，单个交换机提供的可用 IP 数量有限，单个可用区拥有的抢占式实例的总个数也是有限的，因此在实际生产环境中，无论是使用普通 ECI 还是 Spot 类型的 ECI，比较好的实践方式是配置支持多可用区、多交换机。



9.3 成本计算

由于在 Spark 任务提交时，都已明确指定了每个 Executor 的 Cpu、Memory 等型号信息，在任务结束 SparkContxt 关闭之前，我们可以从任务的中拿到每个 Executor 的实际运行时长，再结合单价，即可计算出 Spark 任务的大致花费。由于 ECI Spot 实例是随着市场和库存量随时变动的，该方式计算出来的单任务成本是一个上限值，主要用于反映趋势。

9.4 优化 Spark Operator

在上线初期任务量较少时，Spark Operator 服务运行良好，但随着任务不断增多，Operator 处理各类 Event 事件的速度越来越慢，甚至集群出现大量的 ConfigMap、Ingress、Service 等任务运行过程中产生的资源无法及时清理导致堆积的情况，新提交 Spark 任务的 Web UI 也无法打开访问。

发现问题后，我们调整了 Operator 的协程数量，并实现对 Pod Event 的批量处理、无关事件的过滤、TTL 删除等功能，解决了 Spark Operator 性能不足的问题。

9.5 升级 Spark K8s Client

Spark3.2.2 采用 fabric8(Kubernetes Java Client)来访问和操作 K8s 集群中的资源，默认客户端版本为 5.4.1，在此版本中，当任务结束 Executor 集中释放时，Driver 会大量

发送 Delete Pod 的 Api 请求到 K8s Apiserver 上，对集群 Apiserver 和 ETCD 造成较大的压力，Apiserver 的 cpu 会瞬间飙高。

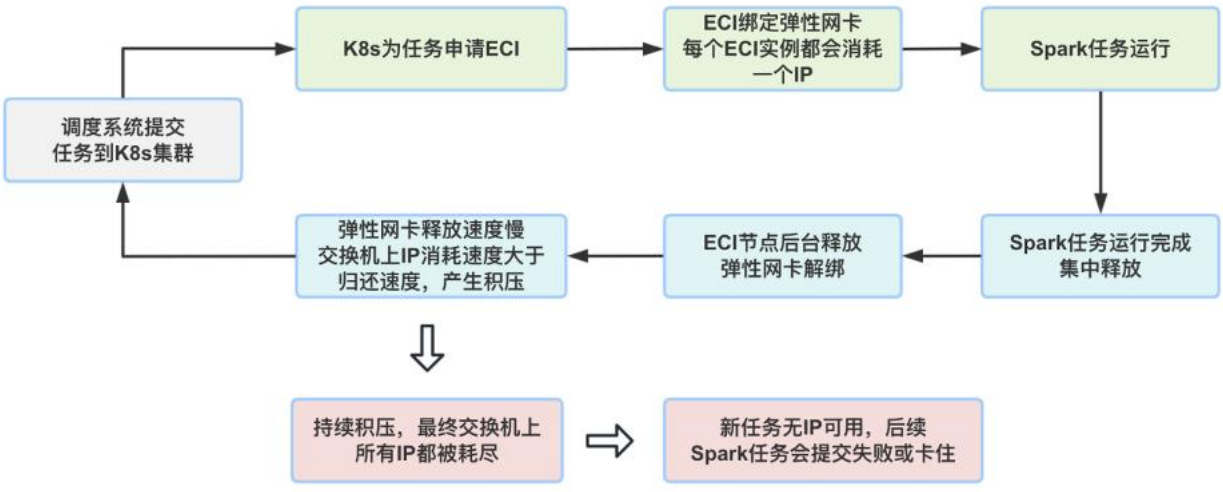
目前我们的内部 Spark 版本，已将 kubernetes-client 升级到 6.2.0，支持 pod 的批量删除，解决 Spark 任务集中释放时，由大量的删除 Api 请求操作的集群抖动。

3. 问题与解决方案

在整个 Spark on K8s 的方案设计以及实施过程中，我们也遇到了各种各样的问题、瓶颈和挑战，这里做下简单的介绍，并给出我们的解决方案。

1） 弹性网卡释放慢

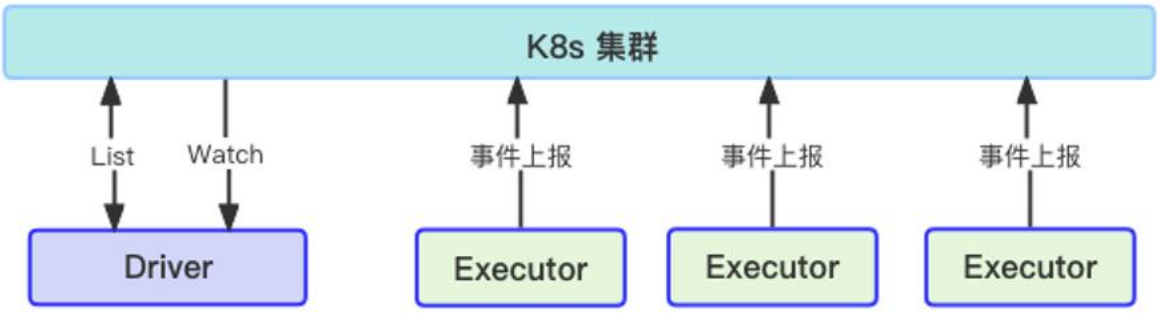
弹性网卡释放速度慢的问题，属于 ECI 大规模应用场景下的性能瓶颈，该问题会导致交换机上 IP 的剧烈消耗，最终导致 Spark 任务卡住或提交失败，具体触发原因如下图所示。目前阿里云团队已通过技术升级改造解决，并大幅提升了释放速度和整体性能。



2） Watcher 失效

Spark 任务在启动 Driver 时，会创建对 Executor 的事件监听器，用于实时获取所有 Executor 的运行状态，对于一些长时运行的 Spark 任务，这个监听器往往会由于资源过

期、网络异常等情况而失效，因此在此情况下，需要对 Watcher 进行重置，否则任务可能会跑飞。该问题属于 Spark 的一个 Bug，当前我们内部版本已修复，并将 PR 提供到了 Spark 社区。



3） 任务卡死

如上图所示，Driver 通过 List 和 Watch 两种方式来获取 Executor 的运行状况。Watch 采用被动监听机制，但是由于网络等问题可能会发生事件漏接收或漏处理，但这种概率比较低。List 采用主动请求的方式，比如每隔 3 分钟，Driver 可向 Apiserver 请求一次自己任务当前全量 Executor 的信息。

由于 List 请求任务所有 Pod 信息，当任务较多时，频繁 List 对 K8s 的 Apiserver 和 ETCD 造成较大压力，早期我们关闭了定时 List，只使用 Watch。当 Spark 任务运行异常，比如有很多 Executor OOM 了，有一定概率会导致 Driver Watch 的信息错误，尽管 Task 还没有运行完，但是 Driver 却不再申请 Executor 去执行任务，发生任务卡死。对此我们的解决方案如下：

- 在开启 Watch 机制的同时，也开启 List 机制，并将 List 时间间隔拉长，设置每 5 分钟请求一次
- 修改 ExecutorPodsPollingSnapshotSource 相关代码，允许 Apiserver 服务端缓存，从缓存中获取全量 Pod 信息，降低 List 对集群的压力

4） Celeborn 读写超时、失败

ApacheCeleborn 是阿里开源的一款产品，前身为 RSS(Remote Shuffle Service)。在早期成熟度上还略有欠缺，在对网络延迟、丢包异常处理等方面处理的不够完善，导致线上出现一些有大量 Shuffle 数据的 Spark 任务运行时间很长、甚至任务失败，以下三点是我们针对此问题的解决办法。

- 优化 Celeborn，形成内部版本，完善网络包传输方面的代码
- 调优 Celeborn Master 和 Worker 相关参数，提升 Shuffle 数据的读写性能
- 升级 ECI 底层镜像版本，修复 ECI Linux 内核 Bug

5） 批量提交任务时，Quota 锁冲突

为了防止资源被无限使用，我们对每个 K8s 集群都设置了 Quota 上限。在 K8s 中，Quota 也是一种资源，每一个 Pod 的申请与释放都会修改 Quota 的内容(Cpu/Memory 值)，当很多任务并发提交时，可能会发生 Quota 锁冲突，从而影响任务 Driver 的创建，任务启动失败。

应对这种情况导致的任务启动失败，我们修改 Spark Driver Pod 的创建逻辑，增加可配置的重试参数，当检测到 Driver Pod 创建是由于 Quota 锁冲突引起时，进行重试创建。Executor Pod 的创建也可能会由于 Quota 锁冲突而失败，这种情况可以不用处理，Executor 创建失败 Driver 会自动申请创建新的，相当于是自动重试了。

6） 批量提交任务时，UnknownHost 报错

当瞬时批量提交大量任务到集群时，多个 Submit Pod 会同时启动，并向 Terway 组件申请 IP 同时绑定弹性网卡，存在一定概率出现以下情况，即 Pod 已经启动了，弹性网卡也绑定成功但是实际并没有完全就绪，此时该 Pod 的网络通信功能实际还无法正常使用，任务访问 Core DNS 时，请求无法发出去，Spark 任务报错 UnknownHost 并运行失败。该问题我们通过下面这两个措施进行规避和解决：

- 为每台 ECS 节点，都分配一个 Terway Pod
- 开启 Terway 的缓存功能，提前分配好 IP 和弹性网卡，新 Pod 来的直接从缓存池中获取，用完之后归还到缓存池中

7） 可用区之间网络丢包

为保障库存的充足，各 K8s 集群都配置了多可用区，但跨可用区的网络通信要比同可用区之间通信的稳定性略差，即可用区之间就存在一定概率的丢包，表现为任务运行时长不稳定。对于跨可用区存在网络丢包的现象，可尝试将 ECI 的调度策略设定为 VSwitchOrdered，这样一个任务的所有 Executor 基本都在一个可用区，避免了不同可以区 Executor 之间的通信异常，导致的任务运行时间不稳定的问题。05

4. 总结与展望

最后，非常感谢阿里云容器、ECI、EMR 等相关团队的同学，在我们整个技术方案的落地与实际迁移过程中，给予了非常多的宝贵建议和专业的技术支持。

目前新的云原生架构已在生产环境上稳定运行了近一年左右的时间，在未来，我们将持续对整体架构进行优化和提升，主要围绕以下几个方面：

- 持续优化云原生的整体方案，进一步提升系统承载与容灾能力
- 云原生架构升级，更多大数据组件容器化，让整体架构更加彻底的云原生化
- 更加细粒度的资源管理和精准的成本控制

第三章

容器 AI 工程化创新

智算时代，基于 ACK 落地云原生 AI

作者：张凯，阿里云高级技术专家

1. 背景

以 GPT（Generative Pre-trained Transformer）和 Diffusion model 为代表的大语言模型（Large language model, LLM）和生成式人工智能（Generative artificial intelligence, GAI）在过往两年，将人们对 AI 的梦想与期待推向了一个新高峰。这一次，AI 带来的“智能”效果和“涌现”能力，吸引着千行百业都在积极思考如何在业务中利用大模型。即便训练一次千亿参数量模型的成本可能就高达百万美元，依然有很多企业希望拥有自己的专属大模型。

今天的云计算已经承载了从业务应用，到数据库、大数据、机器学习和高性能计算等大多数计算负载。面对 LLM 和 GAI 这类对算力和数据都有极高量级需求的新负载，云计算也迎来了“智算”时代。一方面以服务化资源池的方式提供千 GPU 卡，甚至万卡的算力，PB 级存储和单机 Tb 级高性能网络互联，另一方面以云原生标准化交付算力给大模型的生产者和使用者。

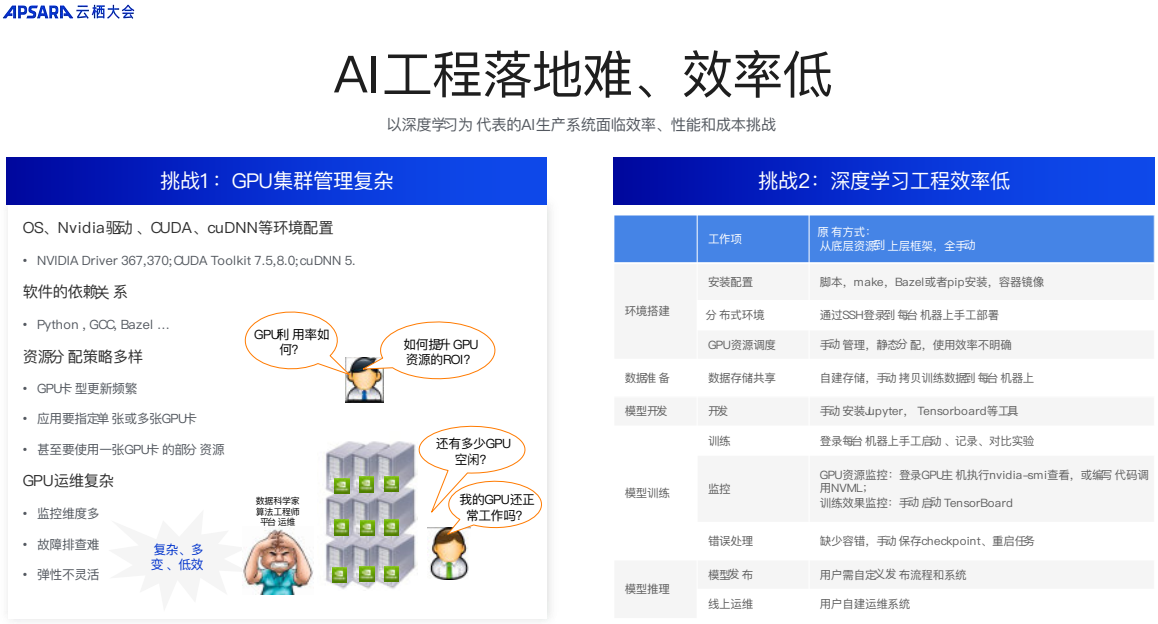
阿里云云原生容器服务借助容器、Kubernetes、微服务等云原生技术，在阿里云弹性计算、存储、网络和灵骏智算服务基础之上，推出了 ACK 云原生 AI 套件产品和解决方案，帮助企业在“智算”时代，更快、更高效地落地云原生 AI 系统。

此次云栖大会的分享共分三部分，介绍云原生 AI 领域的基础知识。首先，解析云原生 AI 所遇到的技术提站和应对方案，随后介绍云原生 AI 领域的关键技术与架构细节，最后分享我们在 ACK 的相关经验及工程实践。

2. 大模型带来的技术挑战

近年来，深度学习引领 AI 的再次快速发展，并已成为主流方向。其中涉及到多种技术，包括有监督学习、强化学习、无监督学习等。以深度学习为代表的技术已在计算机视觉、语音识别等领域取得巨大进展，推动了许多行业的创新。通过云化服务的方式，交付 AI 能力也成为一种趋势。

总结深度学习任务的特性包括：首先，它是一种端到端的工作流程，旨在从大量原始数据中提取特征，进而创建可执行模型。其次，通常需要长时间的训练，任务运行时间可能以小时，甚至以月为单位。在训练过程中，需反复迭代优化，不断调整参数，以使模型能够更好地接近真实数据特征分布。这会导致算力资源和数据的巨大消耗。



理解了深度学习为代表的 AI 工作负载特性后，我们可以发现在实际生产和应用中，AI 和深度学习工程会面临几个主要挑战，尤其是在效率、性能和成本方面。

例如，管理 GPU 集群是一项非常复杂的工作。只有少量 GPU 机器时，个人数据科学家或计算工程师可以独立操作和维护。然而，随着 GPU 数量增长到数十台甚至数百台，形成大规模 GPU 集群时，环境一致性、版本依赖性、配置多样性、效率提升以及运维生命周期复杂度等问题将变得尤为突出。许多算法工程师不得不花费大量时间和精力解决 GPU 集群管理和资源分配等基础问题，从而拖延真正用于深度学习模型开发和训练的工作。

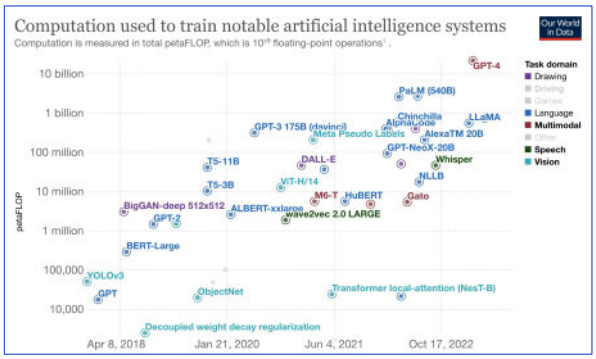
另一方面，即使解决了 GPU 管理问题，AI 工程效率仍然受到生命周期复杂性的限制。整个流程，从开发环境搭建、数据准备到模型开发、模型训练、优化、可视化控制、错误处理等，直到模型推理上线和运维，每一个环节都可能导致 AI 模型的开发周期延长且效率低下。

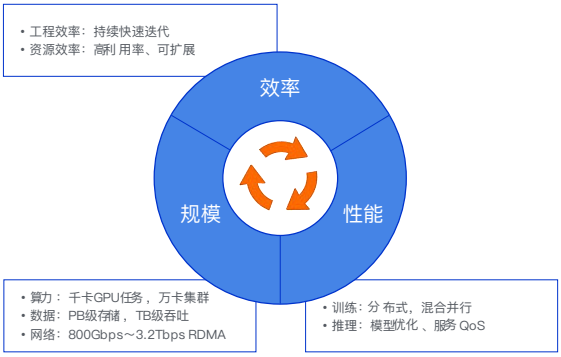
而且这些环节大多需要手动操作或使用各种脚本串联，开发效率较低，协同难度大。因此，提高 AI 的工程效率是另一个亟待解决的问题。

APSARA 云栖大会

大模型对基础设施带来更多挑战

- 大模型对基础设施服务 能力 的挑战是阶跃式的。
- 对“ 规模、性能、效率”的要求，成为 LLM/AIGC快速落地的高门槛。





GPT3：175B参数，单次训练使用45TB数据，近千卡 A100/1个月，成本数百万美元。

当面对 LLM 和 AIGC 等新领域，AI 对基础设施服务能力提出了更多挑战。主要体现在规模、性能和效率三个方面。

在规模上，要训练出具有广泛知识和专业领域理解及推理能力的大语言模型，往往需要高达万卡级别的 GPU 集群和 PB 级的数据存储以及 TB 级的数据吞吐。此外，高性能网络也将达到单机 800Gbps 甚至 3.2Tbps 的 RDMA 互联。

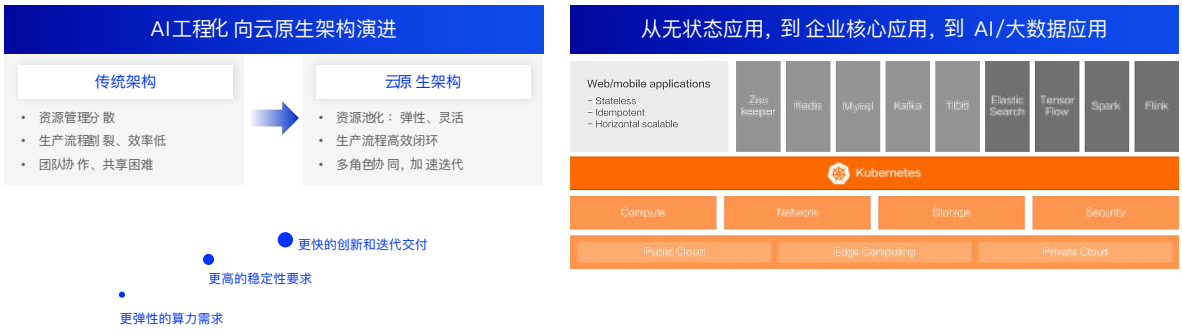
在性能方面，随着模型体积和参数量的增长，单张显卡已无法承载完整的模型。因此需要使用多张显卡进行分布式训练，并采用各种混合并行策略进行加速。这些策略包括数据并行、模型并行、流水线并行以及针对语言模型的序列并行等，以及各种复杂的组合策略。

在推理阶段，模型需要提供高效且稳定的推理服务，这需要不断优化其性能，并确保服务质量（QoS）得到保证。

在此基础上，最重要的目标是提高资源效率和工程效率。一方面，持续提高资源利用效率，并通过弹性扩展资源规模，以应对突发的计算需求。另一方面，要最优化算法人员的工作效率，提高模型迭代速度和质量。因此，资源效率和工程效率成为了最优先考虑的优化目标。

APSARA 云栖大会

基于容器的AI/大数据成为 云原 生时代的技术趋势



- Gartner 预测：到 2023 年 70% 的 AI 应用是基于容器和 Serverless 技术开发 。
- IDC预测：By 2025, Nearly 50% of All Accelerated Infrastructure for Performance-Intensive Computing (AI, HPC, and Big Data Analytics) will Be Cloud Based as These Systems Are Increasingly Integrated with Enterprise Software.

随着云原生技术和架构发展，我们明显观察到 IT 架构的变化。传统的企业级应用、Web 应用、微服务等领域都在从传统架构转向云原生架构。互联网应用大多是基于容器、Kubernetes，Prometheus 等云原生技术实现的，追求弹性、灵活性以及最佳性价比。同时，通过标准化交付，提升生产流程的高效闭环。

在标准 API 和标准架构的指导下，进一步提高多角色之间的协作和迭代效率。同样地，为了获得更多弹性算力供给、更高稳定性保证以及更快的交付，越来越多 AI 和大数据工作负载也运行在云原生架构上。

右图清晰地显示了这一趋势：最早是从可水平扩展的无状态应用程序（如 Web 应用、移动后端应用）开始；然后是 NoSQL 数据库、关系数据库、TensorFlow、PyTorch、Spark、Flink 等大数据和典型机器学习的 AI 计算任务。都能够在 Kubernetes 容器集群上大规模运行。

三方研究机构的预测，显示出同样的趋势。早在 2019 年，Gartner 就曾预测，到 2023 年，70%的人工智能应用程序将基于云原生等技术进行开发。IDC 的研究则预测，到 2025 年，

接近 50%的企业内部的数据密集型或性能密集型计算工作负载都将迁移到基于云的架构上，而基于云的架构正是典型的 Cloud Native 云原生架构。

此前概述了传统的机器学习、深度学习以及目前流行的 LLM 和 AIGC ，带来了一些技术上的挑战，和对基础设施造成的压力。为了应对这些挑战，我们期待借助云原生 AI 来找到解决方案。那么，什么是云原生 AI 呢?

云原生 AI 是指利用云计算的弹性资源、异构算力以及容器、自动化、微服务等云原生技术，提升 AI/ML 的工程效率，降低整体成本，提高可扩展性，并实现端到端的解决方案。在这其中，我们尤为注重工程效率、成本效益、可扩展性和可复制性等因素。

云原生 AI 最核心的两个应用场景:
统一管理异构资源，提升资源利用率。

对 IaaS 云服务或者客户 IDC 内各种异构的计算（如 CPU，GPU，NPU，VPU，FPGA，ASIC）、存储（OSS，NAS， CPFS，HDFS）、网络（TCP， RDMA）资源进行抽象，统

一管理、运维和分配，通过弹性和软硬协同优化，持续提升资源利用率。

通过统一工作流和调度，实现 AI、大数据等多类复杂任务的高效管理。

兼容 Tensorflow，Pytorch，Horovod，ONNX，Spark，Flink 等主流或者用户自有的各种计算引擎和运行时，统一运行各类异构工作负载流程，统一管理作业生命周期，统一调度任务工作流，保证任务规模和性能。一方面不断提升运行任务的性价比，另一方面持续改善开发运维体验和工程效率。

围绕这两个核心场景，可以扩展出更多用户定制化场景，比如构建符合用户使用习惯的 MLOps 流程；或者针对 CV 类（Computer Vision，计算机视觉）AI 服务特点，混合调度 CPU，GPU，VPU（Video Process Unit）支持不同阶段的数据处理流水线；还可以针对大模型预训练和微调场景，扩展支持更高阶的分布式训练框架，配合任务和数据调度策略，以及弹性、容错方案，优化大模型训练任务成本和成功率

接下来探讨支持这些场景，云原生 AI 所需的主要功能。

首先，从异构资源管理的角度，需要一键部署和运行的能力，能够统一管理和操作各种类型的异构资源，如 CPU、GPU 以及各种虚拟化的设备和专业存储、网络设备。在运维过程中，需要多维度的异构资源可观测性，包括监控、健康检查、告警、自愈等自动化运维能力。对于“宝贵”的计算资源，如 GPU 和 NPU 等加速器，需要通过各种调度、隔离和共享的方法，最大限度地提高其利用率。在存储和网络方面，通过统一接口和方式供给上层工作负载。在此过程中，我们还将利用云资源的弹性特征，持续提高资源的交付和使用效率。

另一个主要能力是能够在分钟级内准备好开发环境和集群测试环境，帮助算法工程师开始执行深度学习任务。把端到端的 AI 生产过程通过相同的编程模型、运维方式进行交付。在整个流水线执行过程中，天然支持主流计算框架，如 TensorFlow、PyTorch、Deepspeed 等。

对于大规模分布式 AI 任务，提供丰富的任务调度策略，如 Gang scheduling、Capacity scheduling、Topology aware scheduling、优先级队列等。并使用工作流或数据流的方式串联起整个任务流水线。

此外，在计算框架与算法层面适配资源弹性能力，提供弹性训练和弹性推理服务，优化任务整体运行成本。除了计算任务优化，还应关注数据使用效率的优化。为此，需要统一的数据集管理、模型管理和访问性能优化等功能，并通过标准 API 和开放式架构使其易于被业务应用程序集成。

随着 AI 和 LLM 的不断发展，我们还将面临一些新的挑战和需求。例如，如何快速适应新的开源大模型训练方法（无论是预训练、监督微调还是强化学习），以及如何提高大模型推理性能并确保其质量和稳定性。同时，也需要关注一些前沿技术和创新能力，通过标准化和可编程的方式来集成，不断迭代业务应用，形成 AI+ 或 LLM+ 的新应用开发模式和编程模型。

例如 AutoGPT、多智能体任务等，这些都是我们需要快速掌握、理解，并应用于应用智能化升级的重要工具。

3. 云原生 AI 支持大模型生产的关键技术

前述详细分析了云原生 AI 领域的发展历程、现状以及未来趋势，帮助大家更好地理解这一交叉技术领域。后续我们将转向更为具体的技术层面，介绍已经落地并相对成熟的一些云原生 AI 关键技术。

首先，介绍整个云原生 AI 的系统架构，这是一个典型的分层架构。最底层是云资源服务或数据中心的线下资源，由容器服务平台进行统一的封装和管理。在这一层之上，又分为几个层面来构建云原生 AI 系统。

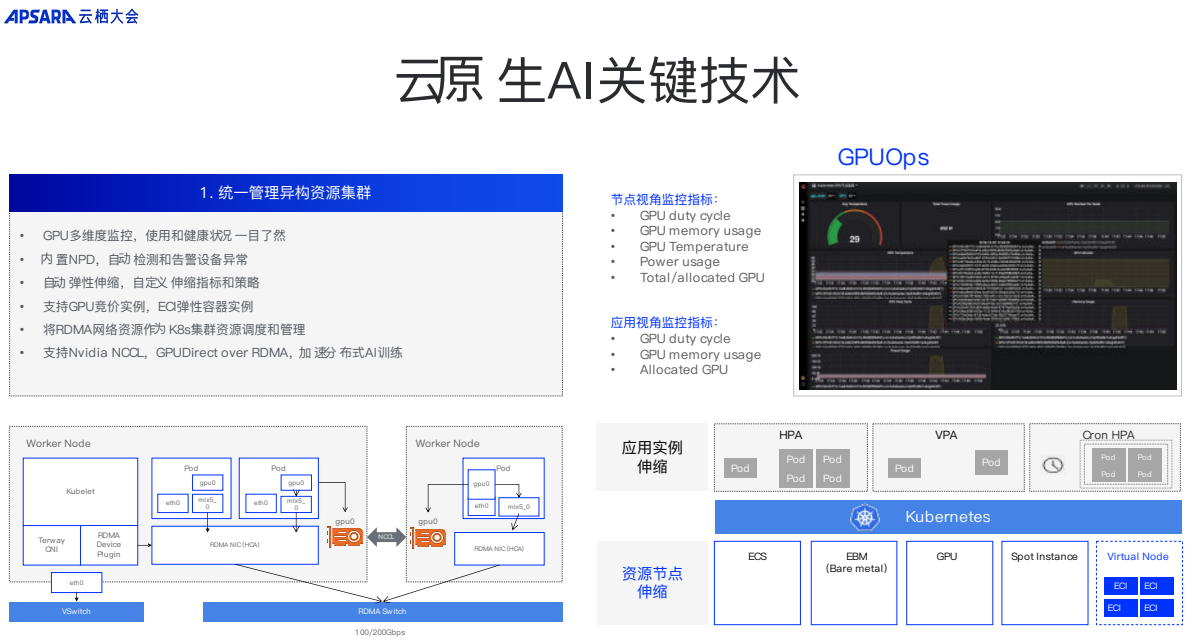
- 第一层是高异构资源管理层，包括对 AI 计算、存储和网络资源的统一管理和运维。
- 第二、三层负责 AI 任务的调度和流水线的构建，支持各种计算框架和训练、推理运行时。

- 第四、五层是任务性能优化和 AI 作业生命周期管理。
- 最后一层是通过统一的工具链和标准 API 向上提供所有这些能力，并与内外部生态集成，包括开源模型、数据，私有业务系统或服务，以及第三方生态系统。

在整个系统中，弹性、运维和安全贯穿于各个层面。此外，我们还注重数据、模型和实验等各种制品的统一管理，以及安全性和隐私性的保障。

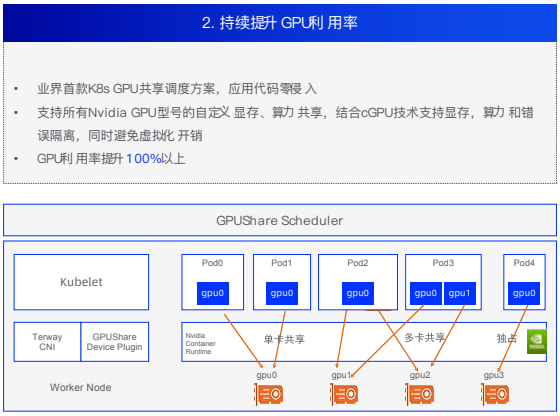
在阿里云的容器服务（ACK）之上，我们提供了云原生 AI 套件的产品，以此作为上述系统架构的一种参考实现。帮助大家更容易理解云原生 AI 系统分层架构的构建方式以及每层的关键技术点。

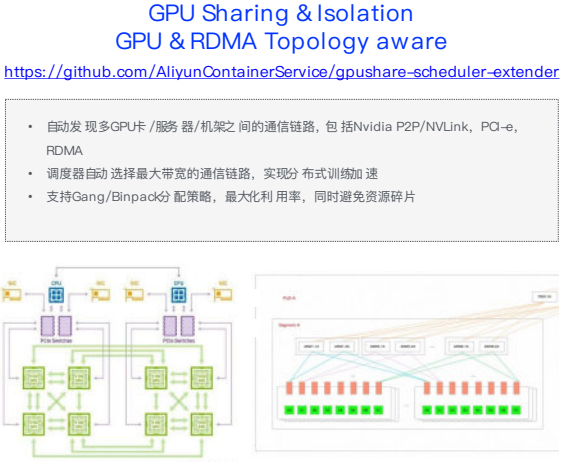
右图展示了基于 ACK 的云原生 AI 套件产品的系统架构。每个方块代表一层的关键技术组件，整个架构是可以组件化拼装、交付和扩展的。用户可以通过组件插拔组合的方式来定制自己的云原生 AI 平台。



APSARA 云栖大会

云原 生AI关键技术





我们将自底向上地阐述每一层的关键技术点。

首先，在统一资源管理这一层，实现对各类异构资源（如 GPU、NPU）进行统一运维、多维度监控和健康检查，并具备自动异常发现和告警功能。此外，还需要追求性价比、弹性交付能力和自动弹性伸缩能力，充分利用云计算的价格优势和技术优势。

其次，使用阿里云提供的竞价实例、按需服务等弹性资源，以降低成本并提高性价比。

在高性能网络方面，采用 KBS 集群统一调度和管理 RDMA 和 GPU 等高性能设备，进行资源抽象与运维屏蔽，实现网络与计算的高效协同工作。

再次，我们需要对 Nvidia GPU 进行全方位精细化支持，包括对 NVIDIA Direct 和 NCCL 的支持，以及针对多 GPU 设备场景的网络拓扑感知调度策略，优化通信效率，加速分布式训练。

在 GPU 资源利用率优化方面，提供 GPU 共享调度方案，使得多个容器可以在同一张 GPU 卡上共享显存和算力，显著提升 GPU 利用率。同时，我们还将结合 阿里云 cGPU 技术实现轻量级设备资源隔离化，确保显存、算力和错误隔离，并最大限度地节省虚拟化开销。

通过以上关键技术点提供坚实的基础支撑，不断优化高性能资源的成本效益，最终提升整体训练效率。

上升到 AI 任务调度层。如果我们理解 AI 或大数据作业的特征，就会发现它们不是简单、独立的多副本组合，而是相互依赖、有拓扑关联的批量任务。然而，Kubernetes 原生调度器支持批量任务的能力相对缺乏，例如复杂的任务调度与抢占、资源额度分配、优先级管理和调度性能优化。

Kubernetes 调度器框架已经发展成一个可插拔式的体系结构，允许我们通过 Plug-ins 进行扩展，以实现更复杂的任务级调度能力。ACK 团队已经在 Kubernetes 调度器中贡献了多个 Plug-ins，如 gang scheduling、弹性容量调度、优先级队列、公平调度、拓扑感知调度等，以最大化满足复杂的训练或推理任务的整体调度效率和成功率。

这些调度能力可以有效解决复杂任务编排和不合理的资源分配导致的资源浪费，以及不同租户作业之间的资源争抢等问题。此外，我们也扩展了数据亲和性的调度策略，让数据与计算更加紧密耦合，减少数据传输的带宽压力和延迟。

阿里云还在 Kubernetes 社区推动成立 Batch Job 工作组，致力于制定标准规范，定义标

准 API，以便能够高效结合这些复杂的任务管理和调度功能。同时，我们将一些主要的批量调度或任务级别调度策略插件贡献给上游开源社区，并已被众多社区用户使用。例如 Open AI 在其高达 7500 节点的模型训练集群中使用了 Co-scheduling 调度功能。

总的来说，在调度方面，仍有许多更复杂、更高级的功能需要继续加强，如租户间的公平性、虚拟配额管理、任务级抢占等。这些能力将通过 ACK 云原生 AI 套件持续提供给用户和社区。

在解决了任务调度的问题后，我们将探讨如何将训练任务或推理服务与云资源的弹性相结合，从而使训练过程和推理服务能够伸缩自如。

我们可以通过自动或手动方式，在有更多资源闲置的情况下，将这些资源动态添加到训练任务中。在资源紧张时，则可以出让一些资源供给其他高优先级的任务使用。同时确保运行中的任务不被打断，并且其执行效率和模型性能不受影响。

我们通过结合 K8S 任务管理和调度、Pytorch、Horovod 等计算框架，实现对常见的 CV 模型、NLP 模型和推荐类模型的支持，使训练任务可以使用从几张 GPU 卡动态扩展到几十张、上百张 GPU 卡，同时也可以反向缩容。确保任务在整个过程中不中断，并保持收敛性

能。

弹性训练的收益会相对明显，尤其是在使用竞价实例的场景下。虽然竞价实例的优势在于可以用较低的价格获取 GPU 或其他高性能计算资源，但也存在每小时就会回收的风险。如果运行中的任务无法在此时间段内结束，则可能会面临缩容问题。

我们可以结合弹性训练和弹性推理等能力，充分利用像 spot instance 和 ECI 这样的高性价比资源。同时，由于大模型训练任务（如 175B 参数级别的 GPT3）需要近千张 GPU 卡并训练近一个月的时间。任何一张 GPU 卡都可能出现故障，这会导致任务失败，造成极大的损失和资源浪费。

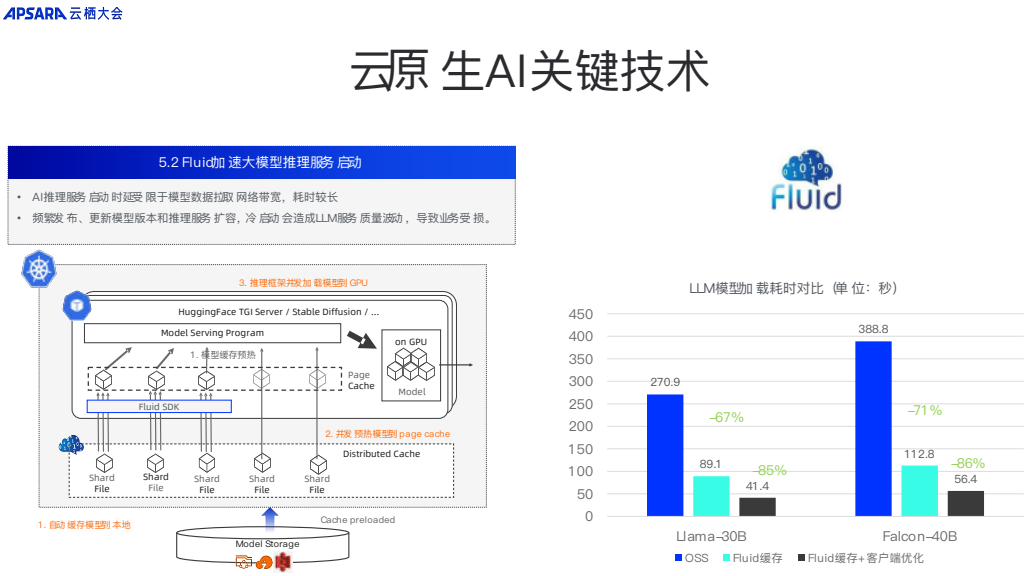
可以结合弹性训练的能力，构建容错场景，即使部分资源失效或计算任务出错，整体任务仍然可以通过缩容形式继续进行训练，以最大化容错。并确保计算资源的投入和计算过程都不会被浪费。这是一种非常有趣且富有挑战的技术问题，即如何利用云的弹性资源来适应算法和计算过程的弹性伸缩。

我们之前提到了任务调度和如何更好地使用弹性资源。其中，计算效率最大化和减少数据读取的时间是非常关键的问题。云计算中，存储和计算分离是一种常见架构，它可以提供更大的灵活性，但也带来了远程访问延迟和带宽限制的问题。

为了解决这些问题，我们通过开源项目 Fluid，结合分布式缓存技术，将数据缓存能力内置到数据集对象中，并将其作为有生命周期的对象进行管理。这样，数据集可以根据应用程序的需求进行缓存数据的亲和性调度，最大程度地减少远程数据访问延迟。还能够在自定义监控的情况下，进行缓存数据的自动弹性伸缩，缓解高并发计算任务访问远程存储的聚合带宽压力。

Fluid 弹性数据集编排和加速项目已经在 ACK 云原生 AI 套件中有相应的产品实现，一些功能子集也已开源至 CNCF 社区，目前正在积极向孵化阶段推进。

通过分布式缓存加速技术，我们可以显著提高分布式训练的效率，如右下角所示的 ResNet-50 训练效果示例。当我们不断增加 GPU 卡的数量时，如果不使用缓存加速，性能加速比并趋于平坦（蓝色线条），不会得到很好的提升。而通过 Fliud 的弹性分布式缓存加速后，随着 GPU 资源的增加，训练性能加速比基本保持线性增长（绿色线条），大幅度地提高了 GPU 计算效率。



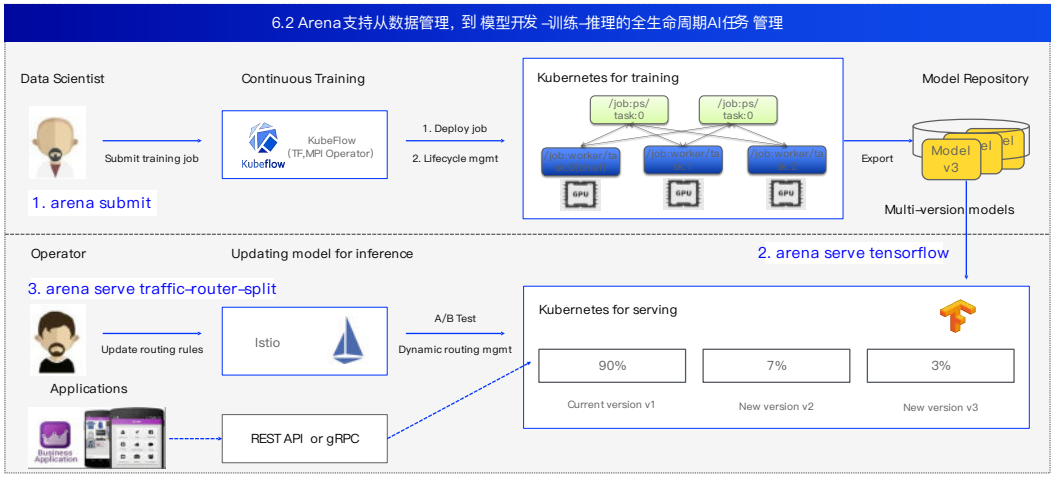
我们不仅将 Fluid 弹性数据集加速的能力应用于分布式训练场景，也可以将其应用大模型的推理服务。在大模型推理服务中存在一个问题，即随着模型体积的增长（现在模型常常达到几 GB 或几十 GB），首次创建，或在运行过程中动态扩容这样的模型服务的冷启动延迟问题会变得非常严重。

这是因为我们需要从远程对象存储或 HDFS 中拉取大模型参数，而这种操作往往具有较高的延迟。我们测试过，在一个 165GB 的大模型情况下，拉取所有参数可能需要近一个小时的时间，这在生产环境中是无法接受的，因为它不能提供在线服务。

为了解决这个问题，我们为 Fluid 缓存加速功能扩展了数据预取、并发加载、Pagecahce 预热等优化手段，并应用到模型服务冷启动的优化中。结果表明，对于 Llama 30B 等流行的大模型，可以实现 70%至 80%，甚至更高的加速启动效果。

APSARA 云栖大会

云原生AI关键技术

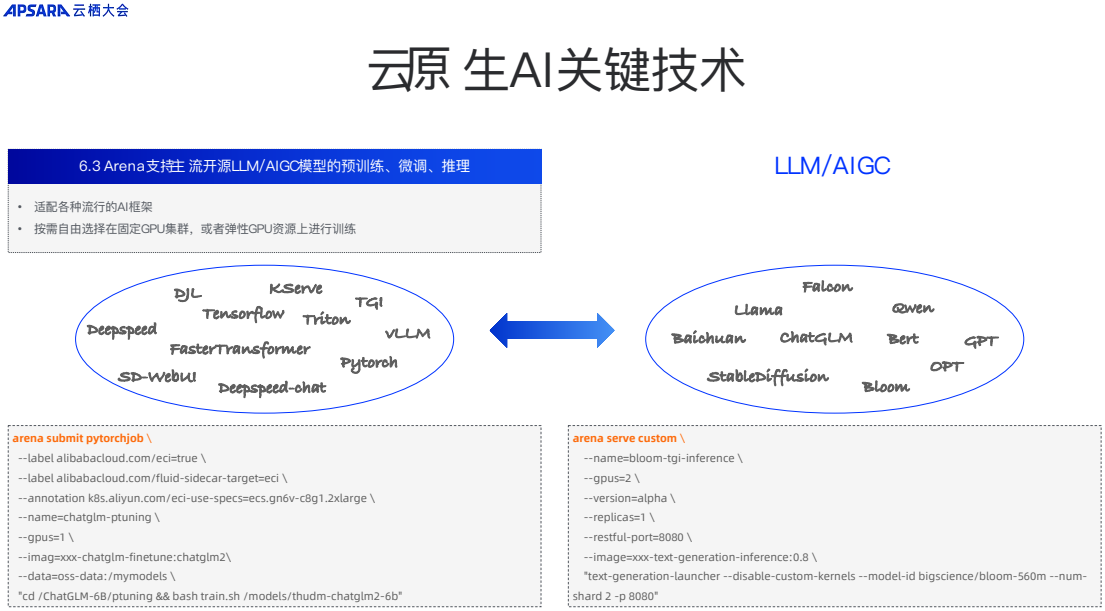


ACK 云原生 AI 套件提供的所有组件都以 Kubernetes 标准接口（CRD）和 API 形式，交付给 AI 平台开发运维人员调用。这对基于 Kubernetes 构建云原生 AI 平台来说是非常方便和易用的。

但对于数据科学家和算法工程师开发训练 AI 模型来说，Kubernetes 的语法和操作却是一种“负担”。他们更习惯在 Jupyter Notebook 等 IDE 中调试代码，使用命令行或者 Web 界面提交、管理训练任务。任务运行时的日志、监控、存储接入、GPU 资源分配和集群维护，最好都是内置的能力，使用工具就可以简单操作。

为此，我们创建了一款名为 Arena 的工具，来屏蔽底层资源、Kubernetes、运行环境等各种复杂度的能力，以统一的方式来管理系统、任务、模型开发、分布式训练、模型评估推理、循环迭代等全生命周期。它可以自动化处理复杂的任务，包括调度、Kubernetes 管理和实时监控。此外，Arena 还支持 Go、Java 和 Python SDK，支持二次开发和封装。

我们还为运维人员和开发者提供了可视化的控制台，使他们在不同终端上都能实现统一的任务管理。只需使用 Arena 的一条命令，就可以将分布式的任务提交到 Kubernetes 集群中，自动运行，并保存训练结果。另一条命令则可以用于创建模型推理服务，也包括发布、A/B 测试、流量管理等运维操作。



目前我们也为 Arena 扩展了对大模型的支持，兼容了主流的 LLM/ AIGC 模型和训练、推理框架。例如，用户可以通过两条命令快速训练和推理 ChatGLM，Llama，Bloom，Baichuan，Qwen 等模型。

4. ACK 云原生 AI 套件工程实践

前述所有这些关键技术，我们需要考虑如何将它们应用起来，怎样将它们快速应用于用户的生产环境中，以启动用户的第一个 AI 训练和推理任务。为此，在阿里云的容器服务（ACK）上，我们提供“云原生 AI 套件”，旨在将上述所有技术及架构在一个完整的产品中提供给我们的客户和合作伙伴。

云原生 AI 套件完全遵循之前介绍的分层参考架构进行实现和交付。用户可以根据自己的需求，在其中选择所需的组件。例如，如果用户对 GPU 管理有特殊要求，则可以利用异构算力中的调度、共享、隔离等能力。如果对任务管理有很多需求，可以利用我们的高级调度策略。如果您需要一个方便快捷的 AI 生产流程管线管理工具，那么 Arena 等套件的能力也可以迅速为您提供帮助。

借助这些功能，可以帮助用户实现 AI 训练速度提高 20%以上，数据访问效率提高 30%以上，而 GPU 利用率则可提高 100%以上。

ACK 的云原生 AI 套件能够带来什么效果？

这里有两个案例来说明。

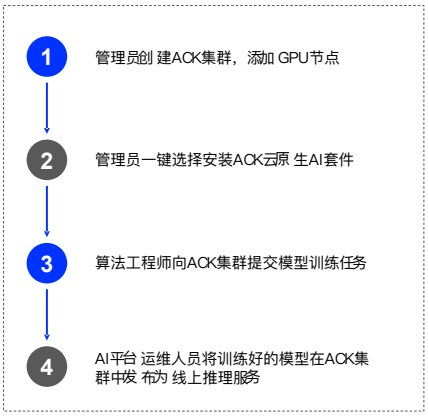
首先，任意门是一个需要大量用户多模态数据进行人工智能驱动的社交平台。客户需要一个能够充分利用云计算弹性资源并具有快速迭代和扩展能力的 AI 平台。借助 ACK 的云原生 AI 套件，任意门构建了一个定制化的基于容器的 AI 平台。现在，该平台已经承载了客户的语音合成、人脸匹配、图像识别和智能聊天等业务场景。

第二个案例是小米的机器学习平台。

客户采用混合云方式，既使用自有 IDC 中的 GPU 资源，又使用阿里云的弹性 GPU 资源，构建 AI 平台。由于跨地域，使得计算存储分离的架构，在数据复制延迟和聚合带宽的问题非常突出。通过使用 Fluid 的分布式缓存加速和数据统一管理方案，解决了线下自建 StarFS 存储的标准化接入和管理问题，并增加了分布缓存线下数据能力，以解决数据加载性能问题。帮助小米构建了一个高效混合云架构下的分布式 AI 平台。

APSARA 云栖大会

ACK云原 生AI套件使用流程

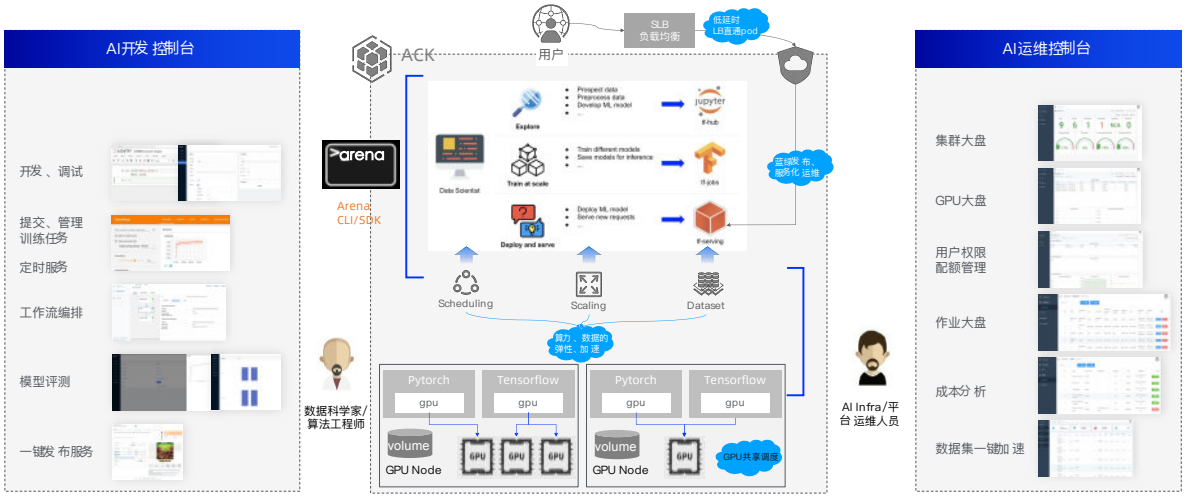


如何使用 ACK 的云原生 AI 套件？只需在阿里云创建一个 ACK 集群，无论是否添加 GPU 节点。一键安装即可将整个云原生 AI 套件部署到集群中。用户可以根据需求选择所需组件，以定制自己的基础 AI 平台。在该平台上进行二次开发和扩展后，开发人员可以通过 Arena 或 Jupyter Notebook 进行模型开发，并向 GPU 集群提交模型训练任务。同时，AI 平台的运维人员可以通过AI套件中的运维控制台和运维命令进行模型生产和线上推理服务的便捷运维。

APSARA 云栖大会

ACK云原 生AI套件使用流程

两类角色通过命令行工具 和控制台 简便操作，高效协 同



AI 平台或 AI Infra 运营人员如何使用 ACK 云原生 AI 套件进行工作的流程： 首先，创建 ACK 集群，接着配置调度策略、创建训练数据集或模型集，并通过 AI 运维控制台监控集群运行状况、GPU 分配情况和健康状况等。

作为开发团队的数据科学家和算法工程师，则通过 Arena 命令行或 SDK 进行模型开发、训练的迭代实验。最后通过评测和压测，选择出符合预期的模型版本，通过 Arena 创建推理服务，将其发布到生产集群中。我们还提供了 AI 开发控制台，以便于不习惯使用命令行或无需二次开发需求的用户，在可视化界面上完成所有开发、训练任务的操作。

ACK 是基于阿里云容器服务托管的 Kubernetes 服务，经过大量国内外客户的验证。ACK 在今年首次参加 Gartner 年度容器管理魔力象限评选，并荣幸地进入了领导者象限，是亚洲地区唯一入选该象限的云服务商。

在 ACK 平台上，我们提供了稳定高效的托管 Kuberntes 集群，以及云原生 AI 能力、服务网格能力、多集群、多云管理能力、边缘计算能力、专有云和混合云交付能力，以及对大规模计算服务的高效支持。所有这些能力都以标准 Kubernetes API 和架构提供出来。

欢迎广大用户和合作伙伴在 ACK 容器服务上，利用云原生 AI 套件快速构建自己的云原生 AI 平台。

云原生场景下，AIGC 模型服务的工程挑战和应对

作者：车漾，阿里云高级技术专家

“成本”、“性能”和 “效率”正在成为影响大模型生产和应用的三个核心因素，也是企业基础设施在面临生产、使用大模型时的全新挑战。AI 领域的快速发展不仅需要算法的突破，也需要工程的创新。

1. 大模型推理对基础设施带来更多挑战

首先，AI 商业化的时代，大模型推理训练会被更加广泛的使用。比较理性的看待大模型的话，一个大模型被训练出来后，无外乎两个结果，第一个就是这个大模型没用，那就没有后续了；另一个结果就是发现这个模型很有用，那么就会全世界的使用，这时候主要的使用都来自于推理，不论是 openAI 还是 midjourney，用户都是在为每一次推理行为付费。随着时间的推移，模型训练和模型推理的使用比重会是三七开，甚至二八开。应该说模型推理会是未来的主要战场。

大模型推理是一个巨大的挑战，它的挑战体现在成本、性能和效率。其中成本最重要，因为大模型的成本挑战在于模型规模越来越大，使用的资源越来越多，而模型的运行平台 GPU 由于其稀缺性，价格很昂贵，这就导致每次模型推理的成本越来越高。而最终用户只为价值买单，而不会为推理成本买单，因此降低单位推理的成本是基础设施团队的首要任务。

在此基础上，性能是核心竞争力，特别是 ToC 领域的大模型，更快的推理和推理效果都是增加用户粘性的关键。

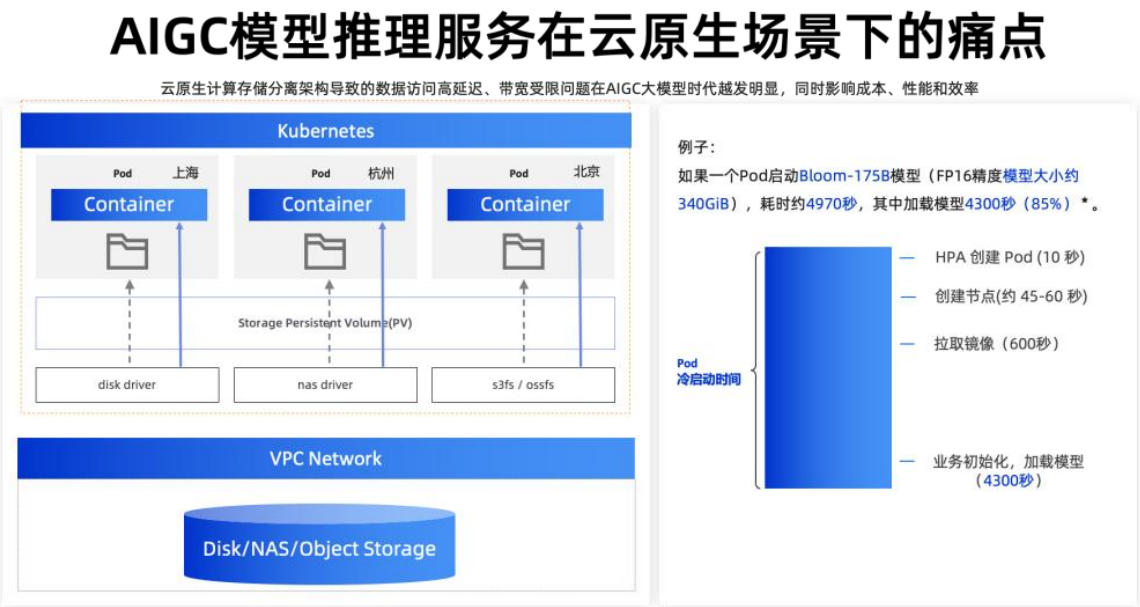
应该说大模型的商业化是一个不确定性较高的领域，成本和性能可以保障你始终在牌桌上。效率是能够保障你能在牌桌上赢牌。

进一步，效率。模型是需要持续更新，这就模型多久可以更新一次，更新一次要花多久的

时间。谁的工程效率越高，谁就有机会迭代出有更有价值的模型。

近年来，容器和 Kubernetes 已经成为越来越多 AI 应用首选的运行环境和平台。一方面，Kubernetes 帮助用户标准化异构资源和运行时环境、简化运维流程；另一方面，AI 这种重度依赖 GPU 的场景可以利用 K8s 的弹性优势节省资源成本。在 AIGC/大模型的这波浪潮下，以 Kubernetes 上运行 AI 应用将变成一种事实标准。

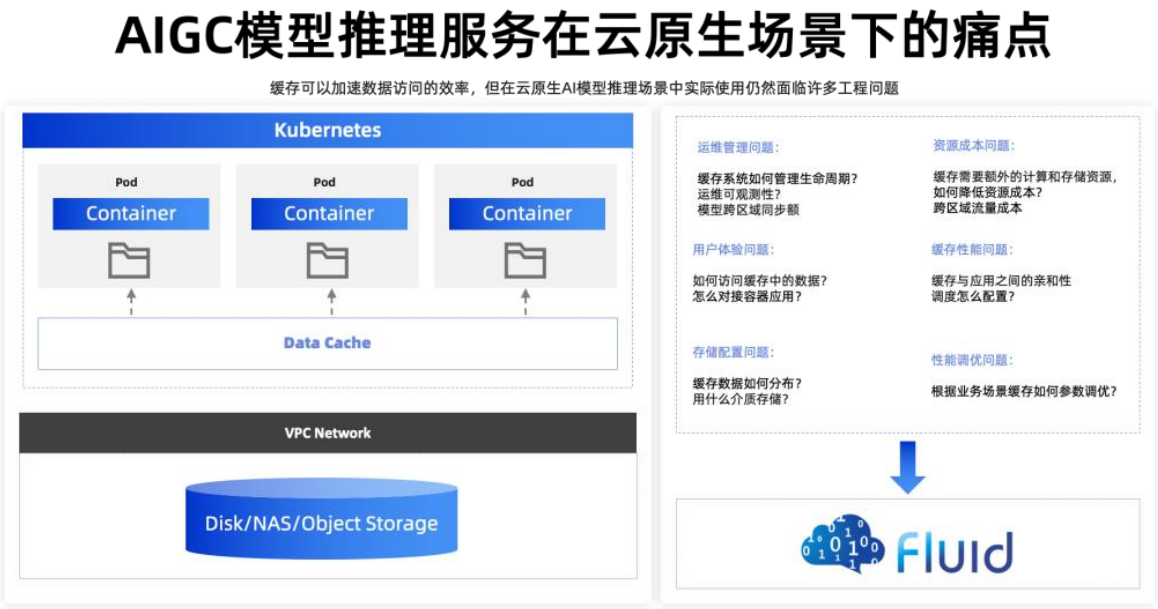
2. AIGC 模型推理服务在云原生场景下的痛点



在 AIGC 推理场景下有个关键的矛盾，就是计算存储分离的架构导致的数据访问高延迟、带宽受限问题和大模型规模不断增长的矛盾，它会同时影响成本、性能和效率。

模型弹性伸缩、按需使用，是控制大模型成本的利器。然而，如上图右所示，以 Bloom-175B 模型（FP16 精度模型大小约 340GiB）为例，模型的扩容耗时为 82 分钟，接近 1 个半小时，为了找到问题的根因，需要对模型启动时间进行拆解，其中主要耗时在于 HPA 弹性、创建计算资源、拉取容器镜像，加载模型。可以看到从对象存储加载一个约 340G 的大模型，耗时大约在 71 分钟，占用整体时间的 85%，这个过程中我们其实可以看到 I/O 吞吐仅有几百 MB 每秒。

要知道在 AWS 上 A100 按量付费的价格每小时 40 美元，而模型启动时刻 GPU 其实是处于空转的时刻，这本身就是成本的浪费。同时这也影响了模型的启动性能和更新频率。



那么，我们有办法解决这个问题吗？一个直观的想法是增加一个缓存层，但是真的增加了缓存层就可以了吗？实践中其实并不是这样的，我们会遇到一系列的问题。

首先就是快的问题：能否用好缓存，如果加了缓存但是速度依旧不快，那么是缓存的规划

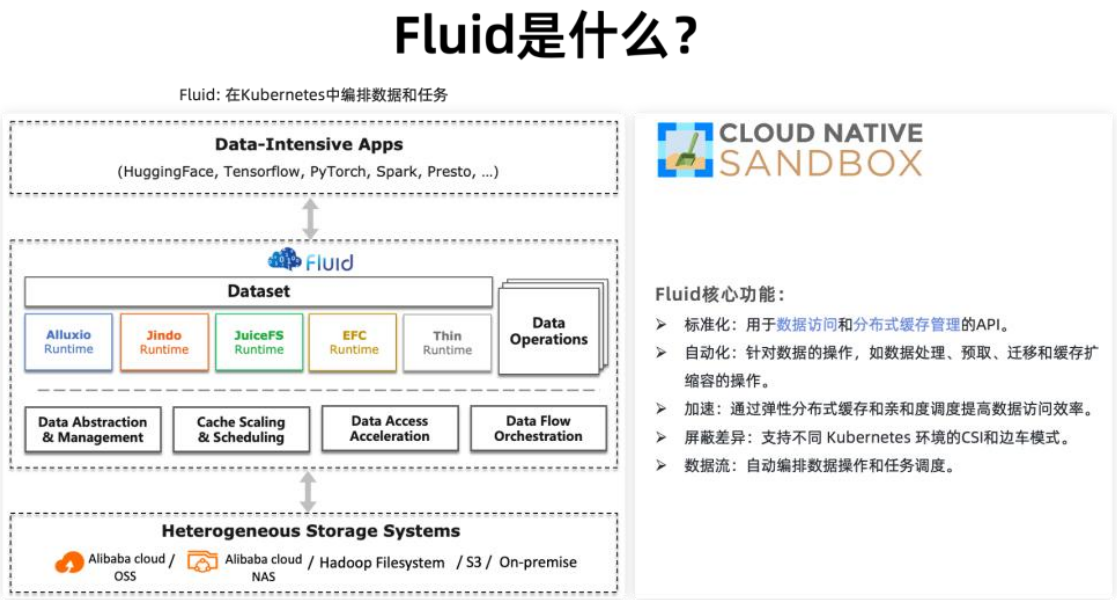
问题？硬件配置问题？还是软件配置？网络问题？调度问题？

其次就是省：关注成本问题，作为缓存的机器通常是高带宽、大内存、本地盘的机器，这些配置的机器往往并不便宜。如何能够实现性能最大化的同时也有合理成本控制。

接着就是好：用户使用复杂不？用户代码是否需要相应的修改。运维团队工作量大吗？模型会不断更新和同步，如何降低这个缓存集群的运维成本。简化运维团队的负担。

正是在这种对于缓存工程化落地的思考中，诞生了 Fluid 这个项目。

3. Fluid 是什么？



首先，让我们来了解一下 Fluid 的概念。Fluid 负责在 Kubernetes 中编排数据和使用数据的计算任务，不仅包括空间上的编排，也包括时间上的编排。空间上的编排意味着计算任务会优先调度到有缓存数据和临近缓存的节点上，这样能够提升数据密集型应用的性能。而时间上的编排则允许同时提交数据操作和任务，但在任务执行之前，要进行一些数据迁移和预热操作，以确保任务在无人值守的情况下顺利运行，提升工程效率。

从 Fluid 的架构图来看，Fluid 向上对接各种 AI/大数据的应用，对下我们可以对接各种异构的存储系统。Fluid 目前支持了包括 Alluxio、JuiceFS 还有阿里内部自研的 JindoFS、EFC 等多种缓存系统。

具体来说 Fluid 提供 5 个核心能力：

1） 首先是数据使用方式和缓存编排的标准化。

一方面，针对场景化的数据访问模式进行标准化，比如大语言模型、自动驾驶的仿真数据、图像识别的小文件，都可以抽象出优化的数据访问方式。

另一方面，越来越多的分布式缓存出现，比如 JuiceFS，Alluxio，JindoFS，EFC 可以加速不同的存储，但是他们并不是为 Kubernetes 而生。如果在 Kubernetes 上使用它们，需要抽象标准的 API；Fluid 负责将分布式缓存系统转换为具有可管理、可弹性，可观测和自我修复能力的缓存服务，并且暴露 Kubernetes API。

2） 其次是自动化，以 CRD 的方式提供数据操作、数据预热、数据迁移、缓存扩容等多种操作，方便用户结合到自动化运维体系中。

3） 加速：通过场景优化的分布式缓存和任务缓存亲和性调度，提升数据处理性能。

4） 随处运行，与 Kubernetes 运行时平台无关：可以支持原生、边缘、Serverless Kubernetes、Kubernetes 多集群等多样化环境。可以根据环境的差异选择 CSI Plugin 和 sidecar 不同模式运行存储的客户端。

5） 数据和任务编排：最终连点成线，支持定义以数据集为中心自动化操作流程，定义数据迁移、预热、任务的先后执行顺序依赖。

4. Fluid 在云原生 AIGC 模型推理场景的优化概述

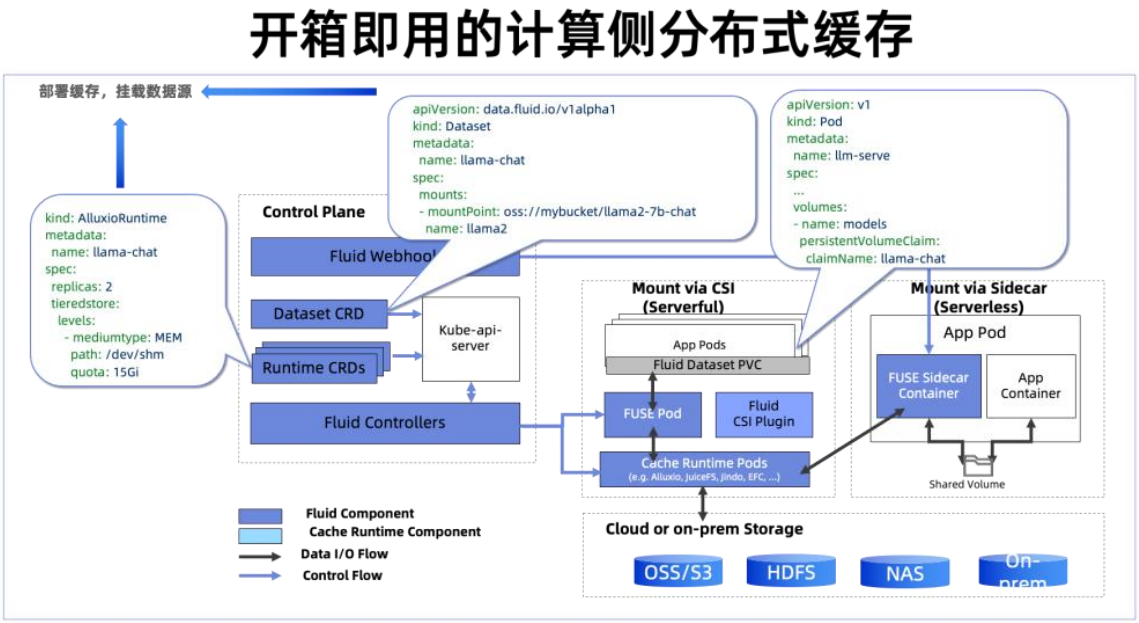
那么回到 AIGC 模型推理场景，Fluid 为这个场景带来了许多优化方案。

首先，分布式缓存使用的复杂度高和运行环境差异大，AIGC 应用需要适配不同运行时，包括 Alluxio，JuiceFS，JindoFS，而运行时环境包括公共云、私有云、边缘云、Serverless 云，Fluid 都可以提供一键部署、无缝衔接的能力。

第二，AIGC 模型推理服务本身有很多灵活多变的业务属性，通过 Fluid 提供的弹性缓存帮您实现需要的时候可以弹出来不用的时候缩回去，能很好地在性能和成本间取得利益最大化。

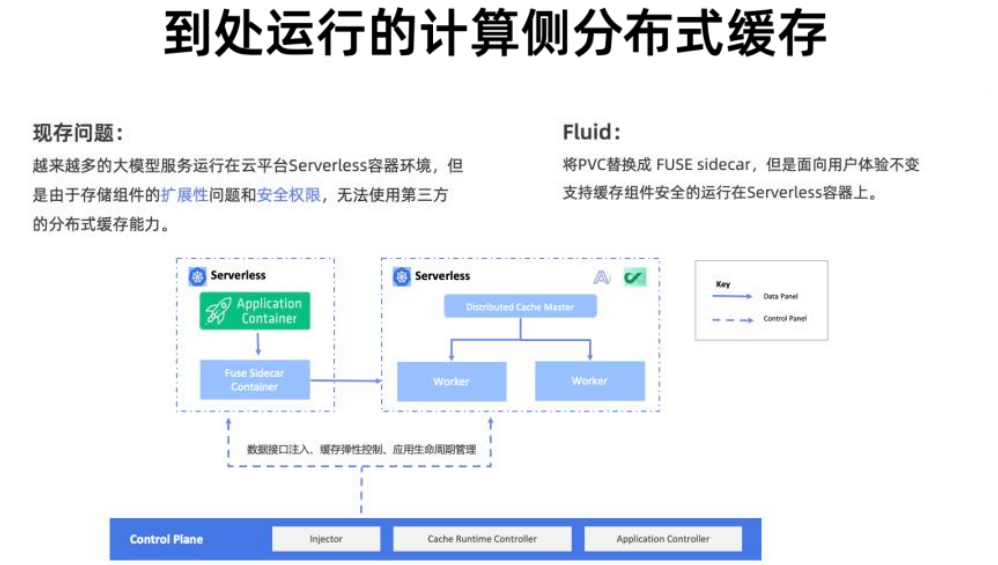
第三，Fluid 提供数据感知调度能力，将计算尽量调度到离数据更近的地方。

第四，Fluid 的数据流编排能力，帮助用户把很多推理的行为和数据的消费行为自动化起来，减少复杂度。最后，在性能上，我们也提供了适合云原生缓存的读取优化方案，充分利用节点资源。



这边是一张 Fluid 的技术架构图。图中可以看到，Fluid 提供 Dataset 和 Runtime 这两种 CRD，它们分别代表了需要访问的数据源和对应的缓存系统。比如在这个例子里面我们使用的是 Alluxio 这个缓存系统，所以对应的就是 AlluxioRuntime 的 CRD。

Dataset 中描述了你需要访问的模型的数据路径，比如 OSS 存储桶中的一个子目录。创建了 Dataset 和对应的 Runtime 后，Fluid 会自动完成缓存的配置、缓存组件的拉起，并自动创建一个 PVC。而对于想要访问这个模型数据的推理应用来说，只需要挂载这个 PVC，就可以从缓存中读取模型数据，这和 K8s 标准的存储方式也是保持一致的。



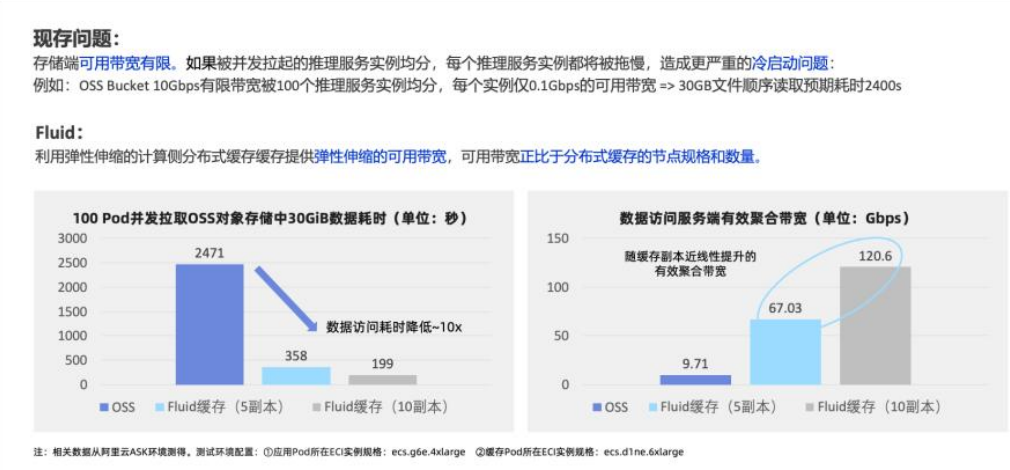
AIGC 推理的运行平台非常多样化，包括云服务的 Kubernetes、自建的 Kubernetes、边缘的 Kubernetes 以及 Serverless 形态的 Kubernetes。Serverless 形态的 Kubernetes 由于其易用性、低负担的好处，已经越来越多的成为用户的选择；但是 Serverless 由于安全的考量，没有开放第三方存储接口，所以只支持自身存储，以阿里云为例子，只有 NAS，OSS，CPFS 有限存储。

在 Serverless 容器平台上，Fluid 会将 PVC 自动转换成可以适配底层平台的 sidecar，开放了第三方的配置接口，可以允许并且控制这个 sidecar 容器的生命周期，保证它在应用容器启动前运行，当应用容器结束后自动退出。这样 Fluid 则提供了丰富的可扩展性，可以运行多种分布式缓存引擎。

AIGC 模型需要共享还是独占，这不是一个“一刀切”的问题，需要结合真实的业务场景进行选择。有些 IP 保护，核心模型是需要访问隔离，而另一些开源模型则没有这部分的担心。有些模型性能高度敏感，特别一些最流行常用文生图的场景，需要在 20 秒内完成出图，这样 8-10G 的模型加载时间要控制到 5 秒以内。

那就需要在缓存侧做吞吐独享、避免竞争、配合特定调优。而对于一些比较新的文生图，用户就需要考虑资源成本。而 Fluid 针对于独占缓存和共享缓存，都提供了完整的支持，通过 Fluid 都可以灵活配置支持。

充分利用带宽的计算侧分布式缓存



Fluid 提供的第二个优化是可弹性伸缩的计算侧分布式缓存。

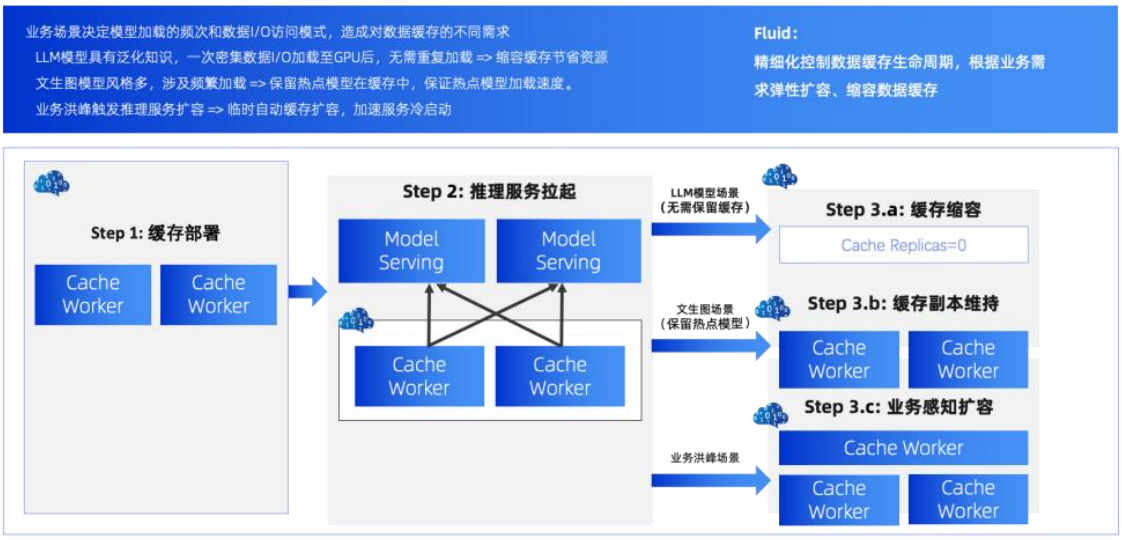
这里讲的是如何提供高性能。为什么需要弹性伸缩的计算侧分布式缓存？只是使用简单的分布式缓存不够吗？我们可以从技术角度来理解这个问题。

在实际的生产场景中，AI 模型推理服务实例往往是多个并发启动的，例如：如果你一次性需要拉起 100 个推理服务实例，每个实例都需要从对象存储中拉取数据，那么每个实例能分到的可用带宽仅有总共可用带宽的百分之一。如果是默认 10Gbps 的 OSSBucket 加载 30G 的模型，这个预期耗时就会是 2400s，而且是每个实例都是 2400s。

事实上，弹性伸缩的计算侧分布式缓存就是把底层存储系统的有限可用带宽转变为了 K8s 集群内可以弹性伸缩的可用带宽，这个可用带宽的大小取决于你分布式缓存的节点数量。从这个角度来说，我们就能根据实际业务场景对于 I/O 的变化需求，变为可随时扩容缩容的分布式缓存集群。

这里有一些测试数据我们也可以看到，如果 100 个 Pod 并发启动，使用缓存都能获得很好的加速效果，而使用更多的缓存 Worker 节点，效果会更好。主要的原因就来自于更大的聚合带宽，使得每个 Pod 均分得到的带宽更多。从右边这张图也可以看到，当你使用更多的分布式缓存节点的时候，聚合带宽也是近线性地提升的。

可弹性伸缩的计算侧分布式缓存



介绍完如何提升性能之后，接下来考虑的问题就是如何在尽可能节省成本的前提下最大化缓存带来的性能提升，如何在成本和性能间取得平衡实质上是与业务场景的 I/O 访问模式相关的。Fluid 在缓存上暴露的可观测性，配合手动扩缩容、HPA、CronHPA 等 K8s 的扩缩容能力，可以根据业务需求弹性扩容、缩容数据缓存。

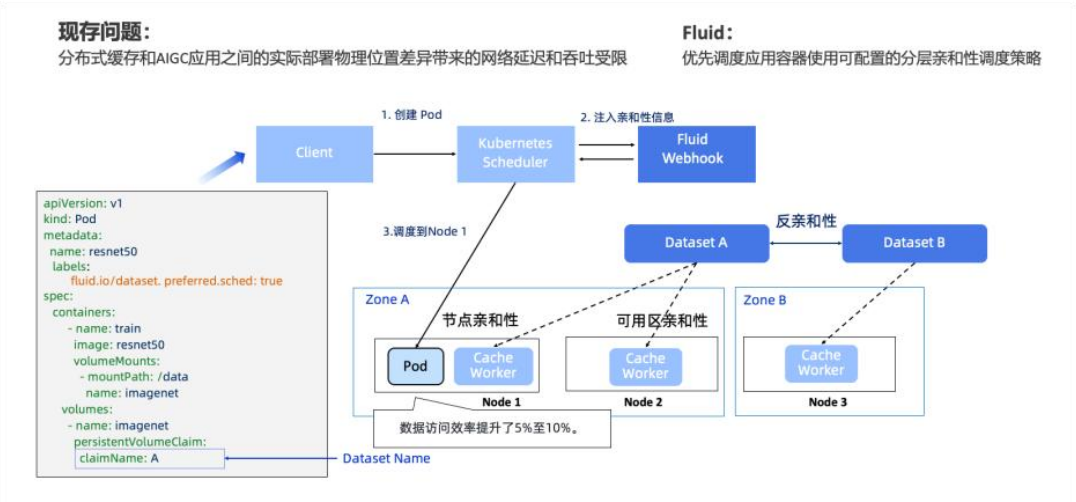
我们可以举几个具体的例子：

对于大语言模型场景，大语言模型一个特点是拥有很强的泛化知识，因此把一个 LLM 加载到 GPU 显存后，它其实可以为多种不同的场景提供服务。因此这种业务的数据 I/O 特点是一次性对 I/O 有很高的要求。对应到缓存的弹性上来，就是一个先扩容，推理服务就绪后缩容到 0 的过程。

再来看文生图 Stable Diffusion 的场景，假如是一种 SD 模型市场的场景，那就会包含大量不同风格的 SD 模型，因此尤其对于热点模型，会有持续的 I/O 需求，此时保持一定的缓存副本就是更好的选择。

而无论哪种场景，如果因为业务洪峰造成服务端需要扩容，数据缓存可以跟随它做临时扩容，缓解扩容时的冷启动问题。

数据亲和性调度



公共云提供灵活的弹性能力和高可用性，这是通过底层的多可用区实现的，多可用区对于互联网应用非常合适；它牺牲一点点的性能获得了应用稳定性。

但是在 AIGC 大模型场景上，通过实际验证，我们发现跨可用区的延时还是有很大的影响，这是因为大模型文件一般比较大，它传的包就会非常多，对延时起到放大的作用。因此缓存和使用缓存应用之间的亲和性就非常重要，Fluid 提供无侵入性的亲和性调度，根据缓存的地理位置调度应用，优先同可用区调度；同时提供了弱亲和性和强亲和性的可配置性，帮助用户灵活使用。

数据操作&数据流编排

现在我们理解了弹性的缓存架构的必要性和优势，但实际用起来也许还是会有一些麻烦。

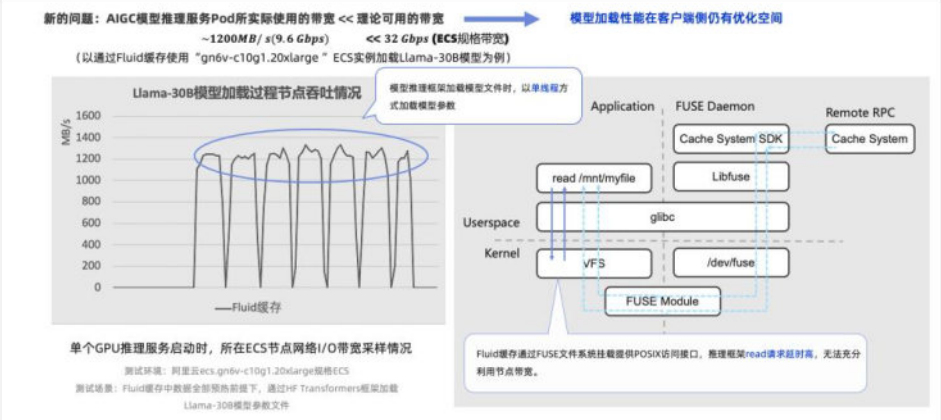
让我们试想这么一个流程，今天有个新的 AI 模型推理业务需要发布上线，为了避免服务冷启动，你先需要部署了一个分布式缓存并扩容到了一定的副本数，接下来你把待发布的模型数据预热到分布式缓存中避免 Cache Miss（这个过程可能要花 30min），最后你拉起 100 个服务实例，等到 100 个服务实例启动完成，这个过程又要花费 10～20 分钟；最后，确认服务上线没问题后，把缓存缩容掉减少成本。

这个过程中的每一步每隔一段时间就需要人工参与，确认状态并执行下一步。数据访问和消费过程运维过程复杂，耗时费力。

Fluid 为了解决这个问题，我们把数据消费过程定义为业务使用数据缓存的过程，以及系统准备数据缓存的过程，对于这些流程我们用数据操作抽象以及数据流编排能力去帮助用户自动化。比如最常见的与数据缓存相关的操作，像是数据迁移、预热以及和业务相关的数据处理，Fluid 都提供了 K8s 级别的抽象去描述。

这些数据操作可以串联成一条数据流。于是刚才我们提到的这个例子，就可以用 5 步数据操作来轻松定义。运维人员只需要一次性提交这条数据流，Fluid 自动地会完成整个 AI 模型推理服务发布的流程，提升使用缓存过程的自动化比例。

数据访问客户端性能优化

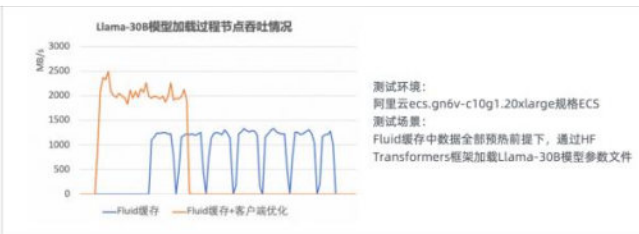


那么刚才提到的“用好缓存”的技巧其实都在资源成本和运维效率方面。但实际测试过程中我们发现，服务启动过程使用的带宽远小于这些 GPU 计算实例可用的带宽，这意味着模型的加载效率在客户端上仍然有可以优化的空间。

从节点吞吐情况上可以看到，这些 AI 推理的运行时框架会以单线程的方式去按序读取模型参数，这在非容器环境是没有什么问题的，如果使用本地 SSD 盘存储模型参数，加载吞吐很容易就可以到达 3～4GB/s。但是在计算存储分离架构下，哪怕我们使用了缓存，缓存也需要使用用户态文件系统（也就是 FUSE）这种技术挂载到容器中。FUSE 自身的开销和额外的 RPC 调用，都使得 read 请求的延时变得更高，单线程所能达到的带宽上限也就更低了。

数据访问客户端性能优化

Fluid SDK：使用多线程读和预读等手段优化客户端模型加载过程，匹配Fluid缓存架构



为了最大化发挥分布式缓存提供的巨大 I/O 吞吐，Fluid 可以提供一个 Python 的 SDK，在用户代码中使用多线程读和预读的方式去加速模型加载过程。

从我们的测试结果来看，额外使用这种客户端的优化，可以在使用计算侧分布式缓存的基础上，将冷启动耗时缩短一半，做到 1 分钟内拉起一个接近 100G 的大模型。从右下角的这个 I/O 吞吐情况，也可以看出，我们更充分地利用了 GPU 计算节点的带宽资源。

为了评估 Fluid 的性能，我们采用了 HuggingFace Text-Generation-Inference 框架来构建大型语言模型(LLM)的推理服务。我们将模型存储在 OSS 对象存储中，并对用户体验以及直接从 OSS 对象存储拉取与通过 Fluid 拉取数据启动推理服务的性能差异进行了对比分析。

我们首先看一下直接访问 OSS 存储的运行效果。

这里我们已经创建好了 OSS 的 PV 和 PVC。接着，我们定义一个 deployment：deployment Pod 中挂载刚才的 OSS PVC，使用的容器镜像是 TGI 镜像。还有声明使用

1 张 GPU 卡，用于模型推理。接着把 deployement 创建下去。然后我们看下这个服务的就绪时间，这边 5 倍速加速了一下。终于就绪了，可以看到整个过程耗费了 101s，考虑到我们的模型大小仅为 12.55G，这个时间可以说是比较长的。

最后，让我们看看 Fluid 的优化效果。我们需要定义 Fluid 的 Dataset 和 Runtime 资源，并将分布式缓存部署到集群中。定义数据源、节点个数以及缓存数据的存储介质和大小。由于我们是初次部署弹性分布式缓存，这可能需要约 40 秒的时间。

缓存准备完成后，我们可以看到一些缓存的监控信息。PVC 和 PV 也会自动创建。然后，我们定义一个新的 deployment，只需要进行几个修改：

- 添加一个 annotation 来触发 Fluid 的自动数据预热
- 将 OSS 的 PVC 更改为由 Fluid Dataset 自动创建的 PVC
- 替换为一个使用了客户端优化的镜像

观察服务的就绪时间，我们可以看到部署只花了 22 秒。我们还可以尝试对现有的 deployment 进行扩容，观察第二个服务实例的启动时间。由于所需的模型数据已被完全缓存，第二个服务实例只需 10 秒就能准备就绪。这个例子展示了 Fluid 优化的效果，我们成功提升了服务的启动速度约 10 倍。

4. 总结

Fluid 为 AIGC 模型弹性加速提供开箱即用、优化内置的方案，在达到更好性能的同时还可以降低成本，同时还包含端到端的自动化能力；在此基础上使用 Fluid SDK 可以进一步充分发挥 GPU 实例的带宽能力实现极致的加速效果。

第四章

容器前沿技术与大模型生产实践

阿里云 ACK 云上大规模 Kubernetes 集群高可靠性保障实战

作者：刘佳旭，阿里云技术专家

1. 引言

2023 年 7 月，阿里云容器服务 ACK 成为首批通过中国信通院“云服务稳定运行能力-容器集群稳定性”评估的产品， 并荣获“先进级”认证。随着 ACK 在生产环境中的采用率越来越高，稳定性保障已成为基本诉求。本文基于 ACK 稳定性保障实践经验，帮助用户全面理解 ACK 稳定性理论和优化策略，并了解如何使用相应的工具和服务进行稳定性保障。

2. K8s 集群稳定性和大规模场景下的挑战

1）K8s 常见的稳定性痛点

Kubernetes 在提供丰富的技术和功能外，架构和运维具有较高的复杂性，也产生了诸多的痛点。

痛点 1：在发布、弹性等高峰期，集群控制面服务时断时续，甚至完全不可用

面对大流量请求，如果控制面没有自动弹性扩容能力，会无法对负载自适应、导致控制面服务不可用。例如：客户端存在高频度持续 LIST 集群中的大量资源，集群 apiserver/etcd 无法自动弹性就可能联动出现 OOM。ACK Pro 托管版 K8s 可以对控制面组件根据负载压力做 HPA 和 VPA，可以有效解决该痛点。

痛点 2：集群节点批量 NotReady 导致雪崩，严重影响业务!

部分节点出现 NotReady，节点上 Pod 被驱逐调度到健康节点，健康节点由于压力过大也变为 NotReady，加剧产生了更多 NotReady 的节点，业务持续重启。ACK 提供了托管节点池功能，可以对出现 NotReady 的异常节点治愈，重新拉会 Ready 状态，可以有效解决该痛点。

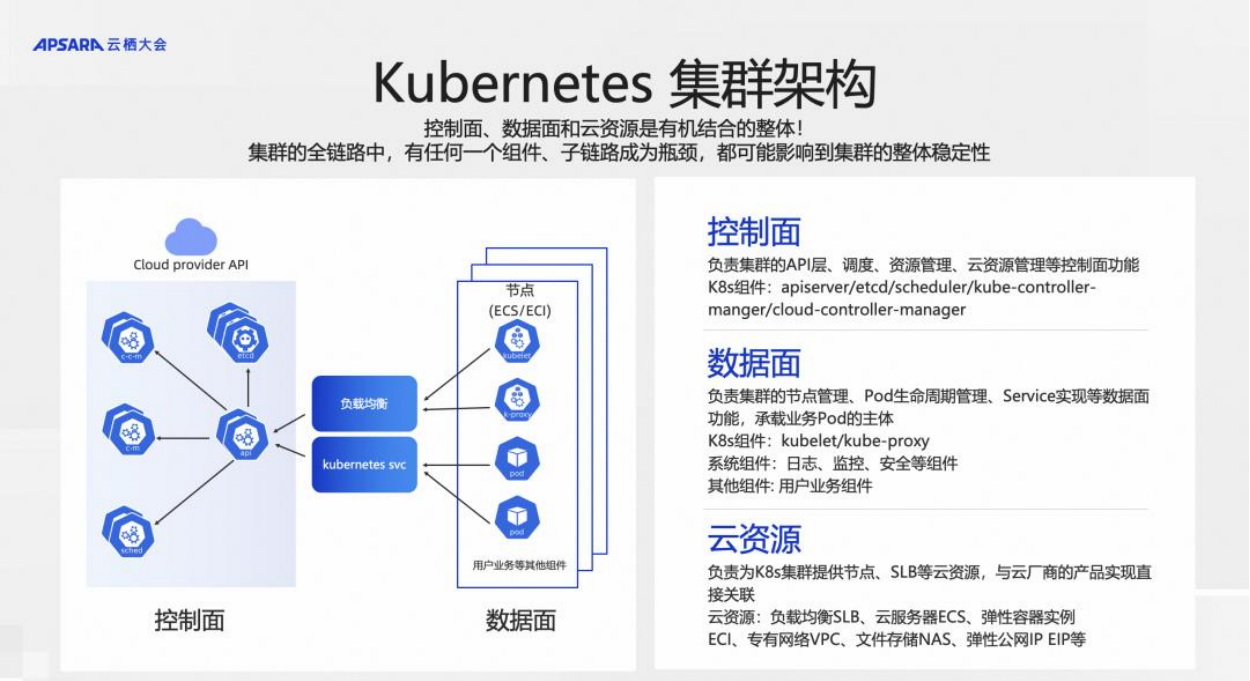
痛点 3：业务高峰期需快速弹性，节点上拉取 Pod 镜像耗时长达分钟级，影响业务

节点上 kubelet 并发拉取镜像遇到网络带宽限制，需要镜像加速功能支持。ACR 提供了基于 DADI（Data Accelerator for Disaggregated Infrastructure）的按需镜像加载和 P2P 镜像加速的功能，可以加速镜像拉取，可以有效解决该痛点。

痛点 4：Master 节点/组件运维复杂度高，包含资源配置、参数调优、升级管理等

需要大量的线上场景分析和优化、故障处理、规模压测等，来分析、整理并落地最佳实践和配置。ACK Pro 托管版 K8s 在全网的规模体量上万集群，具有自动弹性和生命周期管理的运维管理架构，有丰富的优化、应急处理等经验，持续将最佳实践和参数优化对托管组件升级。

2）Kubernetes 集群架构



既然有这些痛点，我们从 K8s 架构的角度来分解一下，看看哪些部分可能出现故障和问题：云上 K8s 集群包含控制面、数据面、以及承载控制面和数据面的的云资源。控制面和数据面通过 SLB 和云网络连接。控制面负责集群的 API 层、调度、资源管理、云资源管理等控制面功能，K8s 组件：

apiserver/etcd/scheduler/kube-controller-manger/cloud-controller-manager。

数据面负责集群的节点管理、Pod 生命周期管理、Service 实现等数据面功能，承载业务 Pod 的主体。包含：K8s 组件，kubelet/kube-proxy；系统组件，日志、监控、安全等组

件；其他组件，用户业务组件。

控制面、数据面和云资源是有机结合的整体！集群的全链路中，任何一个组件、子链路成为瓶颈，都可能影响到集群的整体稳定性。我们需要从 K8s 系统中发现瓶颈、治理以及优化瓶颈，最终实现 K8s 系统在给定云资源情况下的稳定、高效的利用。

3） Kubernetes 稳定性体现

我们已经了解了 K8s 集群架构，那么如何评估 K8s 集群的稳定性呢？集群稳定性涵盖 Kubernetes 服务可用性、处理时延、请求 QPS 和流量吞吐、资源水位等要素。

Kubernetes 稳定性风险和挑战

结合刚才介绍的 K8s 的架构和稳定性体现，我们来看看 K8s 集群的稳定性风险和挑战，在大规模场景下稳定性风险和挑战会更加突出。

挑战 1：集群内资源种类繁多，数量巨大

大规模集群场景下常见。包含原生 K8s 资源和丰富灵活的 CRD 资源。节点是 K8s 的一种资源，节点规模大的集群是大规模集群的一种；从 K8s 治理的角度，集群中某种资源数量巨大，例如 configmap、secrets 等，即便节点数不大，也可以称为大规模集群。

例如：单集群超过 1 万节点规模、单集群有 10W+ 的 namespace 以及 ns 下 secret/configmap 资源。

挑战 2：控制面压力的风险

控制面组件缓存集群的部分或者全部资源。在大规模场景下，每个组件都会有明显的资源压力。超过资源 Limits 就会触发 OOM 等问题。例如 apiserver 将 etcd 中全部资源在内存中缓存以便响应客户端对 Cache 的 LIST 请求。

请求来源复杂。包括随节点规模正增长的 kubelet/kube-proxy/daemonset，也包括系

统组件和用户部署的组件。

挑战 3：数据面压力、以及数据面与控制面同步压力的风险

数据面节点出现压力以及异常。节点负载压力过高，导致 kubelet/运行时响应慢或者无响应，甚至节点状态 NotReady。数据面与控制面同步瓶颈。

数据面与控制面网络带宽打满或者网络不通，kubelet 无法及时更新 node 状态，导致节点状态 NotReady，导致容器调度、service 后端流量转发受影响。

挑战 4：云资源稳定性和高可用稳定性

有限的云资源容量。例如 SLB 的连接数、带宽，ECS 的内存、CPU 等等，存在打满的风险。

集群的核心云资源和组件需要按高可用架构部署。包括跨节点、AZ 等不同高可用等级。

3. ACK 稳定性治理和优化策略

1）ACK K8s 稳定性概述

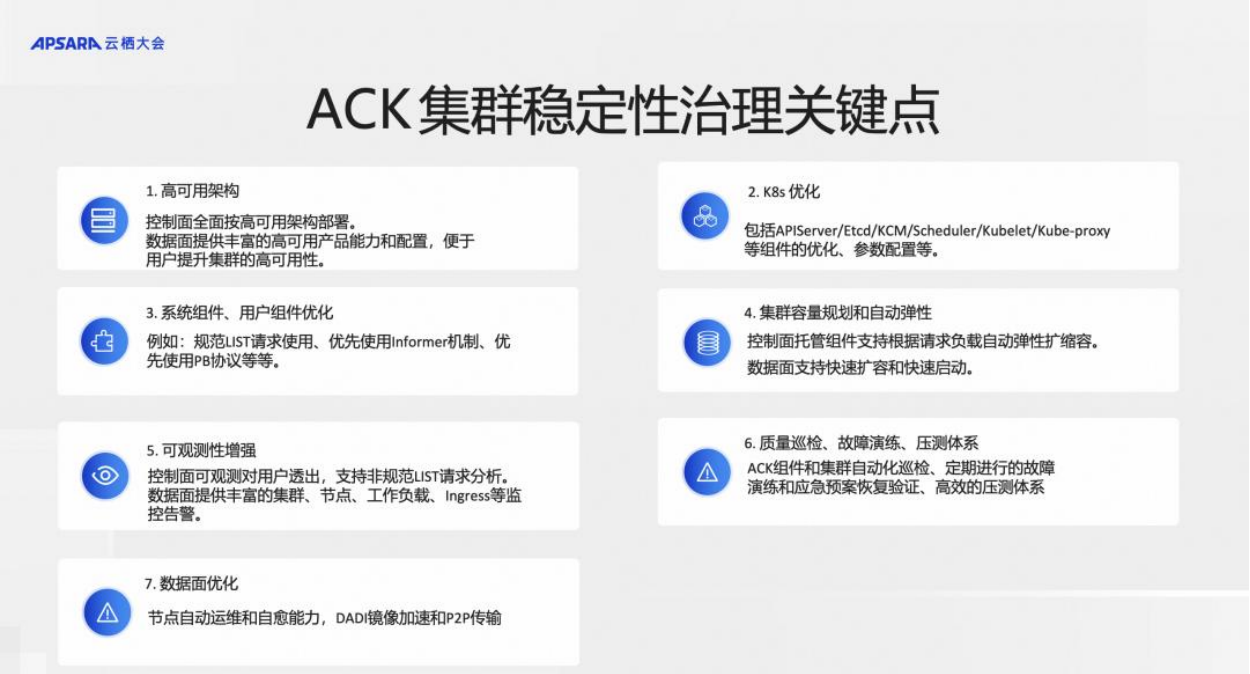
2023 年 7 月，ACK 成为首批通过中国信通院“云服务稳定运行能力-容器集群稳定性”评估的产品，并荣获“先进级”认证。

ACK K8s 稳定性优化，源于大规模实践经验沉淀，具体包括：ACK 全网管理了数万个 K8s 集群，对线上丰富的客户和业务场景提供全面的支持；ACK 作为底座承载了双十一、618 等超大规模的电商业务，经受住了阿里巴巴电商场景的极限压力的考验；对社区原生 K8s 做参数、性能、架构等优化，并形成产品能力。

ACK 针对丰富的业务类型和大规模场景进行优化，例如：

- 云上的大规模化场景，支持单集群上万节点
- Sark/Flink 等大数据场景
- Tensforflow/Pytorch 等 AI 场景
- ECI/Spot 等快速弹性场景
- ArgoWorkflow 等任务流场景

2）ACK 集群稳定性治理关键点



a. 高可用架构

控制面全面按高可用架构部署。

数据面提供丰富的高可用产品能力和配置，便于用户提升集群的高可用性。

b. K8s 优化

包括 APIServer/Etcd/KCM/Scheduler/Kubelet/Kube-proxy 等组件的优化、参数配置等。

c. 集群容量规划和自动弹性

例如：规范 LIST 请求使用、优先使用 Informer 机制、优先使用 PB 协议等等。

d. 系统组件、用户组件优化

控制面托管组件支持根据请求负载自动弹性扩缩容，控制面可观测对用户透出。
数据面提供丰富的集群、节点、工作负载、Ingress 等监控告警。

e. 质量巡检、故障演练、压测体系

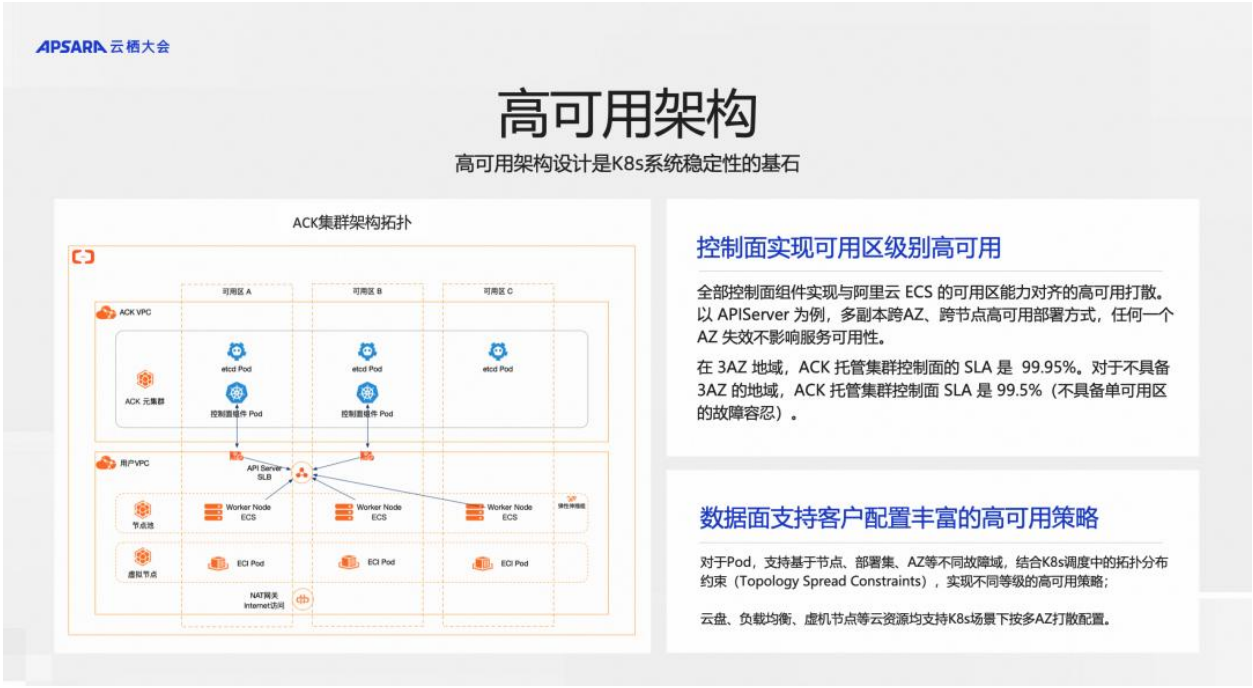
ACK 组件和集群自动化巡检、定期进行的故障演练和应急预案恢复验证、高效的压测体系。

f. 数据面优化

节点自动运维和自愈能力，镜像加速和 P2P 传输。

下面针对部分优化关键点详细展开说明。

3）高可用架构



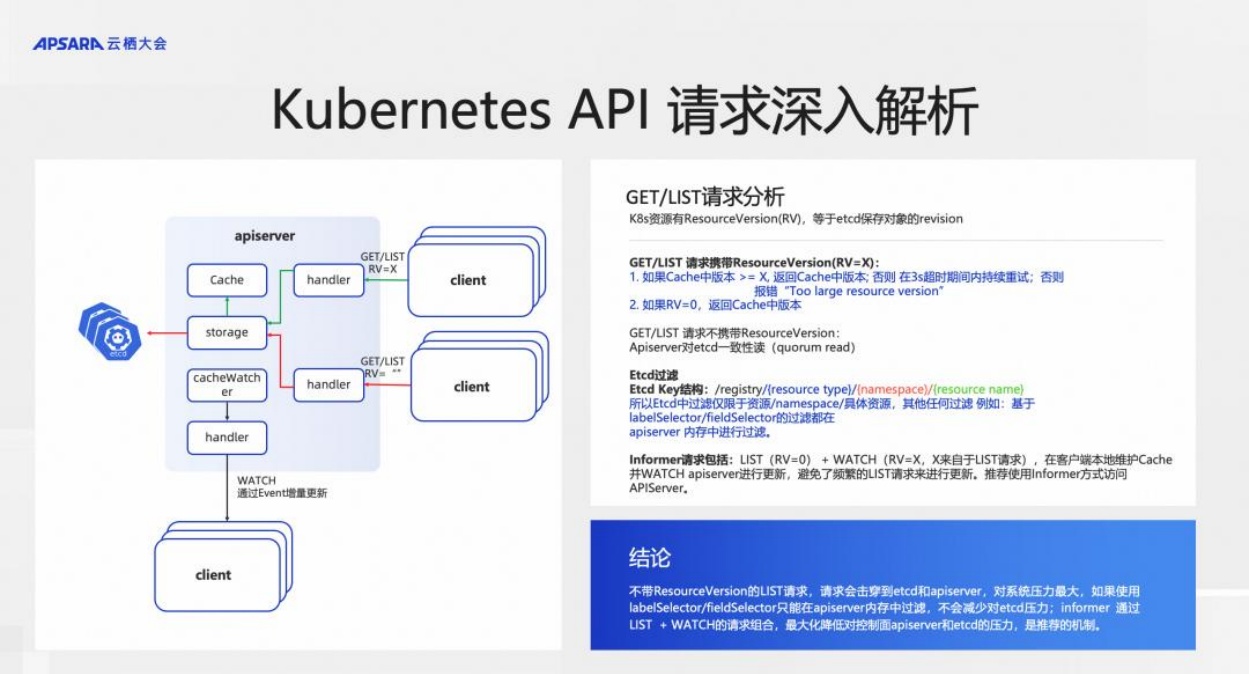
控制面实现可用区级别高可用 全部控制面组件实现与阿里云 ECS 的可用区能力对齐的高可用打散。

以 APIServer 为例，多副本跨 AZ、跨节点高可用部署方式，任何一个 AZ 失效不影响服务可用性。在 3AZ 地域，ACK 托管集群控制面的 SLA 是 99.95%。对于不具备 3AZ 的地域，ACK 托管集群控制面 SLA 是 99.5%（不具备单可用区的故障容忍）。

控制面实现可用区级别高可用全部控制面组件实现与阿里云 ECS 的可用区能力对齐的高可用打散。以 APIServer 为例，多副本跨 AZ、跨节点高可用部署方式，任何一个 AZ 失效不影响服务可用性。在 3AZ 地域，ACK 托管集群控制面的 SLA 是 99.95%。对于不具备 3AZ 的地域，ACK 托管集群控制面 SLA 是 99.5%（不具备单可用区的故障容忍）。

数据面支持客户配置丰富的高可用策略。

对于 Pod，支持基于节点、部署集、AZ 等不同故障域，结合 K8s 调度中的拓扑分布约束（Topology Spread Constraints），实现不同等级的高可用策略；云盘、负载均衡、虚机节点等云资源均支持 K8s 场景下按多 AZ 打散配置。



在分析 APIServer 优化前，先来看一下 K8s API 请求的分析。

这里的结论为：不带 ResourceVersion 的 LIST 请求，请求会击穿到 etcd 和 apiserver，对系统压力最大，如果使用 labelSelector/fieldSelector 只能在 apiserver 内存中过滤，不会减少对 etcd 压力；informer 通过 LIST + WATCH 的请求组合，最大化降低对控制面 apiserver 和 etcd 的压力，是推荐的机制。

4）APIServer 稳定性优化

a. APIServer 自动弹性

ACK 管控基于访问压力和集群容量实现 APIServer 实例自动弹性。

b. 软负载均衡

方法：负载不均会导致个别 APIServer 实例资源开销大、容易触发 OOM。Goaway 特性概率性断开并新建 TCP 连接， 实现负载均衡的效果。作用：帮助稳定运行的集群能解决负载不均衡问题。

c. 托管组件可观测性透出

全部托管组件 apiserver、etcd 等监控告警对用户透出。支持识别可能存在的非规范 LIST 请求的 Grafana 看板，帮助用户评估组件行为。

d. 集群资源清理 关闭不需要功能

及时清理不使用的 Kubernetes 资源，例如 configmap、secret、pvc 等；及时清理不使用的 Kubernetes 资源，例如 configmap、secret、pvc 等。

5）Etcd 稳定性优化

a. Data 和 Event etcd 分拆

Data 和 Event 存放到不同的 etcd 集群。数据和事件流量分离，消除事件流量对数据流量的影响；降低了单个 etcd 集群中存储的数据总量，提高了扩展性。

b. Etcd 根据资源画像 VPA

根据 Etcd 资源使用量，动态调整 etcd Pod request/limits，减少 OOM。

c. AutoDefrag

operator 监控 etcd 集群 db 使用情况，自动触发 defrag 清理 db，降低 db 大小，提升查询速度。

6）Scheduler/KCM/CCM 稳定性优化

QPS/Burst 参数调优。KCM/Scheduler/CCM 的 QPS/Burst 参数在规模化场景下需要提升，避免核心组件出现客户端限流；同时观测 APIServer 监控，避免 APIServer 对核心组件限流。

7）组件稳定性优化

a. 规范组件 LIST 请求

必须使用全量 LIST 时添加 resourceVersion=0，从 APIServer cache 读取数据，避免一次请求访问全量击穿到 etcd；从 etcd 读取大量数据，需要基于 limit 使用分页访问。加快访问速度，降低对控制面压力。

b. 序列化编码方式统一

对非 CRD 资源的 API 序列化协议不使用 JSON，统一使用 Protobuf，相比于 JSON 更节省传输流量。

c. 优选使用 Informer 机制

大规模场景下，频繁 LIST 大量资源会对管控面 APIServer 和 etcd 产生显著压力。频繁 LIST 的组件需要切换使用 Informer 机制。基于 Informer 的 LIST+WATCH 机制优雅的访问控制面，提升访问速度，降低对控制面压力。

d. 客户端访问资源频度

客户端控制访问大规模全量资源的频度，降低对管控的资源和带宽压力。

e. 对 APIServer 访问的中继方案

大规模场景下，对于 Daemonset、ECI pod 等对 APIServer 进行访问的场景，可以设计可横向扩容的中继组件，由中继组件统一访问 APIServer，其他组件从中继组件获取数据。例如 ACK 的系统组件 poseidon 在 ECI 场景下作为 networkpolicy 中继使用。降低管控的资源和带宽压力，提升稳定性。

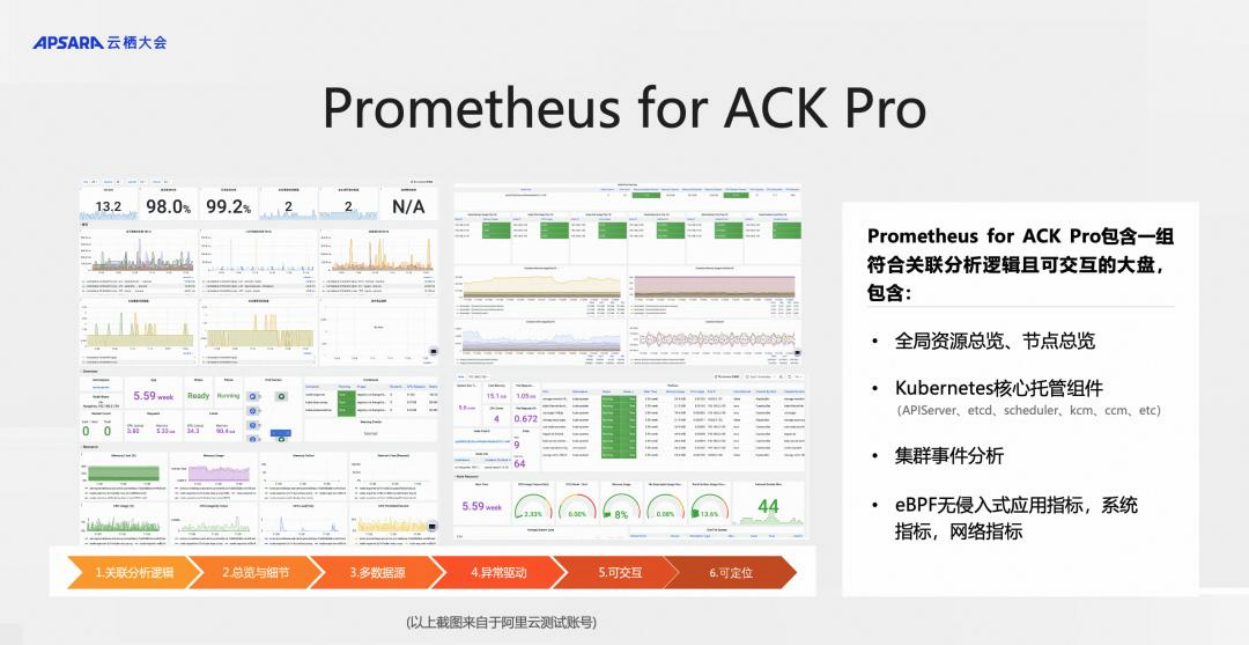
4. ACK 稳定性产品功能和最佳实践器

针对刚才提到的 K8s 稳定性风险和挑战，我们看一下 ACK 是如何进行稳定性治理和优化的。ACK 提供了高效丰富的稳定性产品功能，这里着重从可观测性、故障预防与定位、自动化节点运维角度来介绍产品功能，对应的产品功能分别是：

- Prometheus for ACK Pro
- 容器 AIOps 套件
- 托管节点池

帮助客户提升透明可观测、风险可预测、故障可定位、运维自动化的稳定性保障。

1）Prometheus for ACK Pro



在透明可观测方面，ACK 支持从应用层、APM 层、K8s 层到基础设施层的全景可观测。PrometheusforACKPro 结合容器服务最佳实践经验，提供了可以关联分析、可交互的大盘。

例如：

- 全局资源总览、节点总览
- K8s 核心托管组件的监控，例如 apiserver，etcd 等等
- 集群事件分析
- 在节点层结合 eBPF 实现了无侵入式应用监测

基于 ACKPrometheusforACKPro，可以全面覆盖数据面和控制面的可观测性。

2）容器 AIOps 套件-故障预防与定位

在智能运维方面，ACK 的容器 AIOps 套件凭借 10 年大规模容器运维经验沉淀，自动化诊断能力能够覆盖 90% 的运维问题。基于专家系统+大模型，AIOps 套件提供全栈巡检、集群检查、智能诊断三大功能。

- 全栈巡检，定位集群风险巡检。可以扫描集群健康度和潜在风险，例如云资源配额余量、K8s 集群关键资源水位，并提供修复的解决方案。
- 集群检查，定位运维操作前的检查。例如企业在业务升级过程中经常遇到的 K8s 版本较老，基于各种顾虑不敢升级的问题，阿里云 ACK 可以自动识别出应用是否在使用 K8s 老版本废弃的 API、集群资源是否足够，帮助企业规避升级过程中遇到的风险。
- 智能诊断，定位诊断 K8s 问题。可以诊断异常的 Pod，Node，Ingress，Service，网络和内存。

3）托管节点池

在节点自动化运维方面，托管节点池是 ACK 面向数据面提供的产品功能。定位是让用户专注上层应用部署，ACK 负责节点池基础运维管理。

支持自升级、自愈、安全修复、极速弹性四大功能。

- 自升级是指自动升级 kubelet 和节点组件。
- 自愈是指自动修复运行时和内核问题。例如发现 NotReady 的节点，并治愈恢复为 Ready 状态。
- 安全修复是指支持 CVE 修复和内核加固。
- 极速弹性是基于 ContainerOS 以及通用 OS 的快速弹性。P90 统计算法下，完成 1000 节点扩容只需要 55s。

5. 展望

ACK 稳定性保障建设会持续演进，会继续为客户提供安全、稳定、性能、成本持续优化的产品和稳定性保障！

基于阿里云 ACK 与 ACR 构建企业级端到端 DevSecOps 流程

作者：匡大虎，阿里云高级技术专家

引言

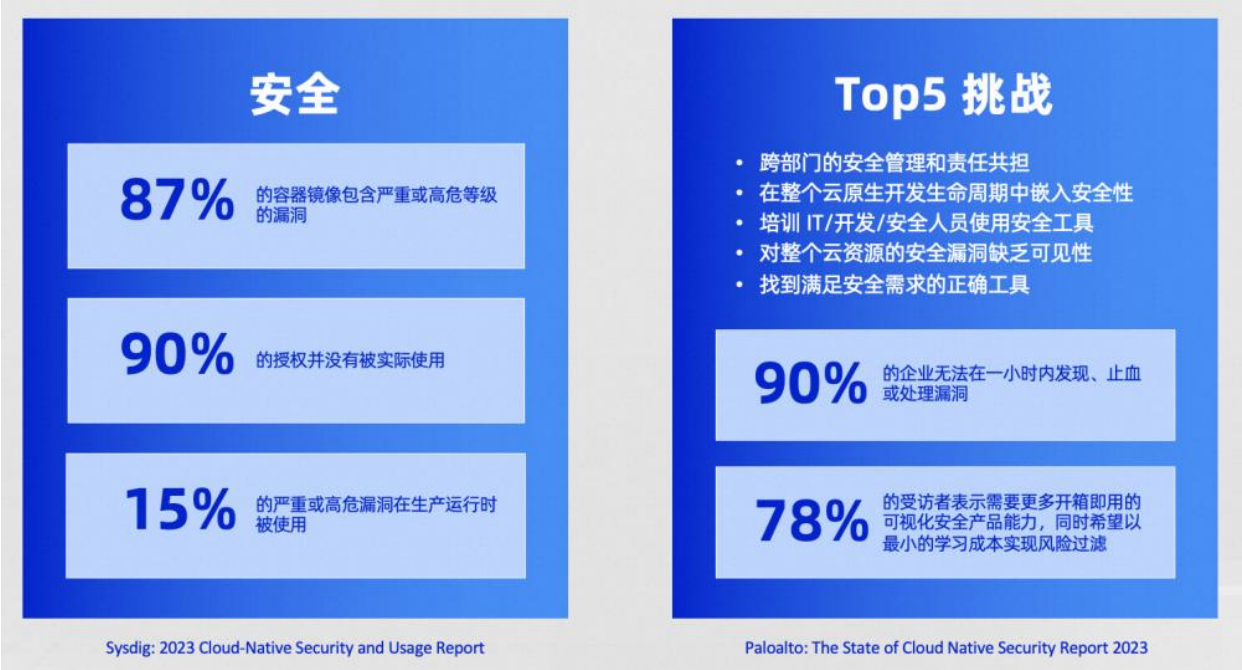
安全一直是企业上云关注的核心问题。随着云原生对云计算基础设施和企业应用架构的重定义，传统的企业安全防护架构已经不能够满足新时期下的安全防护要求。

为此，企业安全人员需要针对云原生时代的安全挑战重新进行系统性的威胁分析并构建适合企业自身的威胁情报系统，同时在云原生安全体系方法论的指导下，结合云服务商提供的安全产品能力构建端到端的 DevSecOps 流程，维持企业应用全生命周期的持续安全水位。

本文分为四部分：

其中第一部分会介绍当下云原生安全的现状以及企业应用在云原生化转型中面临的主要安全挑战；
在第二部分中会概要性介绍云原生安全相对成熟的一部分安全体系方法论；
在第三部分中会结合之前介绍的理论基础，介绍如何通过部署和实时阿里云 ACK 容器服务和 ACR 容器镜像服务中提供的一些实用的安全产品能力，帮助企业实现或优化 DevSecOps 流程；
最后，在第四部分会总结介绍企业在实践 DevSecOps 过程中需要遵循的安全最佳实践。

1. 云原生安全挑战



云原生时代对企业安全的挑战主要来自以下三方面：

- 云原生平台基础设施架构：云原生平台层组件相较于传统架构引入了更多的配置项和隔离层，这就给企业安全管理运维人员提出了更高的运维要求。如何保证平台基础设施层的默认安全性，如何在遵循最小化权限原则基础上进行授权操作，如何建立云原生应用系统的安全审计和监控能力，这些新的挑战都需要云服务商和企业安全管理运维人员协同构建并最终实施到企业云原生化转型后的系统应用架构中。

- DevOps 软件供应链：云原生弹性、敏捷和动态可扩展的特征极大地改变了传统应用部署模式，应用自身的生命周期被大幅缩短，而企业应用的迭代效率则大幅提升，在企业供应链架构变革的同时需要构建和实施适配供应链各阶段的安全防护能力。

- 应用范式上的改变：随着微服务架构的普适，传统的基于南北向流量的安全边界模式已经变得不使用，企业需要更加细粒度的身份认证和访问控制； 同时 Serverless 和函数计算等技术要求云服务商在基础设施层具备更强的安全隔离性和监控能力，而应用的容器形态则需要新的运行时安全监控告警和资产管理模式与之对应。

面对重重的安全挑战，企业的安全现状是如何呢？上图是一些主流云原生安全领域厂商在今年发布的最新报告。其中图片左侧来自 Sysdig 今年的云原生安全使用调查报告，报告显示仍然有 87%的容器镜像中包含严重或高危等级的漏洞，同时 90%的企业应用授权并没有

被实际使用；从右侧 Paloalto 今年的云原生安全现状报告中企业客户反馈的 Top 5 挑战中也可以看出，面对云原生时代新的安全挑战，企业无论在组织架构、文化和安全运维上都还没有做好充分的准备。

供应链安全也是近两年成云原生安全领域的焦点，我们知道创新和效率是企业发展的关键，在云原生时代的企业开发流程中，开源软件和开发工具可以帮助推动企业提升研发效率。在云原生时代，企业对开源生态越来越依赖，三方软件包的安全成为了无法回避的问题。为此 Gartner 预测："到 2025 年，全球将有 45% 的组织的软件供应链受到攻击，比 2021 年增加三倍"。在 sonatype 今年的统计中，仅仅在今年已经有超过 24 万 5 千个软件包中被发现包含漏洞，这个数字是从 19 年到 22 年之合的两倍。

由此可见企业的供应链安全也成为攻击者的主要攻击目标，在调查中我们也发现，多数的受访者都清楚的知道来自供应链的安全风险，但只有不到十分之一的企业受访者表示会在生产供应链生命周期的每个阶段进行安全审核和部署防护措施。由此可见企业安全人员的风险意识与有效的供应链风险管理和防护措施的实施之间还是存在了明显的脱节。

在传统的软件开发流程中，安全人员通常是在系统交付之前才开始介入进行安全审核工作，这样的安全流程显然已经无法满足云原生时代下软件供应链的快速迭代流程。

来自 Gartner 的分析师 David Cearley 早在 2012 年就首次提出了 DevSecOps 的概念。相较与传统软件开发的安全流程，DevSecOps 强调从企业安全文化意识，安全流程左移以及构建全链路的自动化流程等几个要点来加固新时期下企业软件供应链安全。

Gartner 预测，到 2025 年将有 60%的企业采用并实践 DevSecOps。DevSecOps 模型强调安全和整体软件开发流程的紧密结合，同时也强调了在不降低企业应用开发迭代速度的前提下，通过执行全面的自动化安全防护措施来确保应用制品和供应链管道的安全性。

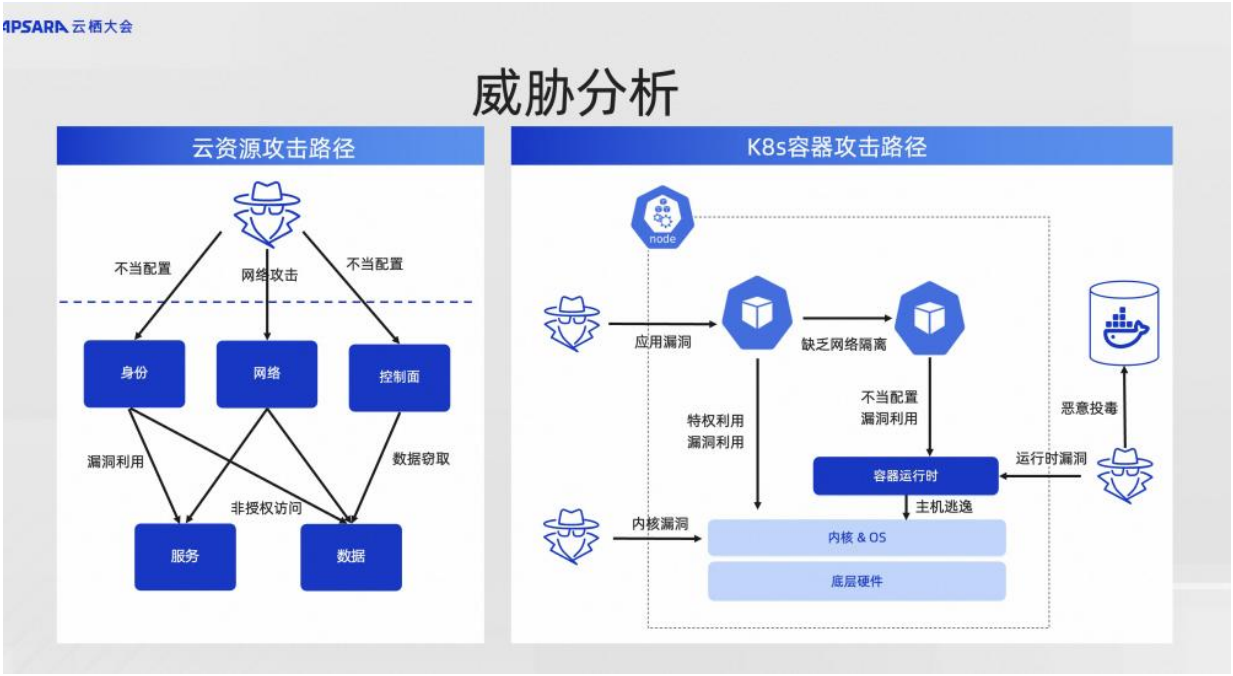
在当下，绝大多数的企业云原生安全的发展都落后于应用的云原生化进程，而主要的改进方向也集中在下面这三个方向：

- 身份和访问管理： 线上授予的权限与实际需要的权限之间存在巨大差异，无疑会给攻击者可乘之机。
- 漏洞和配置管理： 大多数的企业生产镜像都没有经过安全加固和最小化的裁剪收敛，另外很多线上应用因为开发调式的一时方便而在容器层配置了过高的特权
- 监控和响应： 缺少针对容器资产的运行时监控和防护手段，针对突发的攻击事件也无

法有效完成定位和溯源。

这些都是企业应用在云原生化进程中亟需优化和解决的主要安全方向。

2. 云原生安全体系方法论



安全在本质上是一个对系统风险发现、定位和管理的流程。在了解云原生安全，而威胁建模可以在应用设计开发的早期阶段，帮助安全人员识别企业应用架构中潜藏的安全风险和设计缺陷。

在上图中，左边是针对云上资源的典型攻击路径分析，我们看到在传统的云服务架构下，针对身份和控制面的不当配置以及网络攻击是攻击者可以利用的主要途径，攻击者可以通过漏洞利用、非授权访问和数据窃取等手段攻击企业服务和数据；

而在右边的云原生 k8s 集群的典型攻击路径中，由于云原生技术架构的复杂性，容器应用、运行时、k8s 编排引擎、镜像仓库以及内核层都可能给整个应用系统引入新的风险，同时在网络侧，不同容器微服务应用之间的东西向流量也提供给攻击者更多的可利用目标。而近年来不断爆出云原生社区相关的 CVE 漏洞中，攻击者可以利用的攻击方式也是多种多样，

像一般的提权、仿冒、篡改、抵赖、拒绝服务等典型攻击手段都出现在了近两年公开披露的漏洞利用方式中。

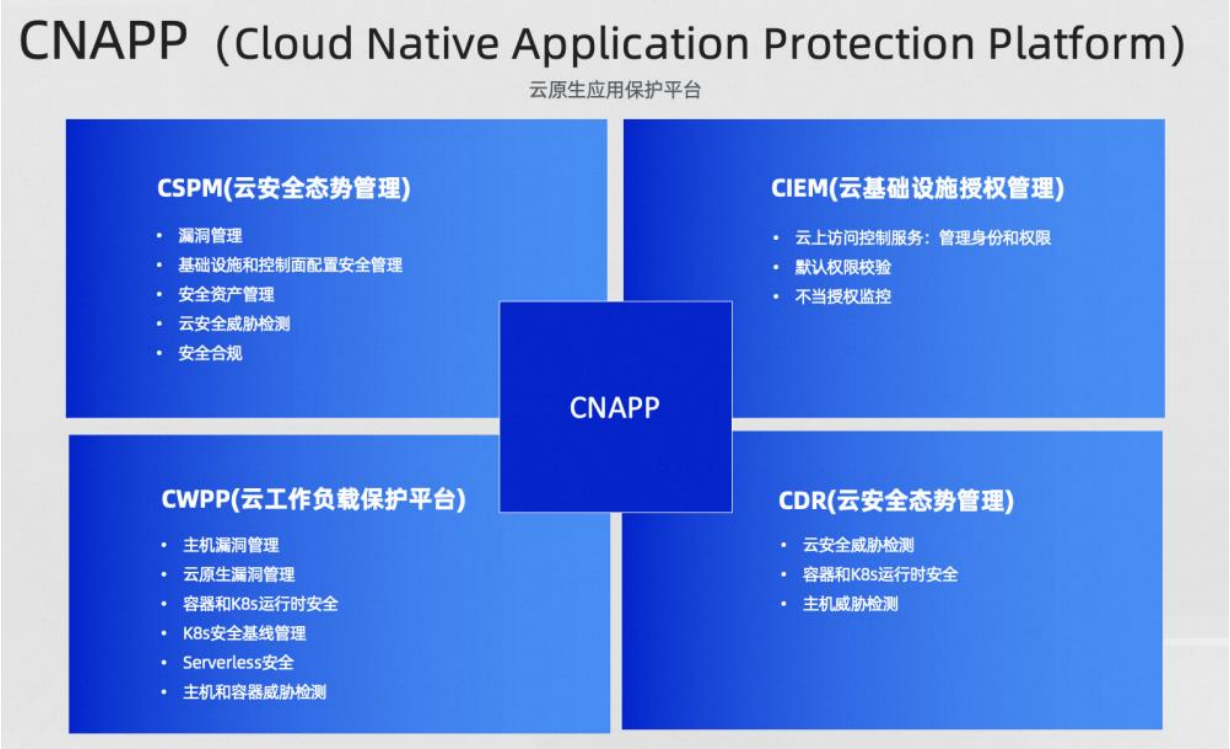
为此，需要企业安全人员在传统基础设施架构威胁之外尽可能全面的获取安全威胁信息。同时在企业应用架构发生动态变化的同时，也需要重新评估当前威胁建模下的安全机制和模型矩阵是否也需要随之调整。

阿里云容器安全ATT&CK攻防矩阵

初始入侵	下发指令	持久控制	权限提升	躲避防御	窃取凭证	探测信息	横向攻击	达成目标
云账号AK泄露	通过kubectl进入容器	部署远控容器	利用特权容器逃逸	容器及宿主机日志清理	K8s Secret泄露	访问K8s API Server	窃取凭证攻击云服务	破坏系统及数据
使用恶意镜像	创建后门容器	通过挂载目录向宿主机写文件	特权RBAC绑定	K8s Audit日志清理	云产品AK泄露	访问Kubelet API	通过Service Account访问K8s API	劫持资源
K8s API Server未授权访问	通过K8s控制器部署后门容器	K8s cronjob持久化	利用挂载目录逃逸	利用系统Pod伪装	K8s Service Account凭证泄露	Cluster内网扫描	Cluster内网渗透	DoS
K8s kubeconfig泄露	利用特权Service Account连接API Server执行指令	在私有镜像库的镜像中植入后门	通过Linux内核漏洞逃逸	通过代理或匿名网络访问K8s API Server	应用层API凭证泄露	访问云厂商服务接口	通过挂载目录逃逸到宿主机	加密勒索
容器内应用漏洞入侵	带有SSH服务的容器	注入恶意admission controller	通过运行时漏洞逃逸	清理安全产品Agent	获取管理面身份凭据	访问私有镜像库	CoreDNS投毒	
私有镜像库暴露	通过云厂商CloudShell下发指令		利用K8s漏洞进行提权		注入恶意admission controller	访问ECS Metadata API	ARP和IP欺骗攻击	
Master节点SSH登录凭证泄露	Sidecar注入		利用Linux Capabilities逃逸			通过NodePort访问Service	攻击第三方K8s插件	
	利用应用漏洞远程命令攻击		获取云资源访问权限					

ATT＆CK 框架是网络攻击中涉及的已知策略和技术的知识库，其中阿里云也是国内首家针对云原生容器和 Kubernetes 的攻防场景，提出并发布相应 ATT&CK 攻防矩阵的云服务商。在 ATT＆CK 矩阵中详细描述了攻击者在云原生和 Kubernetes 环境发起攻击的过程和手段，可以帮助企业构建容器化应用安全体系，也是企业构建云原生威胁情报体系可以利用和借鉴的基本框架。

整个威胁矩阵由不同的行和列组成，其中行代表了攻击技术，列代表了攻击战术手段。矩阵从左至右可以代表一个通常的容器侧攻击路径。通过了解矩阵中每一个攻击阶段攻击者可以利用的技术手段，可以帮助企业安全运维人员有针对性地进行安全设计和测试演练，当安全事件发生时，也可以帮助有效实施对应的止血和预先的防护措施。



为了进一步理解云原生应用安全风险并构建完整的安全防护方案，企业安全运维人员还需要先进的分析方法，覆盖对应用侧风险、开源组件风险、云基础设施风险和应用运行时风险的完整感知。

为此 Gartner 牵头在已有的云安全态势管理(CSPM)、云基础设施授权管理（CIEM）和云工作负载保护平台（CWPP）等传统主流云平台安全模型的基础上，提出了 CNAPP 云原生应用保护平台框架，面向云原生应用从开发到运行时刻的全生命周期流程，帮助企业安全团队和 DevSecOps 架构师提供完整视角的应用风险可见性和相应的解决方案。基于 CNAPP 理论框架，信通院在 2022 年云原生产业联盟年会上发布了《云原生应用保护平台（CNAPP）能力要求》，在 CNAPP 理论框架的基础上，进一步细化了规范要求。

从架构图中可以看到 CNAPP 框架集成了之前多个成熟规范的核心特性，可以：

- 更好地适配云原生应用高速迭代的敏捷特性，通过自动化手段减少错误配置和管理。
- 减少参与供应链 CI/CD 管道的工具数量。
- 降低云原生应用安全合规实施复杂性和成本。

对于企业应用，安全需求分析、安全测试、安全开发和反复的加固措施也同时伴随着应用的迭代。我们知道企业安全文化意识以及开发、安全运维团队之间的流程协同是 DevSecOps 能够有效实施的关键，在 CNAPP 框架中，也同样强调研发和运维侧双向反馈飞轮，加强企业安全可视性和对风险的洞察力，从整体上改善企业安全态势。

上图中的研发和运维侧双向反馈飞轮主要分为下面两个方向：

从开发到生产：基于安全左移原则，将安全集成到开发人员的工具链中，在代码创建阶段就通过自动化构建管道触发安全测试，以降低后续安全风险和运维成本。

从生产到开发：需要企业安全管理人员全面监控线上应用和平台配置，并结合运行时安全配置上下文，提前识别风险并考虑风险处理等级预案，同时将相应的加固措施落实到新的开发迭代流程中。

只有通过这样不断循环反馈，才能保证在云原生下应用的高速迭代的过程中持续的安全水位。

企业在 DevSecOps 的落地实践中可以充分利用云原生技术，同时结合云服务上的安全产品能力以及部署成熟的安全三方产品也可以帮助企业快速构建 DevSecOps 能力。

企业应用的安全性需要贯穿应用程序的整个生命周期。开发是整个应用生命周期的第一个阶段，其结果是创建用于部署和配置应用的云原生模版、容器镜像、应用二进制等云原生制品，这些制品正是运行时被攻击利用的主要源头，相应这个阶段针对不同制品安全扫描就显得格外重要。

构建分发阶段包括基于 CI/CD 自动化流程的各种系统测试，尤其针对开源软件，需要进行明确的漏洞风险分级卡点机制，同时加入自动化的加签机制以支持制品的可信校验。

部署阶段负责进行一系列 “飞行前 ” 检查，以确保将部署到运行环境中的应用程序符合整个企业组织的安全和合规规范。

运行时阶段的安全包括计算、存储和访问控制三个关键领域的安全能力。运行时环境的安全性取决于前几个阶段安全实践的有效性，同时需要企业在配置管理、身份和访问控制以及运行时威胁检测方向上基于安全原则实现高效的自动化监控和管理能力，并且通过全局性的安全资产管理和态势感知能力不断发现风险并反馈到生产开发及供应链各环节。

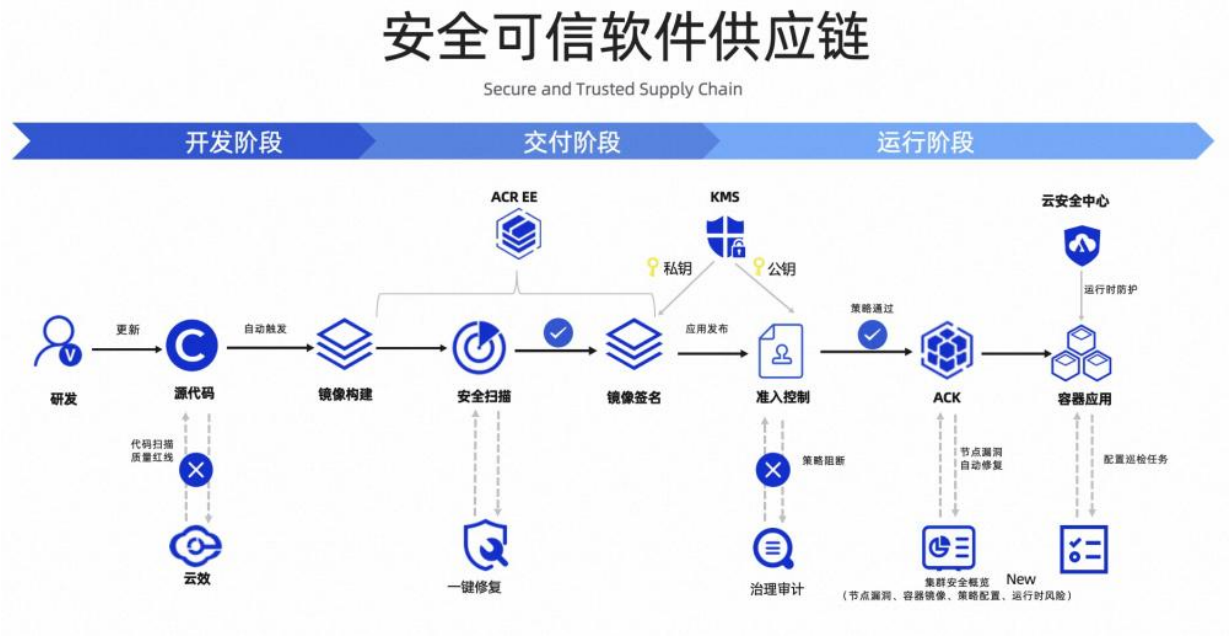
最后，企业应用安全需要防患于未然，完备的审计和溯源分析能力以及精准的风险阻断能力可以帮助企业安全运维人员从容应对突发的攻击事件，并在规划的指导下做出快速的决策和响应。

最后，安全左移和循环反馈是指导企业基于 DevSecOps 理念实施安全防护的主要原则。在应用制品的供应链生命周期中应尽早地以自动化方式嵌入安全，通过引入自动化的安全扫描和巡检机制，在早期发现并治理常见的软件 CVE 漏洞，可以帮助团队以低成本的方式扼杀风险，同时整体提升团队安全意识。

企业在落地并实践了安全左移理念后，并不意味着安全工作的结束。在应用的生产运行阶段，安全管理人员可以采集并掌握更全面的安全上下文信息，并通过威胁分析发现更多在供应链环节无法暴露的安全设计和配置上的问题，只有通过不断的反馈和更新，才能够保持企业应用系统的整体安全水位，同时应对无法预知的安全挑战。

3. 阿里云容器服务 ACK & 容器镜像服务 ACR 安全产品能力

通过上面的介绍，我们对云原生安全面临的挑战以及当下比较成熟的云原生安全理论体系有了初步的了解，下面我会具体介绍阿里云 ACK 和 ACR 服务中面向企业提供了哪些安全相关的产品能力，可以帮助企业实现或优化 DevSecOps 流程。



首先，围绕 DevSecOps 流程中的核心防护能力和自动化要求，阿里云 ACK 和 ACR 服务也沉淀了 DevSecOps 产品能力，帮助企业实现安全可信的软件供应链。

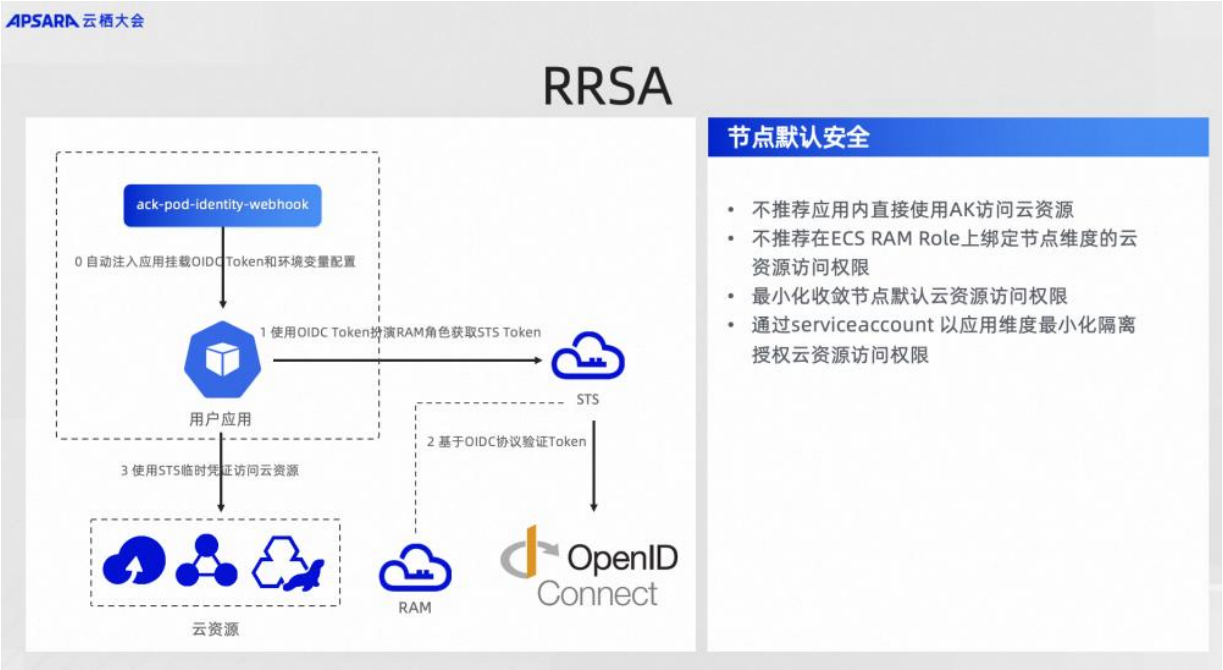
在镜像构建阶段，客户提交源代码变更后，自动触发 ACR EE 的云原生应用交付链功能，开始容器镜像构建，支持自动安全扫描，如果识别到镜像中存在风险，会自动阻断构建流程并钉钉报警。用户可以一键修复镜像中存在的 CVE 漏洞。

在镜像交付阶段，ACR 和 ACK 可以基于客户的秘钥进行镜像的加签与验签，确保整个交付链路无篡改。在容器应用运行时，云安全中心可以对 ACK 集群中运行时风险进行防护、感知，以及阻断处理。

今年，我们推出了集群容器安全概览功能，可以帮助企业安全管理员更好感知集群配置、应用镜像、容器运行时的安全风险，提升整体安全水位。

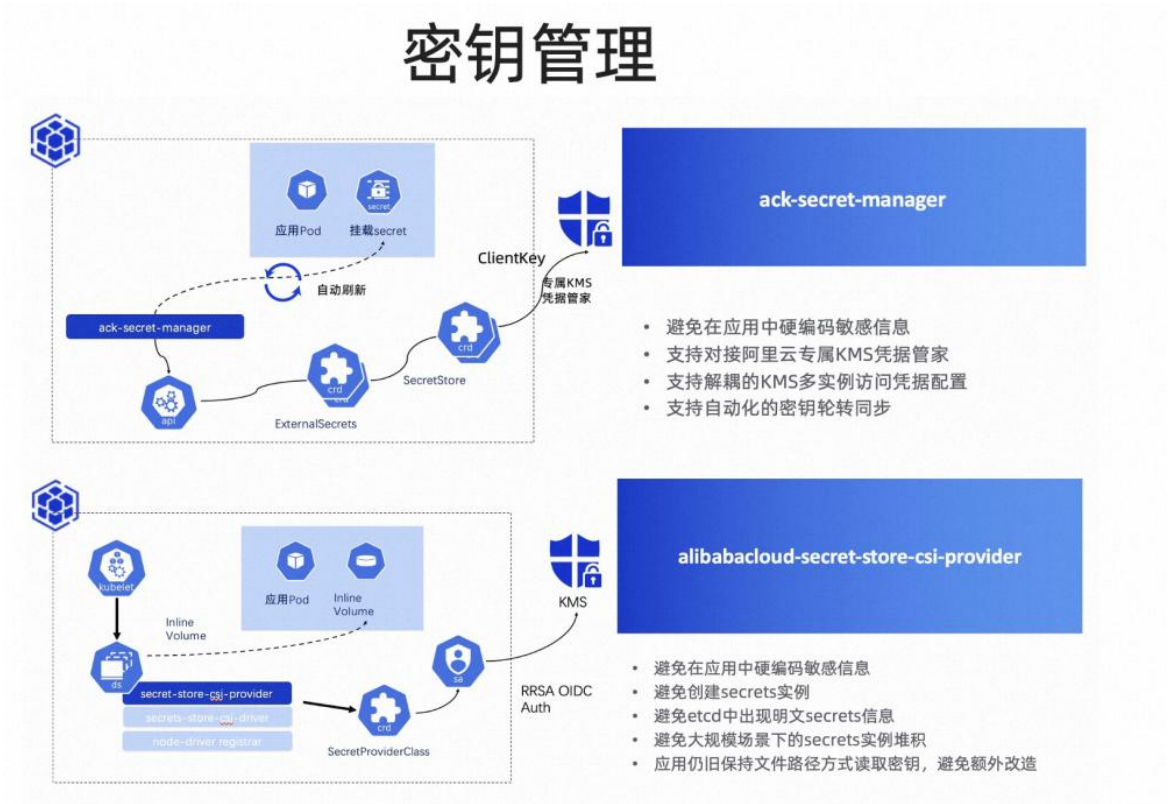
DevSecOps 的实践依赖企业内部深层次的协同，而不同的工具和实践流程会阻碍跨部门协作的高效性和生产力。通过标准化的平台能够简化部门间沟通和学习成本，让供应链流程更加透明高效

ACK、ACR 充分利用云原生技术，在企业供应链流程的关键路径上构建了核心能力，面向企业安全管理员提供了开箱即用的产品能力，安全人员可以通过简单的可视化白屏操作完成制品校验，安全巡检、策略实施等供应链安全防护能力，同时通过不同维度的可视化监控和报表直观的了解并管理应用安全，帮助企业安全管理人员第一时间洞察风险，应对攻击。



默认状态下的安全性是整个系统安全体系的根基，不仅决定了生产系统的稳定性，也直接关联了上层安全方案的运维和实施成本。为此，在企业应用系统的设计阶段，安全性就应当作为基本且必要的需求融入设计环节，并在安全专家的指导下审核架构设计中潜藏的风险。

企业应用使用哪种凭据访问云资源？基于默认安全和权限最小化原则，在应用内直接使用 AK 以及在节点上直接绑定云资源访问权限都是安全上不推荐的做法，此时可以将 RRSA 方案集成到企业应用设计和编码环节，通过 RRSA 方案，可以实现基于 k8s serviceaccount，与阿里云 RAM 服务完成基于 OIDC 协议的访问控制认证，并实现以应用维度最小化隔离授权云资源访问权限。



密钥管理一直是企业应用上云的核心问题，云服务商有哪些安全方案可以帮助保护应用密钥？用户又应该在 K8s 环境中采取哪些安全措施帮助管理和使用密钥？应用密钥应该存储在哪里？这些都是企业客户经常会问到的一些基本问题。

为此，首先应避免密钥在应用中的硬编码问题。在应用系统开发部署的供应链流程中，任何一个环节对敏感密钥的硬编码都有可能导致泄露风险。通过使用云上 KMS 服务，可以在应用开发、测试、构建等生命周期流程中使用统一方式进行密钥的读写，避免硬编码出现；同时云上的 KMS 服务支持自动化的密钥轮转能力，进一步降低了敏感信息泄露传播的安全风险，同时也可帮助企业应用满足合规需求。

通过部署 ack-secret-manager 插件可以将 KMS 凭据管家中管理的企业密钥以 k8s secret 的形式同步导入到业务集群中，企业应用可以直接通过文件挂载的方式使用密钥，在避免密钥硬编码问题的同时保证了业务代码的兼容性，在新版本的 ack-secret-manager 中还支持对接 KMS 服务升级后的专属 KMS 实例，提供更强的密钥安全性隔离保证。

K8s 社区基于 CSI 存储标准扩展实现了 secrets-store-csi-driver 用于将存储在外部密钥管

理服务中的密钥以 volume 存储卷的形式挂载到应用 pods 中。和 ack-secret-manager 方案不同，该机制避免了 K8s secret 实例的创建，带来的好处一是避免 etcd 中出现明文 secret 信息，二来可以在大规模场景下避免 secret 堆积；同时应用仍旧可以保持原先的文件路径方式读取密钥，避免了额外的程序改造代价。

基于该插件机制我们实现了阿里云自己的 secrets-store-csi-driver-provider，并且支持通过 ACK 应用市场在集群中一键化部署该插件，同样可以将阿里云 KMS 凭据管家中保存的密钥凭据以文件形式同步到应用容器中，同时支持后端凭据修改后的同步更新能力，保证业务容器中密钥信息的实时性。

当然这里也会有“最后一把密钥”的问题，由于插件需要调用 KMS 凭据管家服务的权限，如何在集群中保护插件对 KMS 服务请求的凭据呢？这里推荐使用 RRSA 方案，可以将 KMS 凭据的请求权限绑定在插件使用的独立 serviceaccount 上，避免将权限泄露给应用 pod 中。

对于数据安全性有严格要求的场景，比如当下火热的 AI 大模型，金融支付、隐私认证或涉及知识产权的核心数据计算，除了保证这些核心敏感信息在读写和传输过程中的安全性，还需保证机密信息在云上节点内存运算和存储过程中的安全可信。今年我们还和 Intel 以及 CoCo 社区合作，基于 Intel TDX 机型实现了新一代的可信执行加密环境，帮助实现企业数

据全生命周期的安全可信。

在应用运行时，云原生工作负载区别于传统基于虚机的应用服务有如下特点：

- 短暂的生命周期，只有秒级的生命周期；
- 编排引擎会根据节点实时资源动态调度工作负载，网络 IP 等应用元数据可能随应用重启不断变化；

出于基础设施不可变性，对运行时环境的修改在工作负载重启后不会保留。

正因为云原生工作负载自身特点，在应用运行时，当前大多数云原生安全产品对容器侧用户态进程的检测分析都存在不足。而 eBPF 天然的技术优势是提升云原生应用安全可观测性和实现精细化安全事件响应的有力武器。

在 ACK 集群中，我们提供了针对 k8s 应用的 exec 活动审计能力，通过 eBPF agent 可以实时获取容器负载中执行的系统调用，并关联映射到在容器实例中执行的具体进程，从而帮助安全运维人员获取攻击者进入到容器实例后发起攻击的命令审计，有效帮助针对安全事件的溯源和止血。

同时我们也和 SLS 日志服务提供的强大日志分析检索能力结合，针对云原生的典型漏洞，

提供了可疑的漏洞利用活动的溯源和告警能力，并且通过时间线图表的方式直观的展现给企业安全人员。

员工离职后的权限清理问题也是困扰很多企业权限管理员的难题。在 ACK 默认提供的 x509 证书认证模式下，企业安全运维人员很可能由于疏漏在删除 RAM 账号或角色前忘记吊销和清理 kubeconfig 权限。

通过 ack-ram-authenticator 组件，ACK 托管集群可以通过 Webhook 方式、基于阿里云 RAM 完成请求认证。相较于 ACK 集群默认提供的 x509 证书认证模式，使用 ack-ram-authenticator Webhook 认证方式有如下优点：

适配企业通过云 SSO 场景，提供灵活可控的数据面 RBAC 授权；

角色 SSO 对接场景下 apiserver 审计日志中包含企业 IDP 中的身份信息，有效支持对扮演同一角色的不同 IDP 用户的行为审计；

企业员工离职删除 RAM 账号或 RAM 角色时，可自动清理其在账号所有 ACK 集群中的认证访问权限。

4. 企业 DevSecOps 安全最佳实践

“木桶的最大容积取决于最短的一块木板”，云原生安全同样遵循这样的木桶原则。由于云原生技术栈的复杂性，企业安全管理和运维人员更需要在安全准则的指导下，全面充分的了解全局安全风险，提升系统“最低点”安全水位。

零信任安全最早由 Forrester 首席分析师 John Kindervag 在 2010 年提出，其核心思想是“Never Trust， Always Verify”。在零信任安全模型中，只要处于网络中，默认情况下任何用户都不可信，任何时刻任何环境下设备和服务的身份权限都需要被持续验证。

权限最小化原则是企业安全运维中最基本也是最重要的准则之一。传统应用架构下，系统的权限管理员需要基于权限最小化原则对内部人员授权。在云原生架构下，授权管理不止针对企业中的账号系统，同时需要考虑松耦合微服务架构下众多服务身份的授权

安全左移不仅可以有效降低软件自身漏洞而导致的应用风险，同时也能够有效降低企业开发运维成本。

企业安全管理人员需要在安全系统设计中规划和覆盖应用周期中的每个阶段，在安全左移和循环反馈原则的指导下,结合 CNAPP 等规范框架的理论指导下完成安全产品能力的建设。

这里也列举了企业生产供应链中在开发，构建部署、运行时和反馈阶段环节需要具备的一些核心能力，时间关系就不一一介绍了。

DevSecOps 在企业的落地实践离不开企业文化理念上的转变，在 DevSecOps 体系中，安全应当是企业内部团队共同的目标，而不应只是安全团队自身的职责；企业开发、运维和安全团队应当协同起来，设定统一的目标并共担责任，同时定义团队之间的交流互动方式，能够有效提升业务迭代效率。

下面几个方向是企业管理人员在企业文化 DevSecOps 转型中需要关注的重点方向：

人员：拥有合适的人才是 DevSecOps 的基础。安全培训和培养安全拥护者一直是让安全变得重要的首选解决方案；同时 DevSecOps 需要在安全和效率之间取得适当的平衡，另外持续学习，掌握最新的漏洞和策略对于保证应用程序的安全至关重要。

流程：制定正确的流程可以确保每个人都站在同一起跑线上，并为安全一致性和凝聚力奠定基础。

工具：成功实施 DevSecOps 战略的最后一个要素是工具。Kubernetes 安全领域拥有众多工具，可以解决 Kubernetes 和云原生安全的各个层面的问题。当然，在研究和实施安全工具时，也需要避免使用存在重大缺陷的工具。

绩效：在重视生产效率和版本迭代的前提下，可以将 DevSecOps 的实施列入团队绩效考核，并且通过一些具体指标和分级问责机制的建立也是让 DevSecOps 快速融入团队的有效途径。

结语

最后，欢迎大家选择和使用更多的阿里云 ACK 和 ACR 服务中提供的安全产品能力，让我们共同努力，让 DevSecOps 流程落地实践到更多的企业生产流程中。

机密计算容器前沿探索与 AI 场景应用

作者：李鹏（壮怀），阿里云高级技术专家、朱江云，英特尔软件与先进技术事业部高级经理

企业与个人对数据隐私保护日益关切，从数据，网络的可信基础设施扩展到闭环可信的计算基础设施，可信的计算，存储，网络基础设施必定成为云计算的标配。机密计算技术应运而生，其中一个重要的技术是通过芯片的可信执行环境 TEE 实现数据保护。在 TEE 内执行的应用，不用担心来自其他应用、其他租户、平台方甚至运维内部团队的安全隐患。

为了解决企业对数据隐私日益关切，阿里云、达摩院操作系统实验室与 Intel 和龙蜥社区一起，推出基于可信执行环境(TEE)的机密计算容器(Confidential Containers，简称 CoCo)在云上的参考架构。企业可以通过容器服务 ACK TDX 机密沙箱容器节点池实现端到端零信任的应用、数据和模型保护。

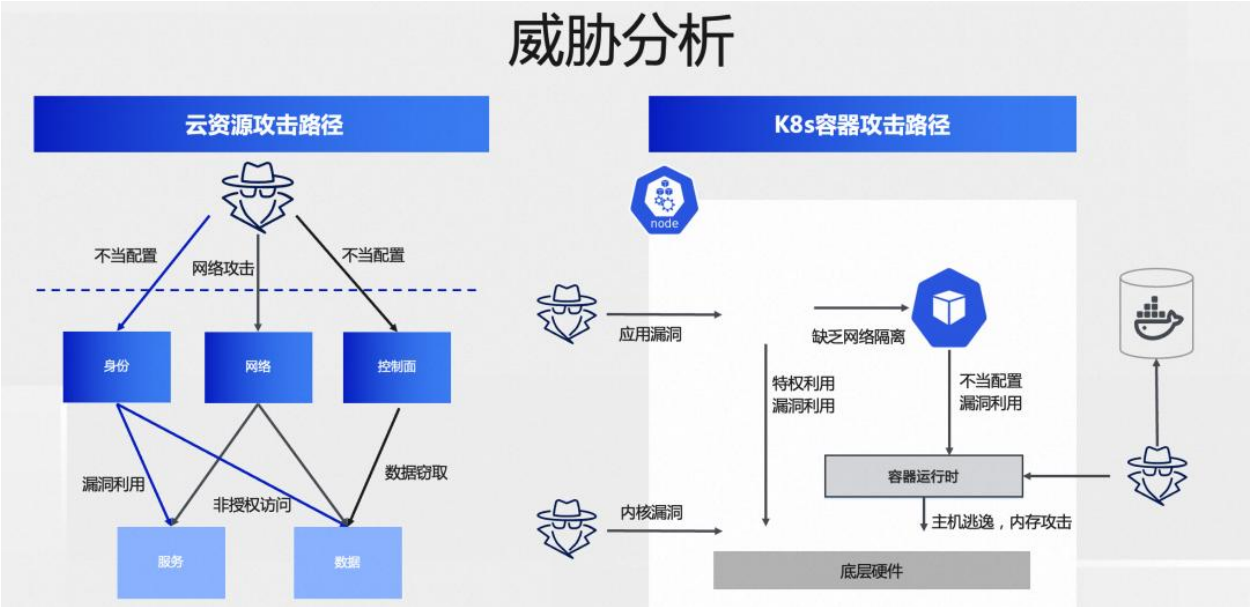
在 2023 年云栖大会现场，阿里云容器服务高级技术专家壮怀和英特尔中国软件与先进技术事业部的高级经理朱江云共同分享了阿里云容器服务团队与社区和生态伙伴一起，在机密容器领域的探索、安全特性的演进，以及关于如何通过机密容器来保护 AI 应用的数据、模型以及计算展开探讨。

1. ACK 端到端可信容器，为数据安全护航

阿里云容器服务高级技术专家壮怀首先分享了对当前容器运行时安全主要威胁的分析、企业应该坚守的安全原则及阿里云容器服务如何与机密计算领域生态伙伴一起，为客户提供端到端可信容器，为企业数据安全保驾护航。

1）容器运行时安全威胁

通常来说，我们对于企业安全的定义是是“在不加剧安全漏洞的情况下，您能否继续高效/安全地工作”。保证容器运行时安全需要通过最小化权限、零信任的原则，以 Never trust, always verfy 的方式思考 IT 设施各个组件之间的交互方式，思考计算如何做到零信任。



云计算构建了 RAM 鉴权体系、KMS 的密钥密文的管理能力、存储的 BYOK 加密技术、VPC、安全组、身份认证、鉴权、策略治理等等，即便如此，企业仍需思考是否足够解决云计算中信任问题，如计算过程数据的安全性如何保护、进程的计算过程对 root 的运维透明性如何防御等。

安全的好坏取决于 “最薄弱的环节”，是大家都知道木桶原则，短板决定了容量，短板决定了安全水位，云计算信任问题在解决了存储和网络相关信任问题，更聚焦到了计算的信任问题。实现安全的过程是对企业资源、所需专业知识、时间管理、实施成本、数据备份/恢复等的 “风险管理”。当今的安全趋势是以安全左移，安全贯穿于开发，构建等更早期的阶段，数据的安全性依然需要贯彻于存储、网络和计算的三项基础设施。

企业对安全的需要是全天候的、持续不断的、永无止境的，安全就是 “在不对网络、生产效率和预算造成负面影响的情况下，以最快的速度学习所有可以学习的知识”。

今天第三代的安全容器技术，正是遵循这个原则，从早期的需要侵入式的改造的 SGX1.0，到可以对更大内存空间做机密计算的 SGX2.0，到今天应用无感的平滑迁移进入安全容器技术（TDX/SEV/CCA）。从金融领域，扩展到今天的通用人工智能 AGI 的数据，模型保

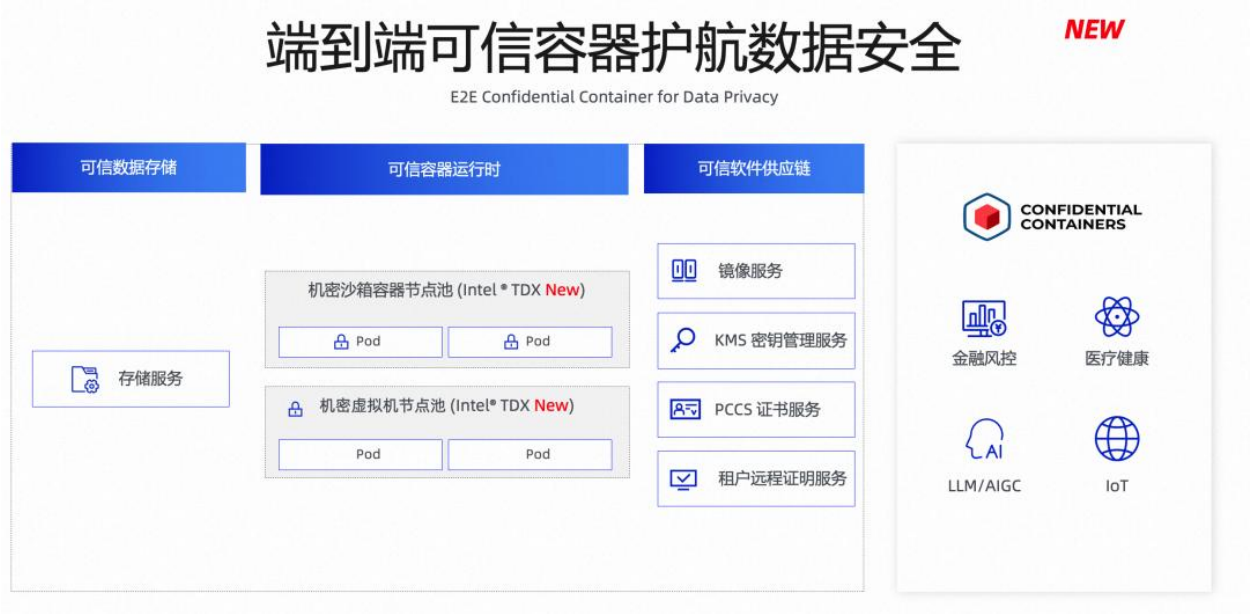
护。从数据，网络的可信基础设施到闭环可信的计算，可信的基础设施必定成为云计算的标配。运行时的安全，有以下 5 个主要的安全威胁都可能会导致租户容器内的敏感数据遭到泄露：

- 非授权部署
- 错误配置
- 恶意镜像
- 漏洞利用
- 提权攻击和内存溢出/数据攻击

在云环境中运行容器时，底层基础设施的安全性和云服务提供商的可信度变得至关重要。如果云服务提供商受到入侵或缺乏适当的安全措施，容器内的敏感数据（如凭据、加密密钥或个人身份信息）可能会被未经授权的人员访问或窃取。今天云原生的安全手段通过相应的手段来治理和防护：

- OPA 策略治理应对授权和部署攻击
- 配置巡检应对配置漏洞
- 镜像扫描和 BinaryAuth 防范恶意镜像攻击
- CVE 修复和自动化运维升级抑制漏洞利用攻击

而对于上述第 5 中提到的“提权攻击和内存溢出/数据攻击”， 则需要使用机密虚拟机或者机密沙箱容器来做软硬一体的可信的计算来从根源上治理。

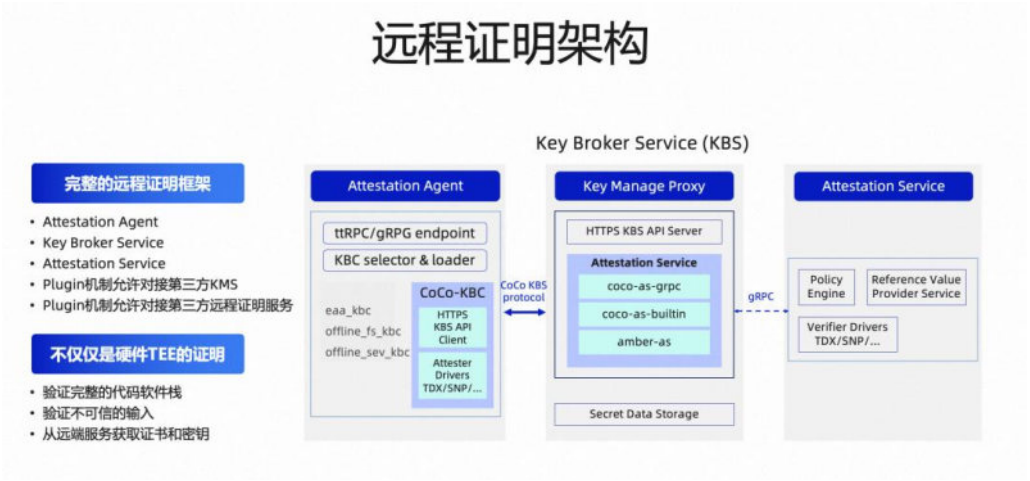


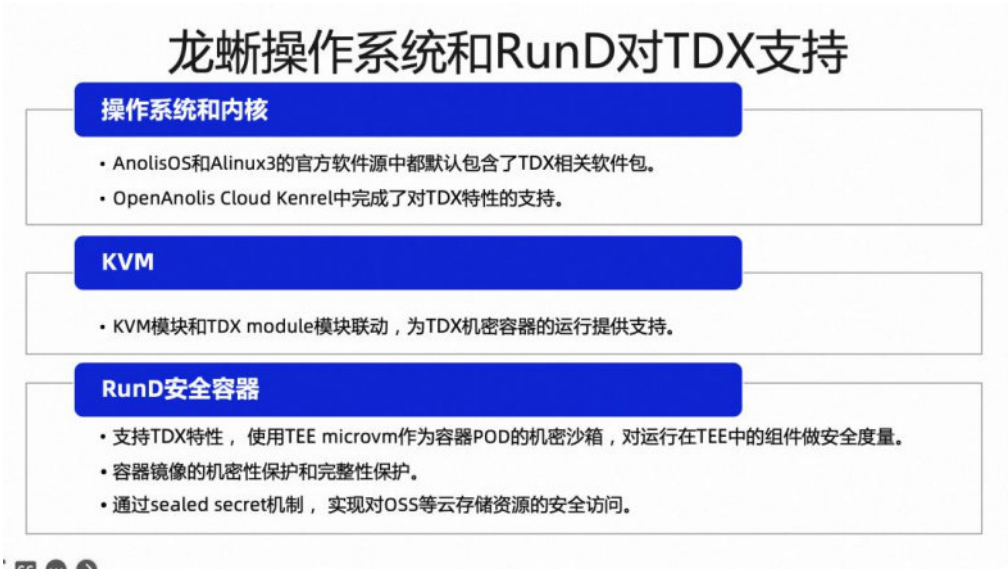
阿里云与 Intel 和龙蜥社区一起，推出机密容器和通用云服务融合的参考架构，三方结合阿里云八代裸金属（Intel）和八代的机密虚拟机实例，KMS，OSS，ACK/ACR 等云服务提供参考解决方案。通过 ACK TDX 机密沙箱容器实现端到端零信任的应用，数据和模型保护。

通过 ACK 机密虚拟机节点池，企业无需对应用本身修改，直接部署云原生应用到机密虚拟机节点池，应用可以无缝切换高安全水位，支持多种机密计算场景，如金融风控、医疗健康数据隐私保护，AIGC/LLM 推理和微调，机密数据库，大数据应用等等。

2）操作系统和 RunD 对 TDX 支持

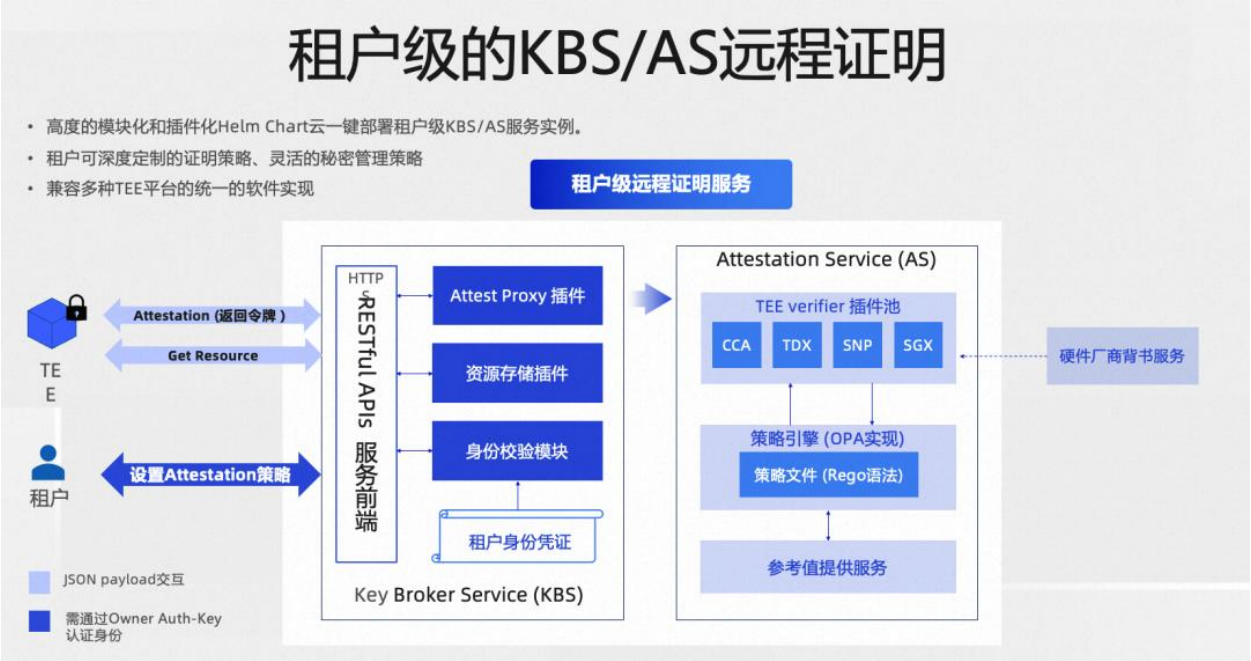
RunD 安全容器是龙蜥社区开源的下一代容器解决方案，包含 Rust Kata runtime 和 Dragonball VMM。RunD 安全容器已经于 2022 年由龙蜥云原生 SIG 开源至 Kata Containers 社区，且作为 Kata Container 3.0.0 release 的重要特性。目前龙蜥社区已经完成 Host OS 、Guest OS 和 RunD 安全容器对 TDX 硬件的支持工作，并提供机密容器解决方案的端到端支持。



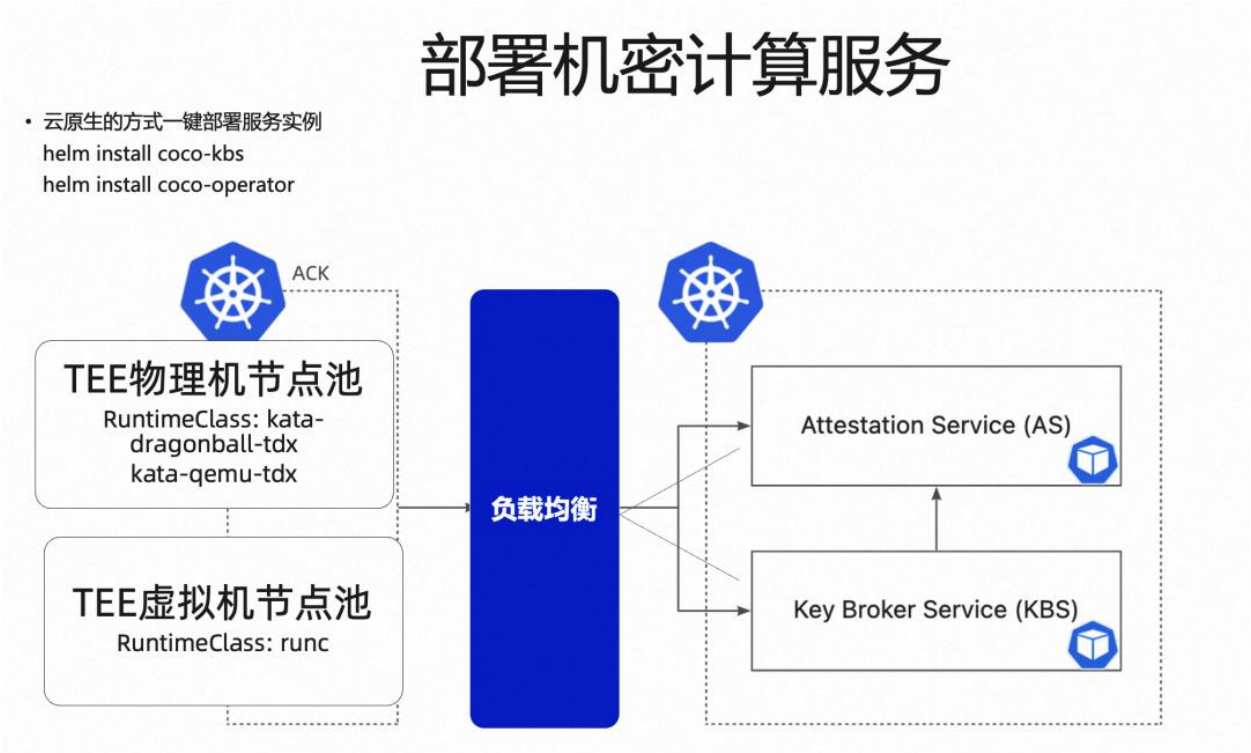


3）租户级的远程证明

ACK 提供的多租户的远程证明服务提供了完整的租户级远程证明框架，用以支持建立用户对 TEE 从硬件到软件的全栈信任，从而实现注入密钥和证书注入等一系列关键的安全需求。达摩院操作系统实验室致力于研究远程证明架构对应用负载的完整可信，通过 Attestation Agent 运行在 TEE 内（这里的 TEE 主要包括机密虚拟机和机密沙箱内部）收集证据并发送给租户级服务 KBS，KBS 通过将证据转发给后端的 Attestation Service 对证据进行验证，然后向 TEE 内返回证明结果以及所需的秘密资源数据，从而达到对于应用负载，代码，配置，输入的安全度量。



远程证明体系整体采用模块化和插件化设计，以统一的软件架构兼容多种 TEE 平台。KBS 通过 RESTful API 接收来自 TEE 或者租户的请求，在 KBS 内部我们实现了灵活的资源存储插件和 Attest Proxy 插件，从而允许在实际场景中对接不同的第三方存储服务和 Attestation Service。在后端的 Attestation Service 中，集成了 OPA 实现的策略引擎以支持租户深度定制的证明策略。通过 ACK 应用市场可以实现远程证明服务的组件化部署和定制化。



在 ACK Pro 集群中可以通过部署远程证明服务，添加节点池，和部署运行时三个步骤来部署机密计算服务。

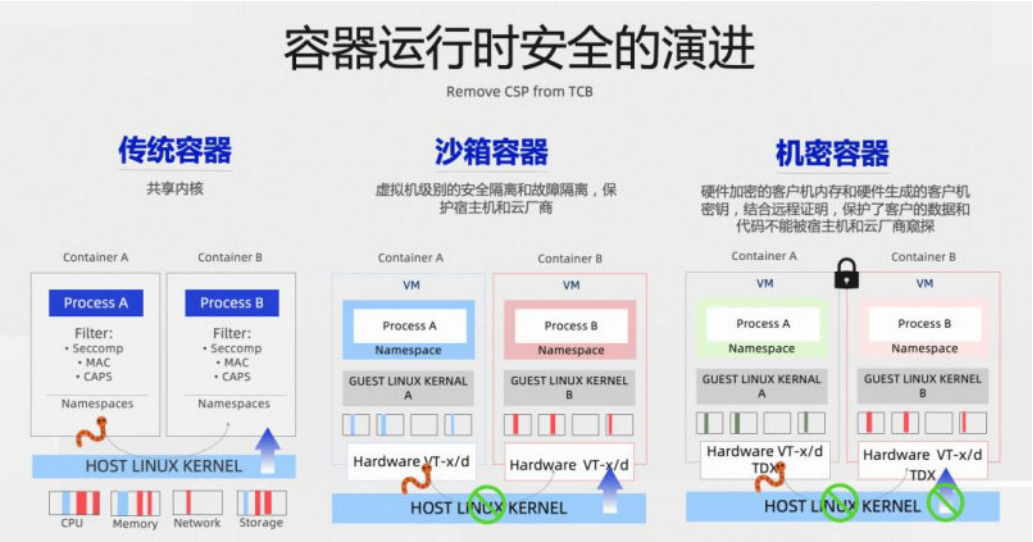
通过选择 ECS 8 代 Intel 物理机来构建 TEE 的安全沙箱容器节点池，或者选择 ECS 8 代 Intel 的虚拟机开启机密特性来构建 TEE 的机密虚拟机节点池。

通过 ACK 应用市场，云原生的方式一键部署远程证明和代理服务实例，helm install coco-kbs。

通过 ACK 应用市场部署 coco-operator 来提供两种新的容器运行时，kata-dragonball-tdx，kata-qemu-tdx 以及增强安全特性后的 runc，helm install coco-operator。

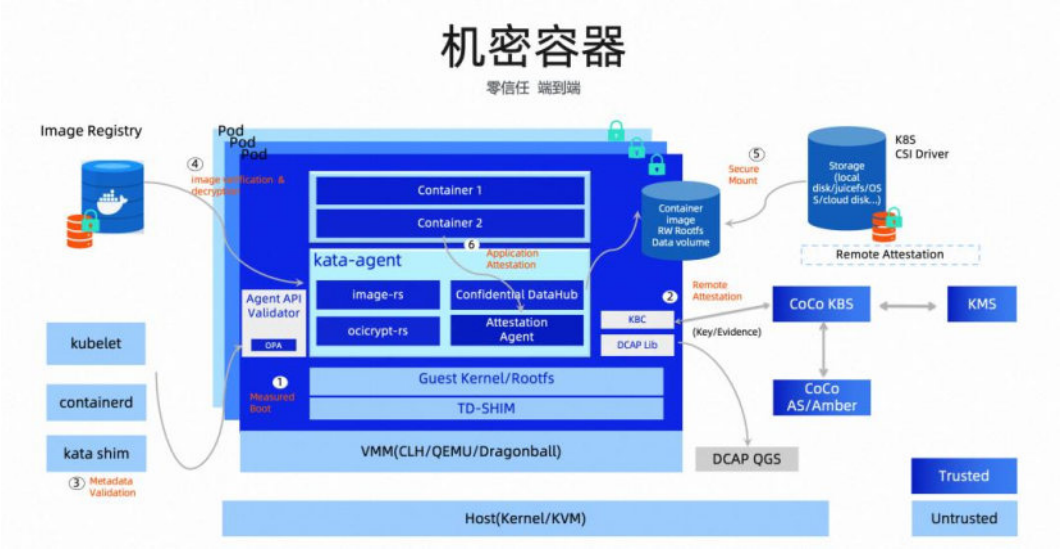
2. 机密容器关键安全特性探索实践

来自英特尔中国软件与先进技术事业部的高级经理朱江云代表 ACK 机密容器生态合作重要伙伴，向观众分享了容器运维行安全的演进、机密容器关键安全特性的发展以及在 AI 等前沿领域的探索落地。



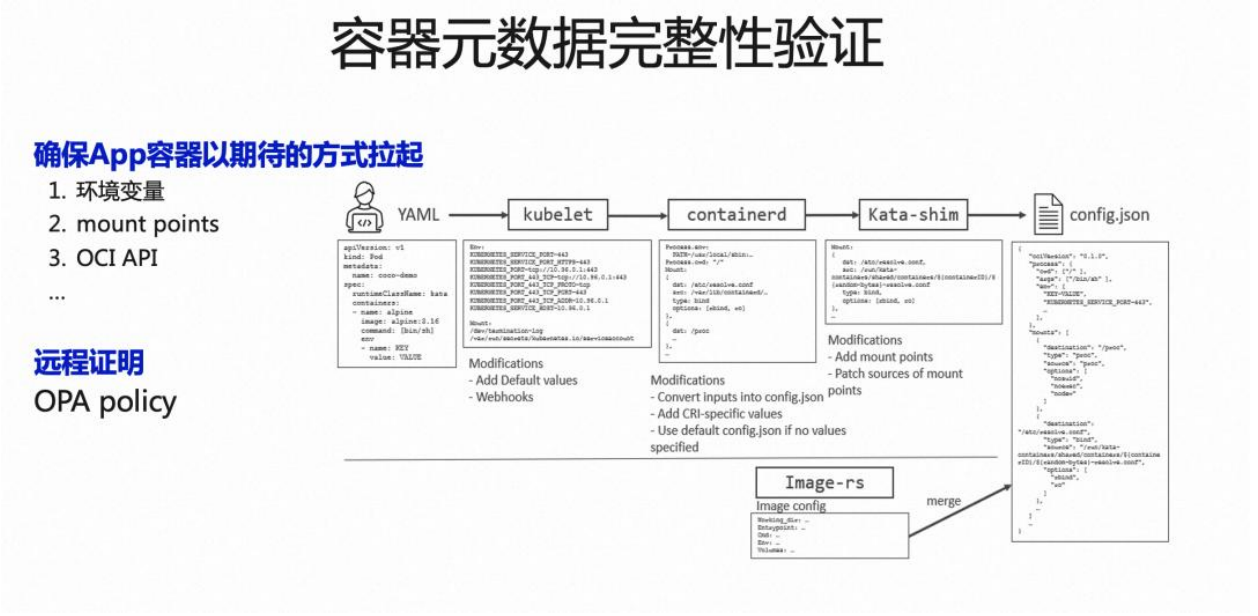
容器的运行时，共享内核的 runc 仍然占据主流的部署；随着安全需求的提升，独立内核的沙箱容器出现带来了更好的隔离性和更小的攻击面，降低了宿主机和云厂商的安全风险；随着对用户数据隐私要求的进一步提升，硬件加密的客户机内存和硬件生成的客户机密钥，结合远程证明，进一步保护了客户的隐私数据和代码，避免了硬件所有者窥探计算过程中的数据。

机密容器（Confidential Containers）是云原生基金会（CNCF）旗下的一个沙箱项目，它使用硬件信任执行环境（TEE）为容器化的工作负载提供机密性和完整性。机密容器两大设计原则就是易用和安全。从易用性角度，无缝对接 Kubernetes 和容器生态， 确保应用能够平滑迁移；从安全性角度，机密容器有着更严格的威胁模型，通过提供 Pod/VM 级 TCB，对 IT 运维人员和云厂商也可以做到计算过程的零信任。



结合 KMS，BYOK OSS，BYOK EBS，VPC，ACK，ACR 等云服务，端到端把零信任覆盖计算，存储和网络和配置， 对所有 POD 之外的输入做验证，所有 POD 里的非应用组件做度量，实现完整的应用可信和安全加固。

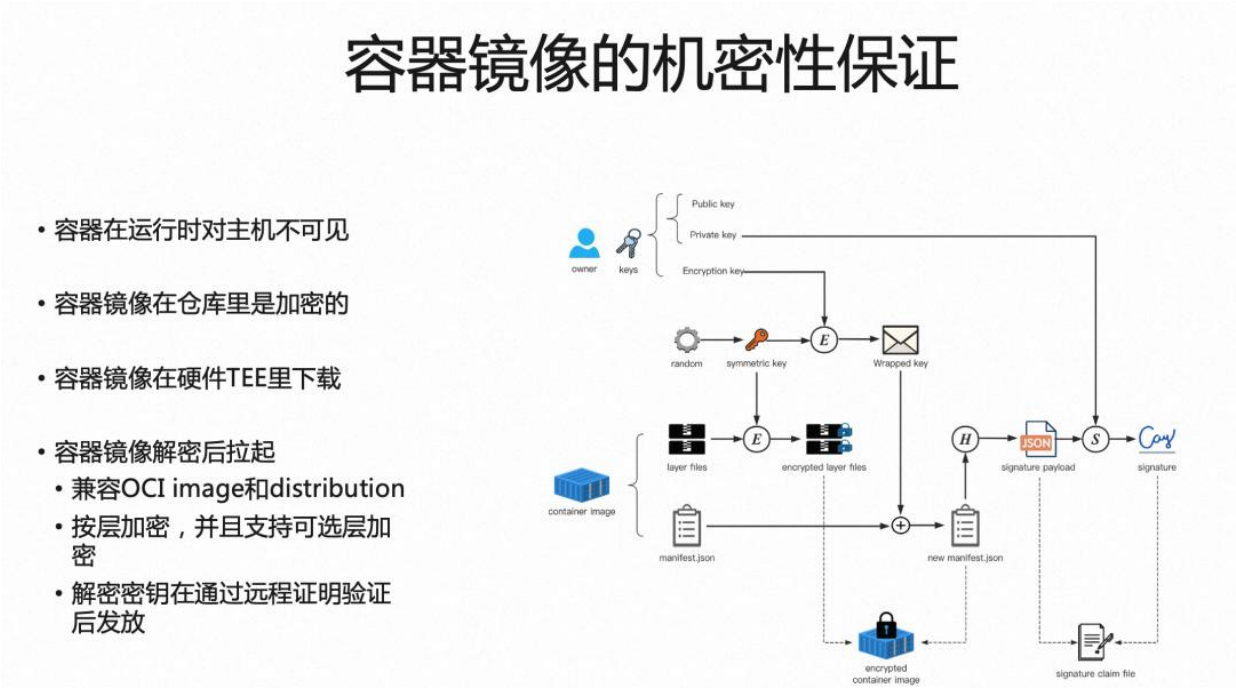
为了确保 App 容器运行在可信运行时环境 不被恶意篡改，安全容器参考架构提供了可度量的 guest rootfs，并利用 dm-verity 通过远程证明服务提供根文件系统的完整性，并且保证了启动性能。



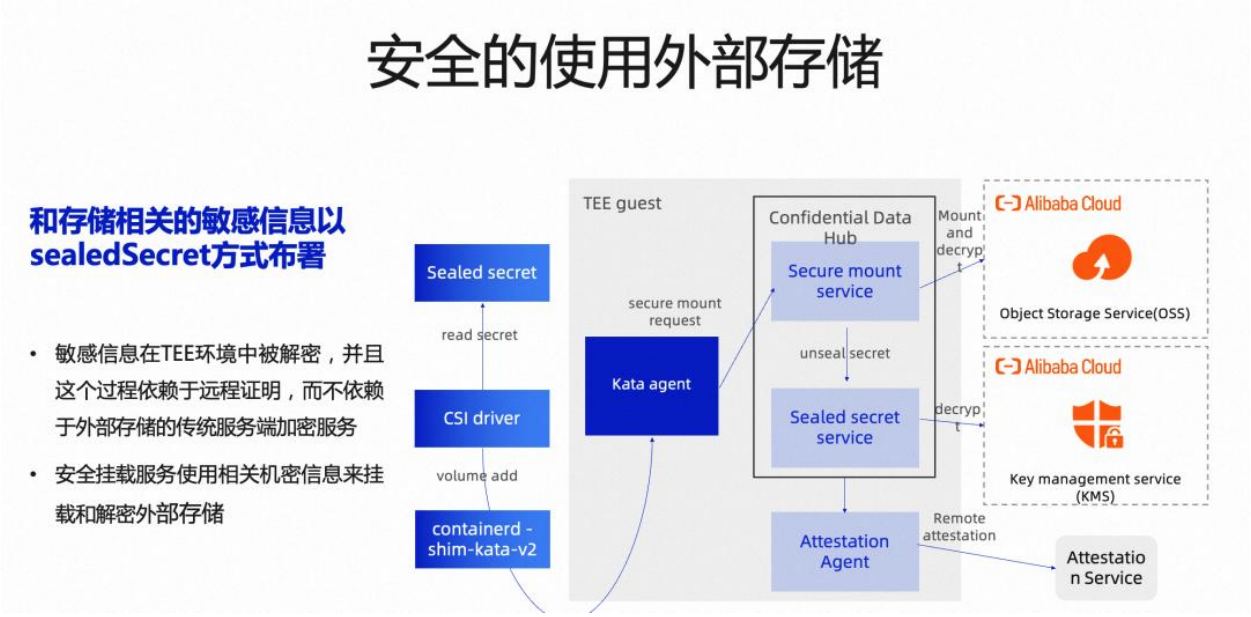
为了确保 App 容器以期待的方式拉起，需要通过 OPA 策略定义和度量容器的元数据，包括：

- 环境变量
- mount points
- OCI API

为了确保容器镜像的完整性，确保拉起过程中没有被恶意修改或者替换，使用镜像签名机制完成镜像校验，从 Key Broker Service 获得校验密钥，校验 Policy 并通过 CoSign/sigstore, GPG key 等方式校验镜像的完整性。



为了保护镜像的机密性和不可窥探性，容器在运行时需要对主机不可见，通过镜像加密保证容器镜像对服务提供商不可获取，容器镜像在硬件 TEE 里下载和解密对运维人员不可见，加密后的容器镜像支持 OCI 和 distribution，支持按层加密和可选层加密主要针对模型和私有代码的保护，解密密钥在通过远程证明验证后发放只对 TEE 可见。



安全的云上存储访问 ，存储相关的敏感信息以 sealedSecret 方式布署，敏感信息在 TEE 环境中被解密，并且这个过程依赖于远程证明，而不依赖于外部存储的传统服务端加密服务，安全挂载服务使用相关机密信息来挂载和解密外部存储。

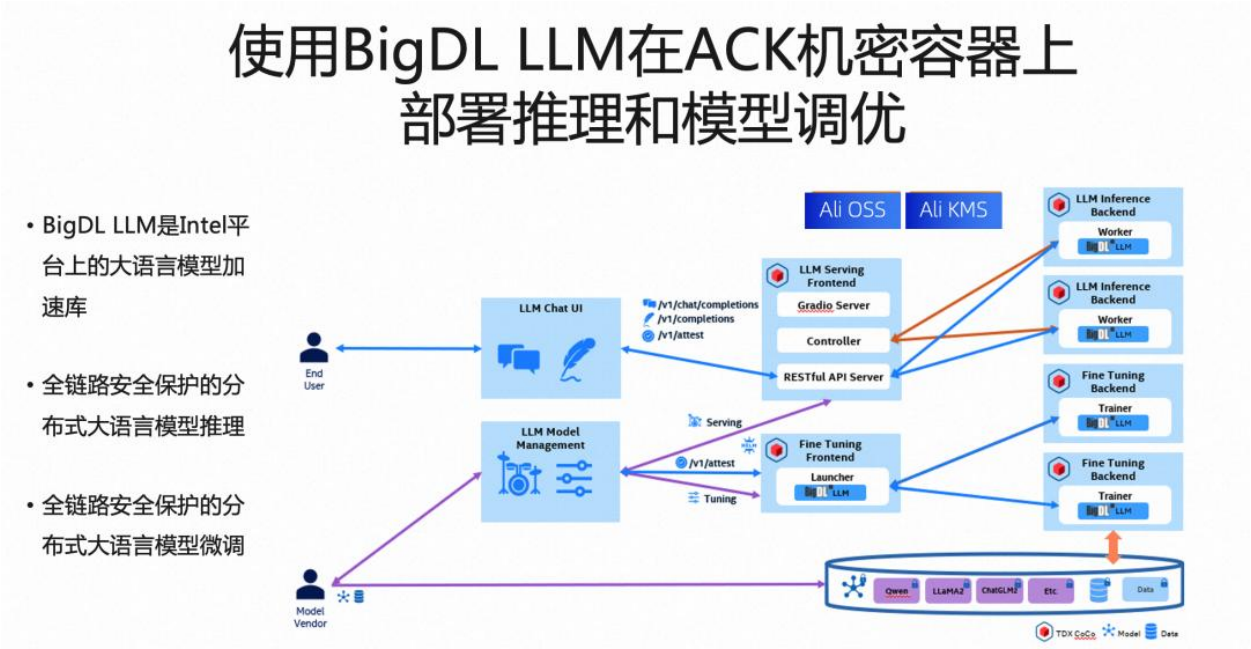
3. 基于机密容器构建可信 AI 应用

生成式人工智能（AIGC）等创新浪潮驱动了人工智能的新一轮增长，模型训练和模型推理成为云服务器的重要负载。如何在云上保护大数据分析和人工智能应用的数据安全和隐私，是数据科学家和云服务提供商共同面临的挑战。

为了应对这个问题， 阿里云容器服务推出基于英特尔® TDX 的机密容器服务解决方案，通过 ACK TDX 机密容器实现端到端零信任的数据和模型保护，基于第四代英特尔® 至强®平台的 高级矩阵扩展（AMX）的 INT8（推理）和 BFloat16（训练/推理）内置 AI 加速能

力， 可以实现高安全和高性价比的推理和微调服务：

- 安全可信 - 通过加密 AI 模型存储和加密的私有应用镜像，保障模型数据的机密性与完整性，实现可信 AI 模型推理和微调
- 高性价比 - 基于 Intel® AMX 指令集和 Intel® PyTorch 扩展，32 核可以实现秒级出图的推理能力
- 低损耗 - 加密计算 TDX 性能损耗控制在 3% 以下



使用 BigDL LLM 在 ACK 机密容器上部署推理和模型调优 ，BigDL LLM 是 Intel 平台上的大语言模型加速库，结合数据加密和阿里云存储和密钥服务，全链路安全保护的分布式大语言模型安全，也可以全链路安全保护的大语言模型微调数据的安全，通过 BigDL 和 ECS 8 代实例实现模型推理和微调的加速。

4. 容器是承载可信基础设施最好的平台服务

云计算提供存储（BYOK 加密）和网络（非对称传输加密）的可信基础设施，发展到今天可信的计算（TDX/SEV/CCA）。大胆的猜测，可信的存储、网络、计算的基础设施必定成

为云计算的标配，而容器正是承载可信基础设施最好的平台服务， 这也是我们为可信计算落地阿里云的初衷。

Koordinator 助力云原生应用性能提升——小红书混部技术实践

作者：张佐玮，阿里云技术专家、宋泽辉，小红书容器资源调度负责人

编者按：Koordinator 是一个开源项目，是基于阿里巴巴内部多年容器调度、混部实践经验孵化诞生，是行业首个生产可用、面向大规模场景的开源混部系统，致力于提升应用服务质量，优化资源使用效率。自 2022 年 4 月正式开源以来，吸引了业界众多优秀工程师的贡献参与和讨论。

小红书是 Koordinator 社区的活跃成员，自项目诞生初期就深度参与了一系列重要功能的演进。本文是基于 2023 云栖大会上关于 Koordinator 分享的实录，Koordinator 社区成员宋泽辉（小红书）、张佐玮（阿里云）为大家介绍了小红书混部技术实践以及 Koordinator 的近期规划。

1. 背景介绍

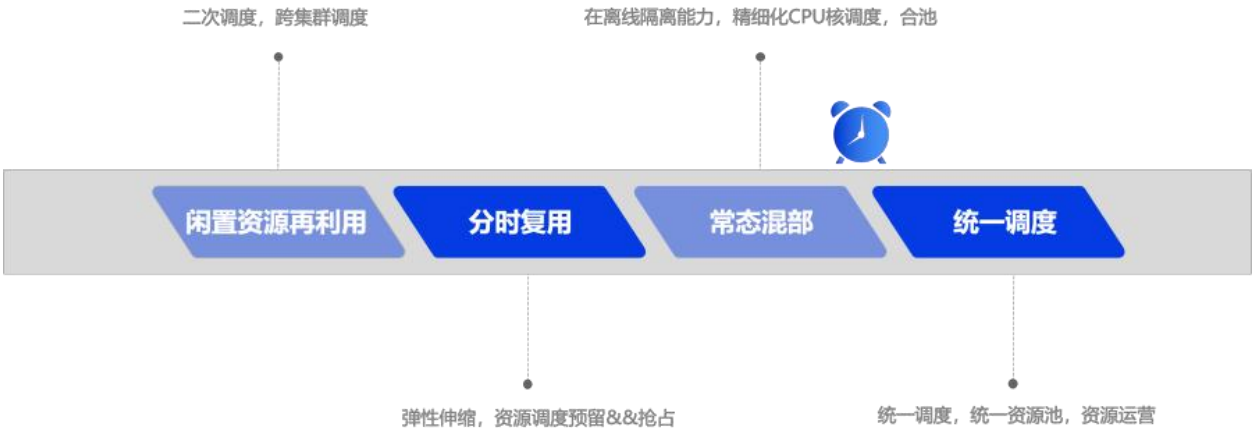
随着小红书业务的高速发展，各类在线，离线业务对于计算资源的需求也在快速增长。与此同时，部分在线集群天均利用率水位却维持在较低水平，造成这一现象的主要原因有以下几点：

- 在线服务资源使用量随着终端用户的使用习惯呈现稳定的潮汐现象，夜间 CPU 利用率极低，导致集群均值 CPU 利用率较低；
- 业务保有大量的独占资源池，资源池割裂产生大量的资源碎片，拉低 CPU 利用率；
- 业务为了稳定性考虑，会过量囤积资源，进一步拉低 CPU 利用率；

基于以上背景，为了帮助业务降低资源使用成本，提升集群 CPU 利用率，小红书容器团队从 2022 年开始，通过规模化落地混部技术来大幅提升集群资源效能，降低业务资源成本；

2. 技术演进

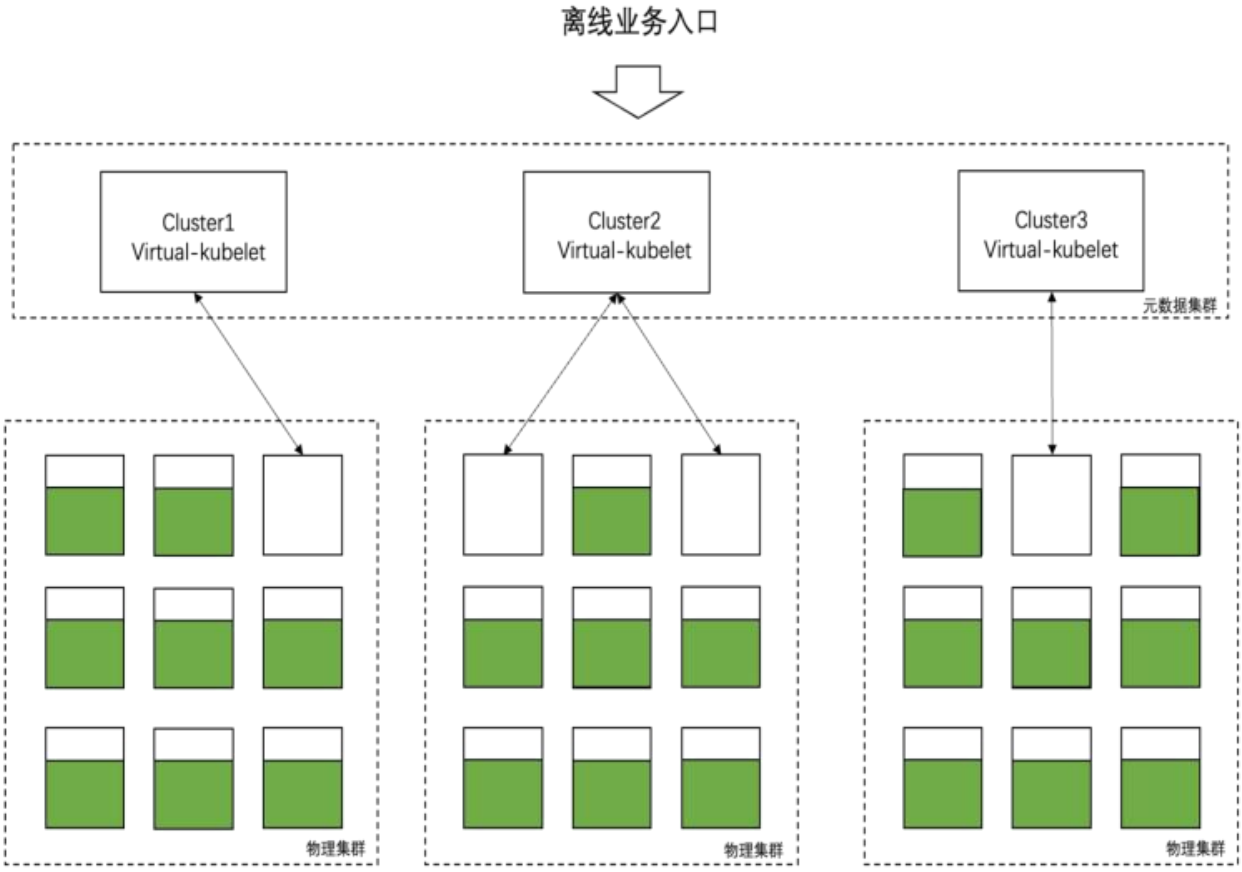
小红书混部技术演进分为以下四个阶段：



阶段一：闲置资源再利用

早期集群资源管理粗放，集群中存在大量业务独占资源池，因为资源碎片等因素存在大量低分配率的低效节点，散落在各个集群中的低效节点形成大量资源浪费。另一方面，部分基于 k8s 发布的转码类近线/离线场景，全天时段均存在大量计算资源需求。基于以上背景，容器平台通过技术手段将集群中的闲置资源收集起来，分配给转码类业务场景使用。

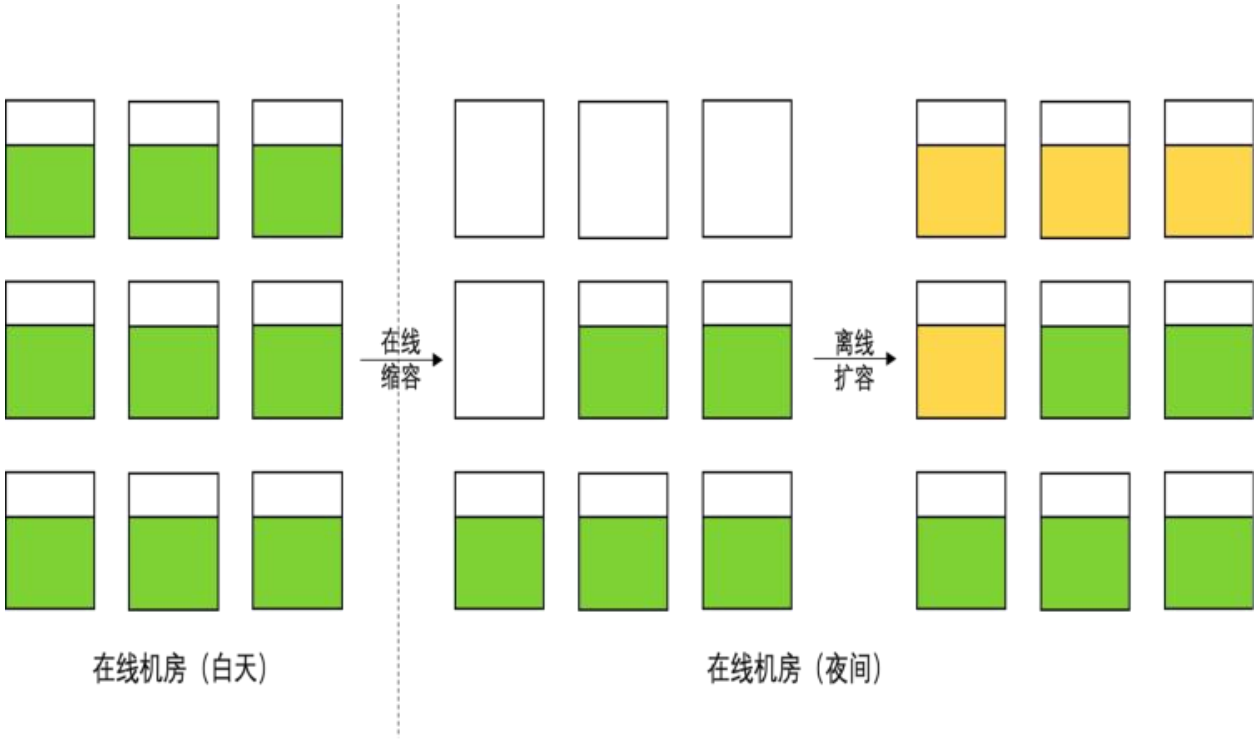
我们通过 virtual-kubelet 打通元数据集群与物理集群，将闲置资源汇聚起来，在元数据集群分配给转码类场景近线/离线计算服务。策略方面，二次调度器负责巡检集群所有节点，识别为低效节点后标记出来，virtual-kubelet 获取物理集群中的低效节点可用资源作为集群闲置资源二次分配给离线转码，同时二次调度器需要保证一旦在线服务有资源需求，将会立刻驱逐离线 pod 并归还资源。



阶段二：整机腾挪分时复用

搜推广等业务的独占资源池，CPU 利用率潮汐现象明显，夜间利用率极低，资源池中的单个节点往往也只部署一个大规格业务 Pod，基于以上背景，平台通过弹性能力（HPA），在凌晨业务低峰期按比例对在线业务缩容，腾挪空出整机，并将转码，训练等离线 pod 在该时段运行起来，起到利用率“填谷”的效果。

具体实施时，需要确保在线服务能在规定的时间内全部被拉起，为此，策略方面我们实现了离线提前退场，并通过调度器抢占机制兜底，确保在线服务在业务高峰期来临之前能被全量及时拉起。



阶段三：常态混部

为了降低资源碎片率，降低业务资源持有成本，平台持续推进业务大规模合池，将业务由独占池迁至平台托管的公共混部池，通过合池，资源超卖等技术手段，CPU 分配率得到有效提升，但依旧无法解决合并后的资源池夜间利用率较低等问题。

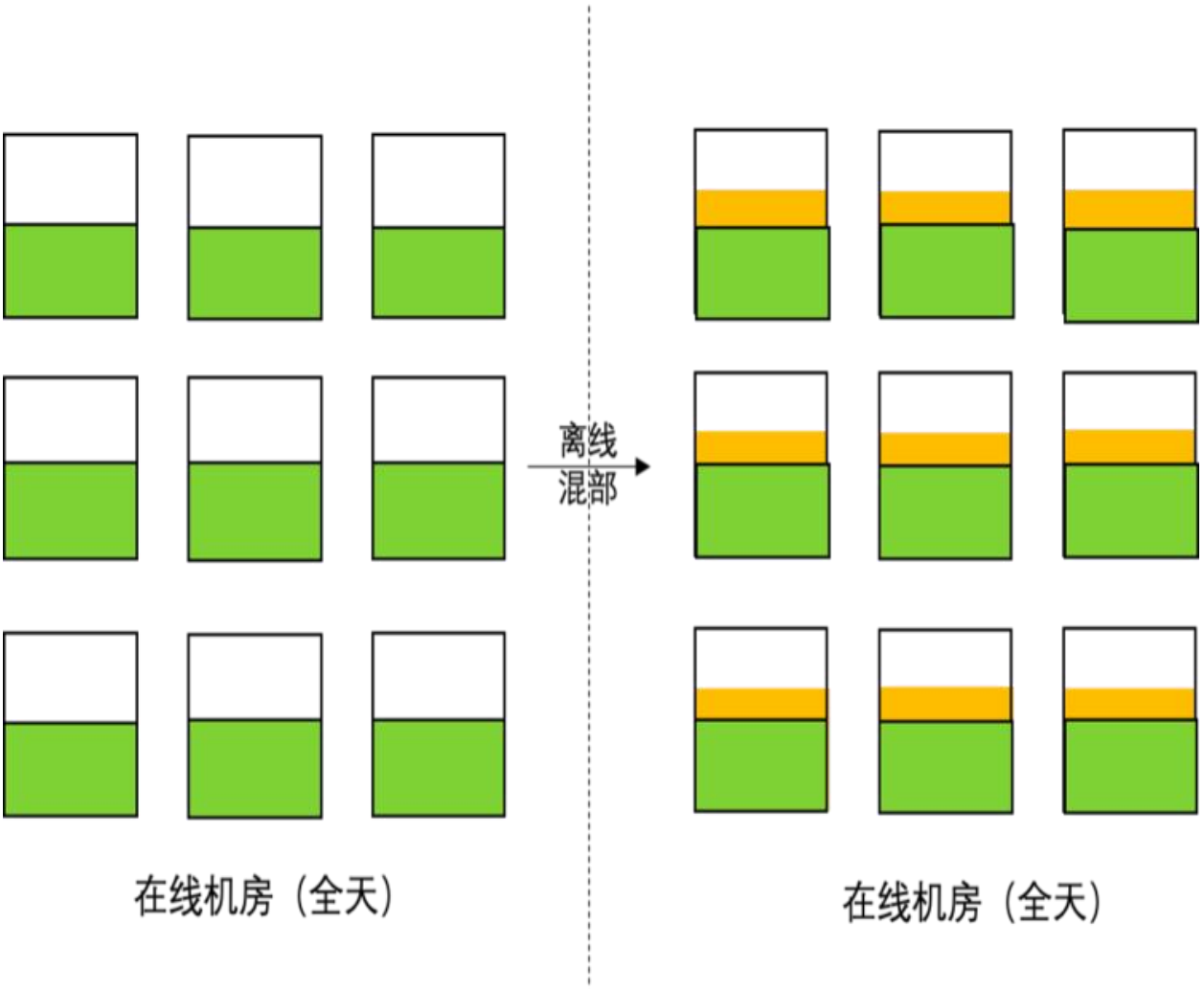
另一方面，合池后的复杂混部场景下，整机腾挪分时混部离线的调度策略很难再继续实施，平台需要通过建设更为细粒度的资源管理与调度能力来实现均值利用率提升的目标，具体包含以下几点：

- 调度侧：
    - 通过动态超卖技术获取可二次分配给离线的可用资源量，并抽象出离线资源视图让 k8s 调度器感知到，调度器调度离线负载到对应节点上，实现离线对节点利用率的“填谷”效果；
    - 通过负载调度，尽可能避免在线服务被调度到高负载机器，让集群中节点负载更加均衡；

- 通过二次调度，驱逐负载热点机器上的高利用率服务，使得集群负载处于动态均衡状态；

- 单机侧：
  - 支持 Qos(Quality of service)保障策略，根据服务的 Qos 等级提供差异化的运行时资源保障能力；
  - 支持干扰检测、离线驱逐等能力，当离线对在线敏感服务产生干扰时，第一时间驱逐离线；

通过以上技术手段，可以有效保障服务混部时的稳定性，从而常态化的让在线离线工作负载混跑在节点上，实现利用率填谷效果的最大化。



3. 架构设计与实现

小红书容器资源调度架构设计如图所示：

小红书各类业务场景通过各类发布平台、任务平台提交后，通过上层负载编排能力，以 pod 形式下发到统一调度系统。统一调度系统基于不同的调度需求，对在线服务提供强保障的资源交付能力，差异化的 Qos 保障能力，对离线服务提供最小资源需求的保障能力和极致的弹性能力。

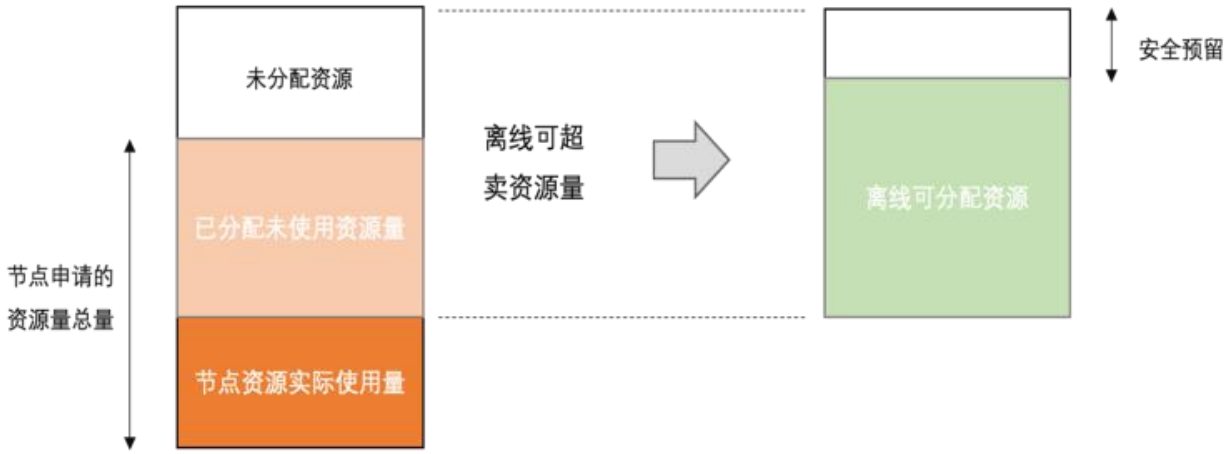
- 调度侧：
  - 离线调度：coscheduling；
  - 二次调度：热点驱逐，碎片整理；

- 负载调度：基于 CPU 水位；
- 资源视图：模拟调度；

- 单机侧：
  - 压制策略：bvt 压制，内存驱逐；
  - Qos 保障：绑核，超线程干扰抑制等；
  - Batch 资源上报：batch 可用资源计算，上报；
  - 指标采集(from kernel)：psi，sched info 等；
  - 干扰检测：基于 cpi，psi，业务指标的干扰检测；

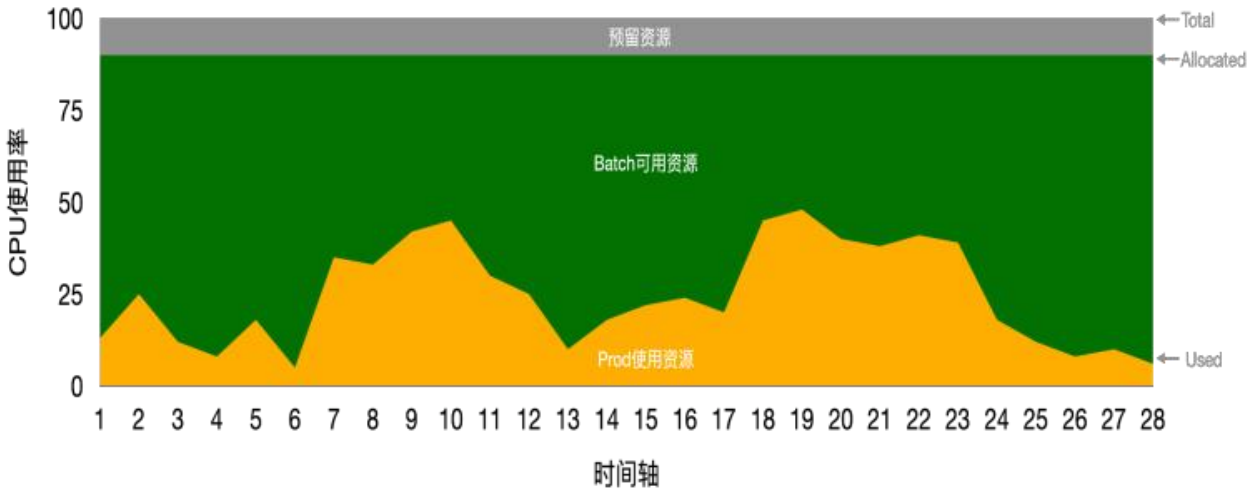
1）离线调度资源视图

离线服务资源调度的基本原理是基于在线服务负载感知能力的动态超卖，具体实现是将节点空闲资源二次分配给离线业务：

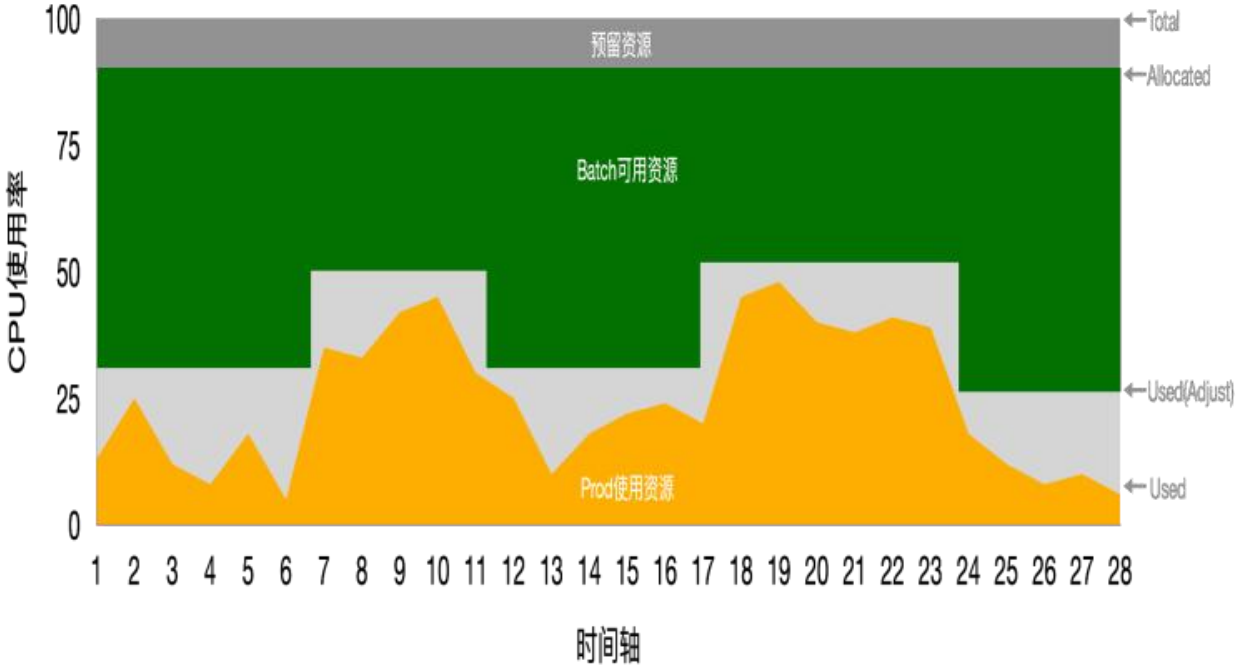


其中离线可用资源为节点上的空闲资源（包含未分配资源和已分配未使用资源之和），扣除安全预留资源之后剩余资源，离线可用资源计算公式如下：

将计算出的离线可用资源量按照时间分布后如图所示（图中绿色部分）：



实际落地过程中，为了避免离线可用资源随在线服务资源使用波动而大幅波动，从而影响离线资源质量和离线服务运行稳定性，通过资源画像对上述公式中的在线服务实际使用量数据进一步处理，去除数据噪点，最终计算出一个相对稳定的离线可用资源量（图中绿色部分），如图所示：



2）混部 QoS 保障策略

QoS 分级

按照业务对于服务质量（QoS: Quality of service）的需求，我们将小红书的业务类型简单的划分为三个 QoS 级别，如下表所示：

Qos 等级	说明	业务场景
latency-sensitive	最高 Qos 保障等级，延迟极为敏感服务	搜推广延迟极为敏感场景
mid	默认 Qos 保障等级，容忍部分干扰延迟	网关，java 微服务
batch	最低 Qos 保障等级，延迟不敏感，资源随时可能被抢	转码，spark，flink，训练等计算场景

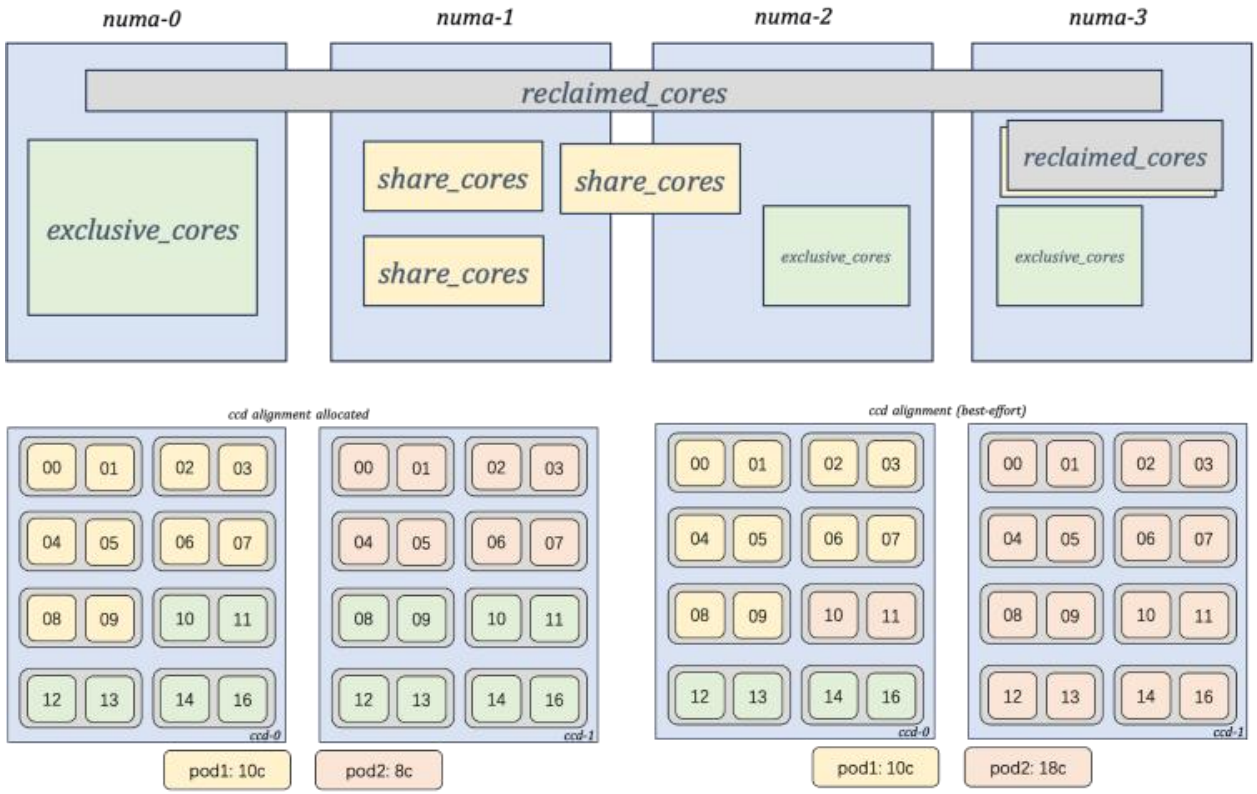
QoS 保障

根据服务的 QoS 需求，节点侧会做 Pod 粒度的分级资源保障，实现各个资源维度差异化 QoS 保障策略，具体的保障参数如下；

资源	特性	latency-sensitive	mid	batch
CPU	cpu burst	enable	enable	disable
	调度优先级	最高	默认	低
	绑核	share(默认)	share(默认)	reclaimed

	numa	强保证	prefer（默认）	none
	L3 cache	100%	100%（默认）	30%（默认）
	内存带宽	100%	100%（默认）	30%（默认）
内存	OOM 优先级	最低	默认	最高
	内存回收水线	调高	默认	调低

在 CPU 核调度层面，分别设置了三种绑核类型，并设计了一套精细化 CPU 核编排策略，分配示意图如下：



三种绑核类型分别为：

- exclusive（不推荐）
  - 特点：绑定 cpuset 调度域，CCD 感知，numa 绑定，独占排他
  - 场景：极为敏感的搜推广大规格延迟敏感服务
- share
  - 特点：绑定 cpuset 调度域，CCD 感知，numa （可选）绑定，share/exlusive 排他，可与 none 类型业务共享
  - 场景：容忍部分干扰的 Java 微服务，应用网关，web 服务
- reclaimed
  - 特点：无 cpuset 绑定，可能与非 exlusive 绑核模式业务共享核，核的分配完全交由内核，CPU 资源并非 100%能得到满足
  - 场景：batch 类离线服务，部分对延迟无要求的计算服务

3）离线驱逐

极端场景下，如整机内存使用率较高，有触发 OOM 风险，或者离线业务 CPU 长期得不到满足，单机侧支持按照离线服务内部定义的优先级配置，资源用量，运行时长等多维度综合算分排序后按序驱逐；

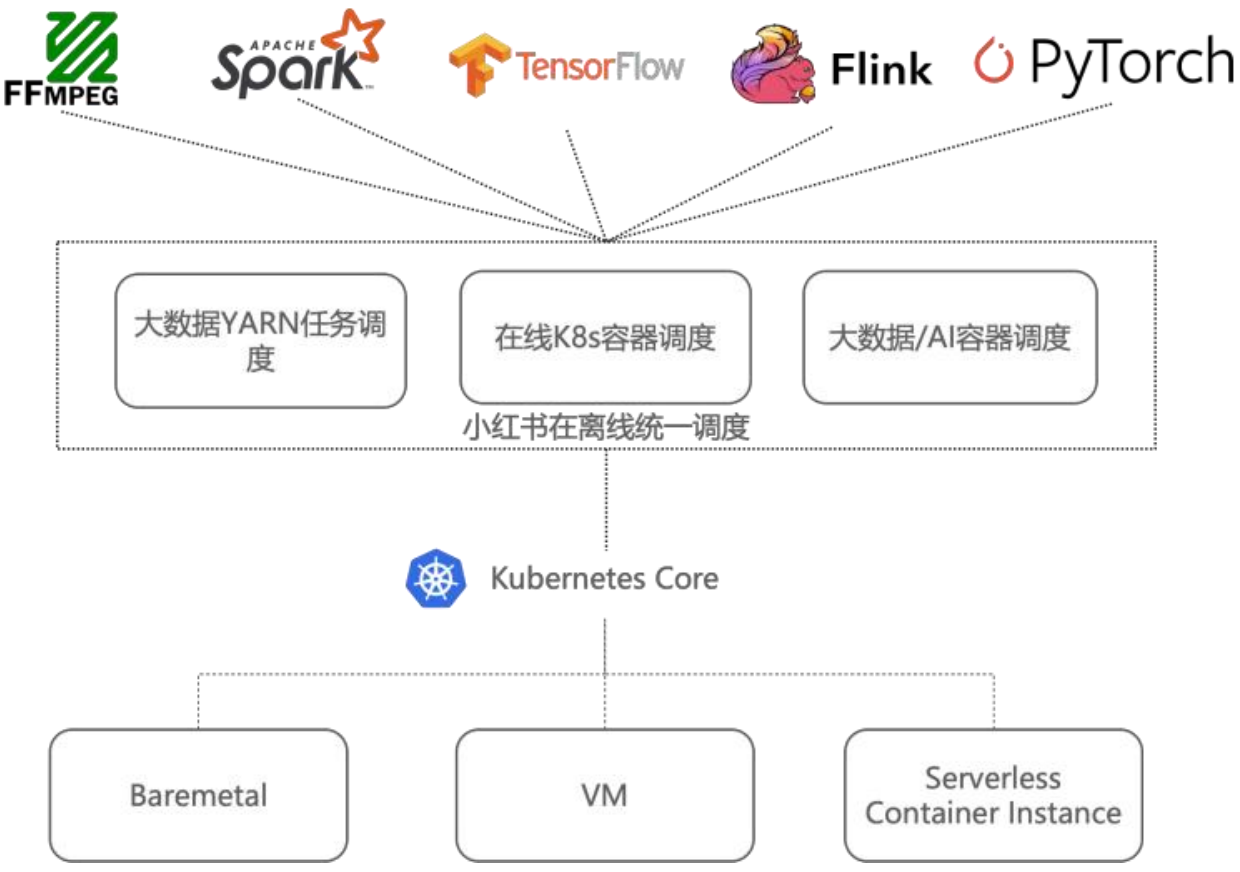
4. 离线业务场景

小红书作为一个数亿用户的内容社区，其离线业务场景丰富多样，其中包含大量视频类，图片类转码场景，搜推，cv/nlp 算法推理训练，算法特征生产，数仓查询等离线场景，具体来讲，包含以下业务类型：

- 近离线转码场景（已容器化）

- Flink 流式/批式计算（已容器化）
- Spark 批式计算 （容器化，on yarn）
- cv/nlp 算法回扫场景（已容器化）
- 训练场景 （已容器化）

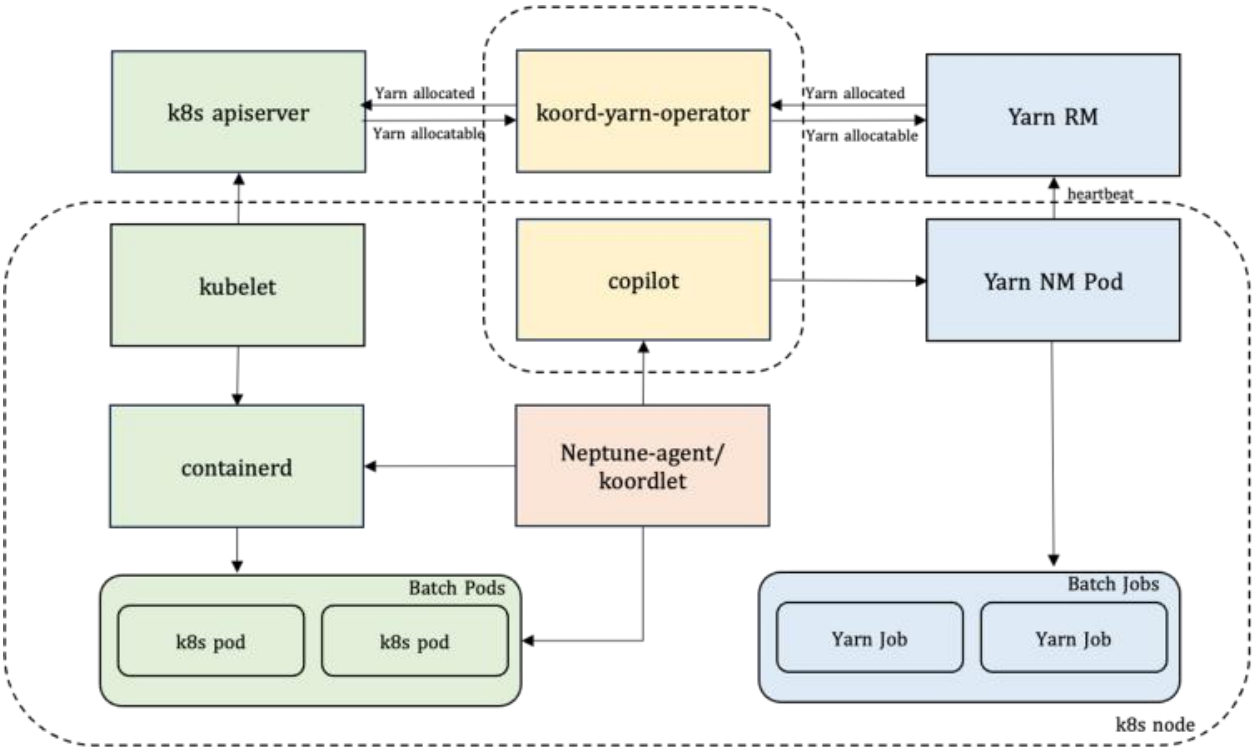
通过提供以 K8s 为底座的在离线统一调度能力，将这些离线业务与在线服务混合部署在统一计算资源池内，为在线服务提供差异化的资源质量保障，为离线服务提供海量的低层本算力，实现资源效能的提升。



1） K8s 与 YARN 混部方案

小红书内部商业化，社区搜索等业务存在大量的算法类 spark 任务因为离线集群资源紧张导致任务堆积，不能得到及时处理，同时在线集群在业务低峰时段资源使用率较低；另一

方面，相当占比的 spark 任务资源调度仍旧运行在 Yarn 调度器上，在这样的背景下，为了降低业务迁移成本，方案选型方面，我们选择与 kooridinator 社区合作，采用 Yarn on k8s 混部方案来快速落地 Spark 离线场景混部，具体方案如图所示：



其中容器化的在线、离线工作负载通过 k8s 链路发布到在线集群内，Spark 作业通过 Yarn ResourceManager 调度到具体节点，并由节点上的 Nodemanager 组件拉起。其中 Nodemanager 通过容器的方式部署在在线 k8s 集群内，除此之外，还涉及到以下组件：

- 调度侧

koord-yarn-operator ： 支持 k8s 与 yarn 调度器资源视图双向同步；

- 节点侧
  - copilot：NodeManager 操作代理，提供 Yarn Task 管控接口；
  - Neptune-agent/koordlet：离线资源上报，节点离线 Pod/task 管理，冲突解决，驱逐，压制策略；

- 支持 K8s 与 YARN 混部的核心能力目前已经在社区研发完成，将于 11 月下旬，在 Koordinator 1.4 版本进行发布。

2）多调度器资源同步

K8s 调度器与 YARN 调度器之间原本独立且相互不感知，为了共享分配在线集群节点上的总可用离线资源，需要通过 koord-yarn-operator 组件来做两个调度器之间的资源双向同步和协调，并实现两个同步链路：

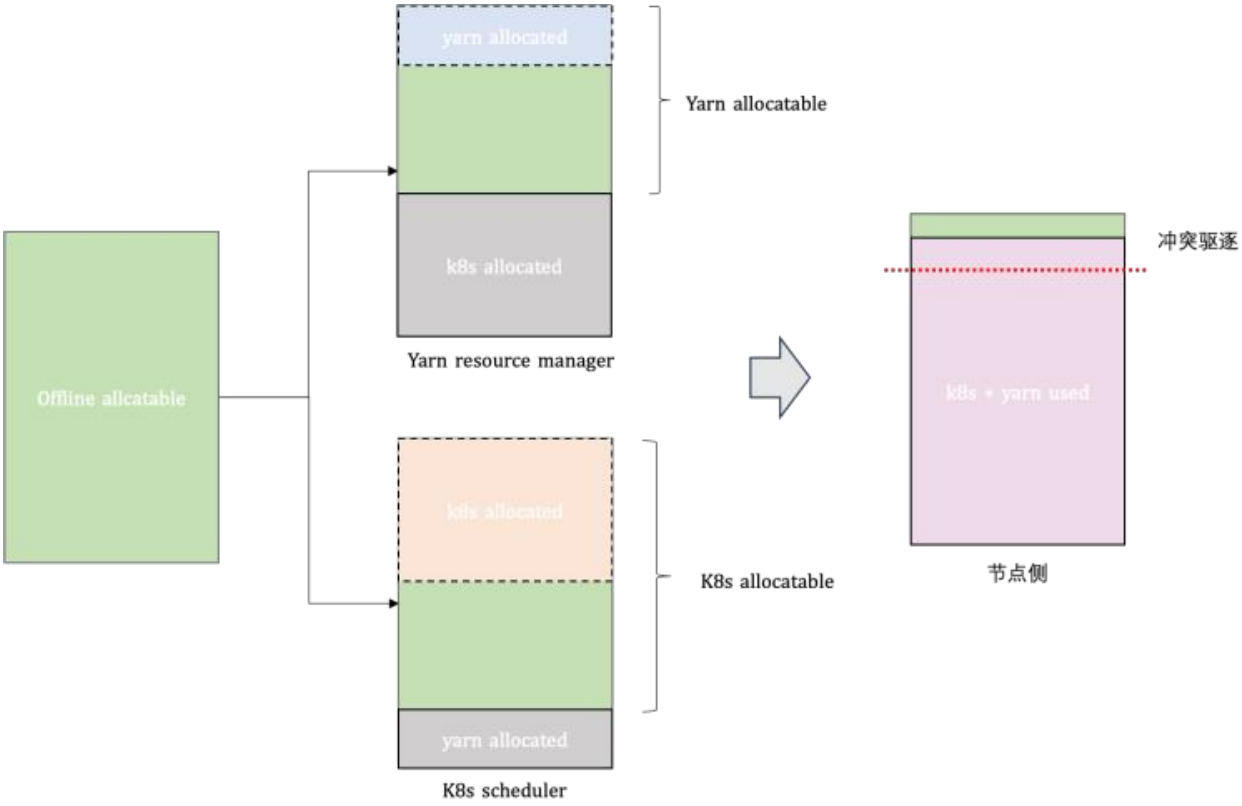
1K8s->YARN 调度器资源同步链路，负责同步 Yarn 视角离线资源总量，其中 YARN 离线资源总量计算如下：

YARN 离线资源总量=离线总可用量-K8s 侧节点已分配

1YARN->K8s 调度器资源同步链路，负责同步 YARN 已分配资源量，其中 k8s 离线资源总量计算如下：

k8s 离线资源总量=离线总可用量-YARN 侧节点已分配

基于各自节点离线资源视图，两个调度器分别做出调度决策，调度 K8s 离线 Pod 与 YARN Task 到节点上，由于同步过程不适合加锁，可能会出现资源被过量分配的问题：



具体解决措施是在单机侧增加了仲裁逻辑，当节点已分配离线服务资源量长期超过节点可用离线资源，且离线使用率持续较高，存在离线服务得不到资源被饿死的可能，单机侧则会根据离线服务的优先级，资源占用量，运行时长等因素综合算分并按序驱逐。

3）阿里云 EMR 产品化支持

与此同时，阿里云 EMR 团队在产品层面提供了混部功能的开发支持，在兼容 EMR 原有日志，监控，运维逻辑的基础上，支持了 k8s 集群弹性扩缩容 NodeManager Pod 的能力。

5. 落地收益

截止目前，小红书混部能力覆盖数十万台机器规模，覆盖算力规模数百万核，支持数万规模在线、离线场景服务的资源调度。通过大规模容器混部的持续推进，小红书在资源成本效能等方面都取得了显著收益，具体包含以下两方面：

CPU 利用率

在保证在线服务服务质量的前提下，在线混部集群天均 CPU 利用率提升至 45% 以上，部分集群天均 CPU 利用率可稳定提升至 55%。

通过在离线混部等技术手段，在线集群 CPU 利用率提升 8%-15% 不等，部分存储集群 CPU 利用率提升可达 20% 以上。

资源成本

在保证离线业务稳定性的前提下，为小红书各类离线场景提供数百万核时的低成本算力。混部集群 CPU 分配率提升至 125% 以上，相较于独占资源池，资源碎片率明显下降。

6. 社区共建历程

小红书是早期参与 Koordinator 社区的公司之一，2022 年 4 月，Koordinator 正式开源，同年 6 月，小红书内部启动了在离线混部项目，开始参与 Koordinator 方案设计与代码提

交。2022 年 8 月，小红书与社区共建了 runtime-proxy 组件，并在内部场景落地。2023 年 4 月，小红书在社区主导启动了 YARN 与 K8s 混部项目，2023 年 8 月，该方案在小红书内规模化落地。

截止目前，依托 Koorindiator 的助力，小红书的混部已经覆盖公司数万台节点，提供数十万核离线资源，整体混部集群的利用率提升至 45%以上。取得了不错的落地效果。

7. 总结与展望

在小红书近一年多混部技术探索过程中，我们在资源效能提升方面积累了较为丰富的落地经验，并取得了不错的提升效果，随着公司业务规模逐步增长，场景愈发复杂，我们将会面临诸多新的技术挑战。下个阶段我们的目标是建设面向混合云架构的统一资源调度能力，具体工作将围绕以下三方面展开：

- 混合工作负载调度能力支持：包括大数据，AI 在内的任务型工作负载调度能力建设，满足小红书所有业务场景的资源调度功能，性能需求；
- 资源效能进一步提升：面向混合云架构，推进更大规模的资源合池，推进 quota 化资源交付，通过更加激进的弹性，混部，超卖等技术手段，实现集群资源利用率的进一步提升，资源成本的大幅下降；
- 更高服务质量保障能力：在更为激进的 CPU 利用率目标背景下，通过建设 Qos 感知调度能力，干扰检测能力，依托安全容器等技术手段，解决深水区混部中可能遇到的各类混部干扰问题；

8. Koordinator 社区近期规划

再接下来的几个版本中，Koordinator 将在以下几个方面进行重点投入：

- 调度器性能优化：支持等价类调度，通过合并 request 相同的 pod，避免 filter、score 等调度过程的重复计算。

- Network QoS：网络维度容器服务质量，保障高优先级带宽，设计 request/limit 模型，保障最低带宽需求。

- 大数据负载：支持 Gang 调度原子抢占，按分组整体抢占 Pod；面向 Hadoop YARN 任务的 QoS 策略适配。

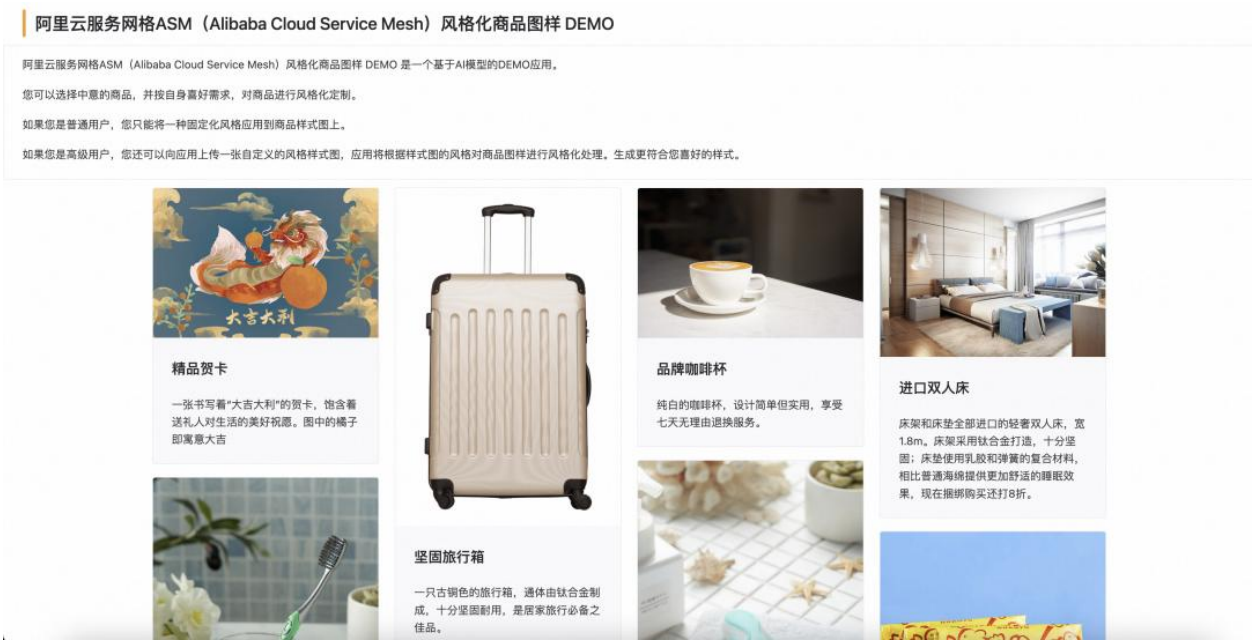
- 资源干扰检测：基于底层指标、感知容器资源竞争情况，识别异常 Pod，消除干扰并反馈调度链路。

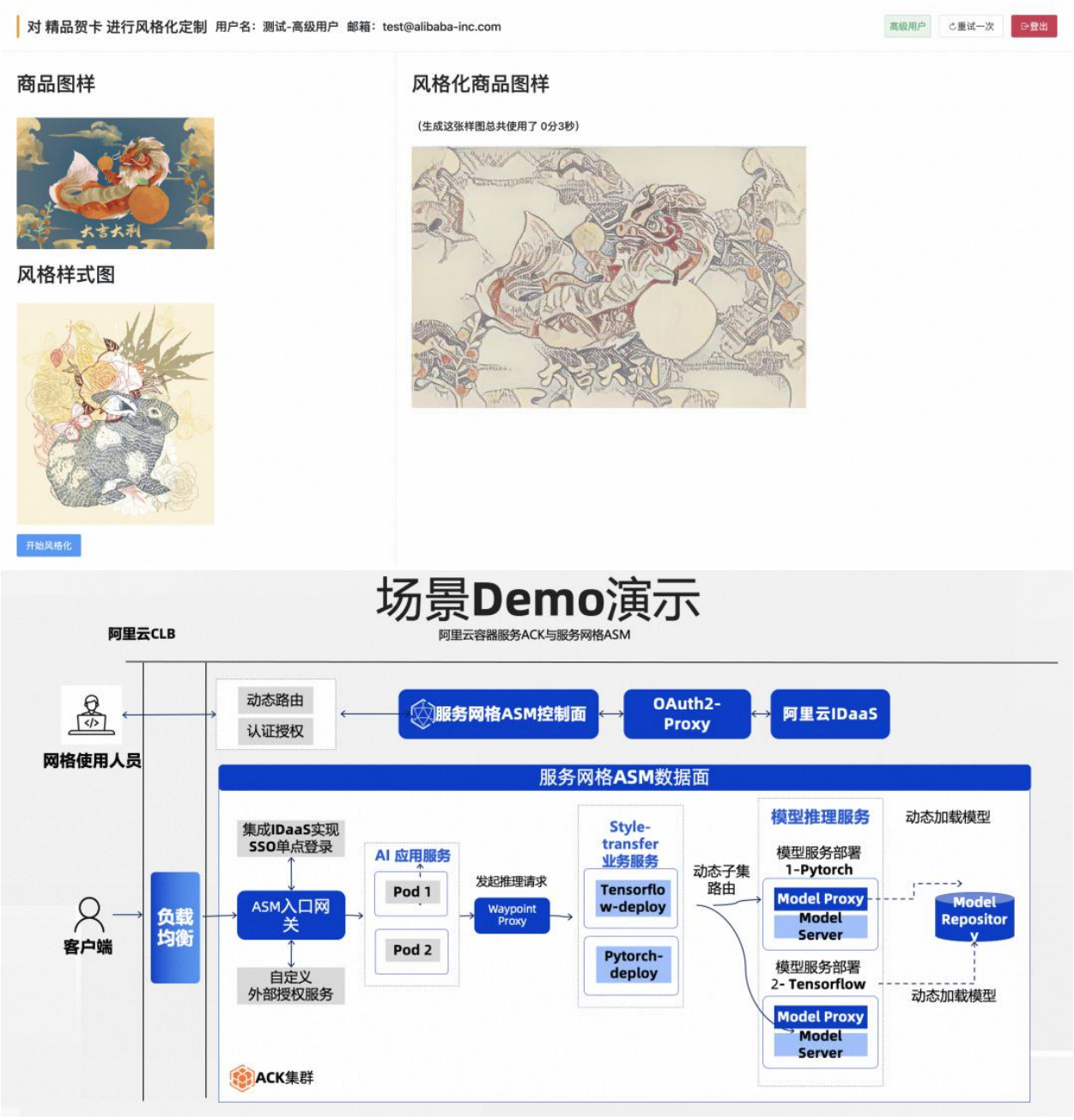
可以使用钉钉扫描下方二维码或搜索群号 33383887 加入 Koordinator 社区钉钉群：

轻松搭建基于服务网格的 AI 应用，然后开始玩

作者：尹航，阿里云研发工程师

在 2023 年的云栖大会中，阿里云服务网格 ASM 推出了《两全其美：Sidecarless 与 Sidecar 模式融合的服务网格新形态》主题演讲，并在演讲中展示了一个基于服务网格 ASM 各项能力构建的 DEMO AI 应用。该应用集中展示了 ASM 在模型服务、请求处理、请求路由和安全中心集成单点登录等各项能力，且这些能力还完全是以 Sidecarless 的形态来实现的。





看完我们的演示，您也许也会想尝试一下，从零开始构建这样的一个应用来玩玩吧！当然！我们向您保证，我们能搭出来的东西，您一定也能搭出来。本文就是这样一篇给各位的入门指引，我们这就开始吧！

1. 从零开始搭建一个基于服务网格 ASM 的 AI 应用

前提条件

一个 ACK 集群、一个 ASM 实例以及相关的 istioctl 等工具是一切的根基，我们先来准备一些实验环境。

- 已创建 ASM 实例，且实例版本在 1.18.0.131 及以上。具体操作，请参见创建 ASM 实例[1]。在创建服务网格页面配置数据面模式时，选中启用 Ambient Mesh 模式。
- 已创建 Kubernetes 集群，且满足 Kubernetes 集群及配置要求[2]。关于创建集群的具体操作，请参见创建 Kubernetes 专有版集群[3]或创建 Kubernetes 托管版集群[4]。
- 已添加集群到 ASM 实例。具体操作，请参见添加集群到 ASM 实例[5]。
- 已按照实际操作系统及平台，下载 Istioctl 服务网格调试工具。详细信息，请参见 Istio[6]。

搭建模型推理服务

1）开启 ASM 的多模型推理服务生态集成能力

对于一个基于 AI 模型推理的应用服务来说，将训练好的模型快速转化为弹性、灵活的模型推理服务无疑是工作的重心之一。

作为应用感知的下一代云原生基础设施，服务网格 ASM 也通过其丰富的生态集成能力、集成了云原生推理服务框架 KServe（参考 ASM 集成云原生推理服务框架 KServe[7]）、为 AI 模型推理的服务化提供了一站式解决方案。

在服务网格 ASM 的最新版本中，我们 alpha 阶段地引入了模型推理服务集成的多模型服务框架（modelmesh）。在全新的 modelmesh 服务框架之内，不同的模型、其推理将交给多个运行时工作负载来完成。每个运行时支持不同的模型格式；并且可以同时提供多个模型的推理服务。当我们使用 InferenceService 资源定义一个模型后，模型文件将根据模型的格式、动态地加载到对应的运行时工作负载之中。一个运行时可以同时提供多

个模型的推理服务。

我们可以通过以下步骤来集成多模型推理服务框架 modelmesh：

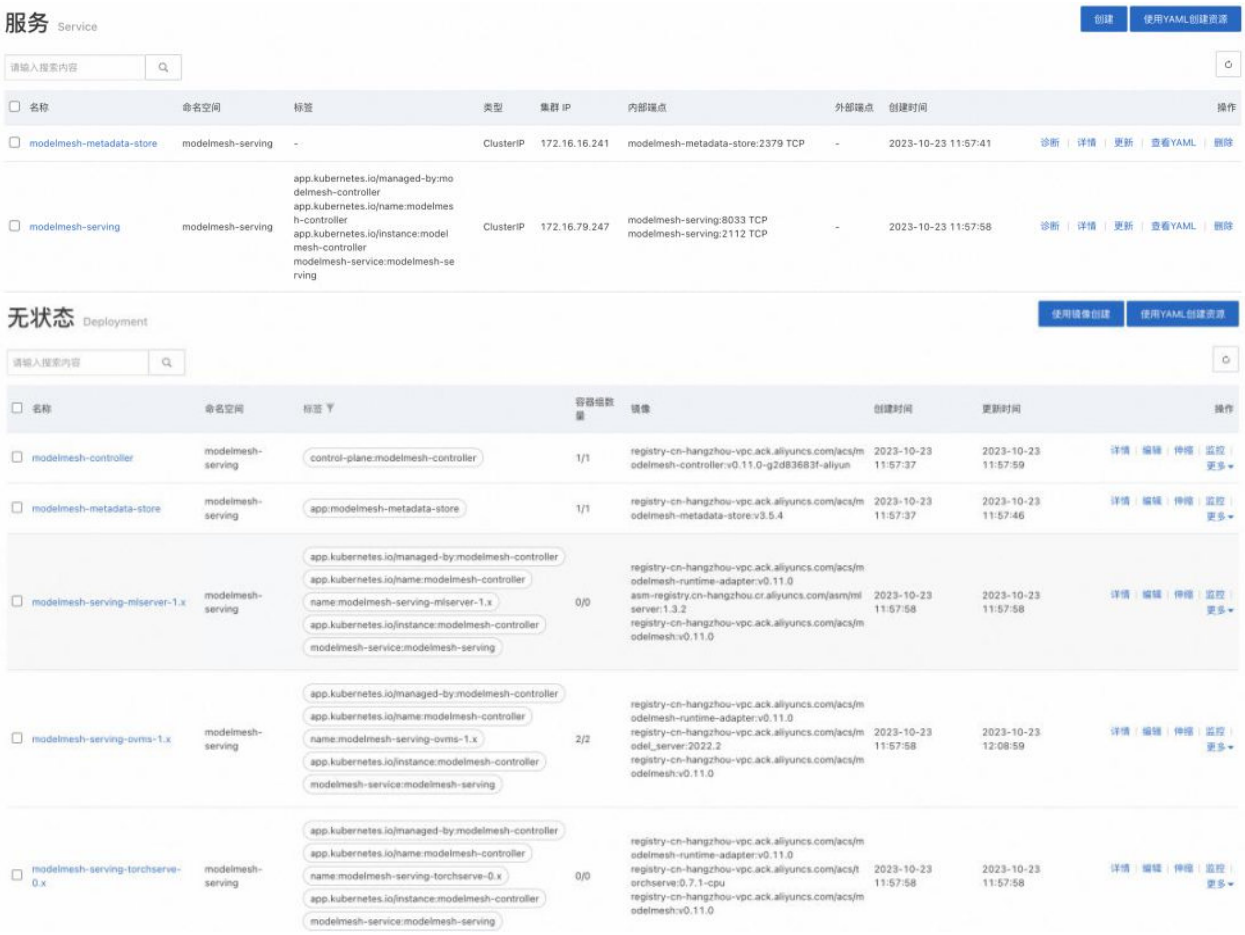
- 在 ASM 实例中创建一个名为 modelmesh-serving 的全局命名空间（参考管理全局命名空间[8]）
- 要使用这个能力，我们首先使用 kubectl 连接到 ASM 实例（参考通过控制面 kubectl 访问 Istio 资源[9]）
- 使用以下这个文件，创建 asmkserveconfig.yaml

```
apiVersion: istio.alibabacloud.com/v1beta1
kind: ASMKServeConfig
metadata:
  name: default
spec:
  enabled: true
  multiModel: true
  tag: v0.11.0
```

- 使用 kubectl 执行以下命令，打开模型推理服务框架集成

```
kubectl apply -f asmkserveconfig.yaml
```

执行完此步骤后，我们可以看到 ACK 集群中出现一个 modelmesh-serving 命名空间，内部包含有模型推理 Servicemodelmesh-serving、以及提供服务的各种运行时工作负载，这就代表模型推理服务已经就绪。



2）准备模型文件，声明推理服务

模型推理服务框架就绪后，接下来我们需要准备好训练的模型文件，并将模型加载到运行时工作负载中，成为可以对外暴露的推理服务。

a. 准备模型文件

机器学习模型在经过训练后，可以通过各种序列化方式被保存下来（例如：saved_model、pkl 等），模型推理服务器可以加载并利用这些模型文件对外提供训练好的机器学习模型的推理服务。

在本 DEMO 应用中，我们也需要准备这样的模型文件。事实上，我们准备了两个训练好的模型。这两个模型分别基于 tensorflow 与 pytorch，其中 pytorch 模型生成的图片风格固定，而 tensorflow 模型可以抽取图片风格，进行不同的风格化处理。

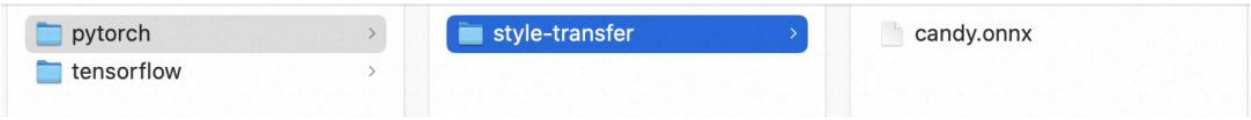
模型的获取也非常简单，不需要大家去自己训练了。我们只需要通过 Tensorflow 和 Pytorch 的官方渠道即可获取了。

- TensorFlow 模型可通过 Tensorflow Hub 获取，访问这里来下载：https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2
- 至于 Pytorch 模型，我们在本例中使用了官方 DEMO 例子中的模型，并将其转换成了 ONNX 格式。我们可以参考这个教程来下载并转换模型文件：https://pytorch.org/tutorials/advanced/ONNXLive.html（注意：在转换成 ONNX 模型的一步，我们是使用了 512*512 的图片作为输入，注意输入图片尺寸，这个对 ONNX 格式的模型很重要）。demo 中提供四种固定风格的模型，我们可以任选一款，在我们的 demo 中选择了 candy 模型。

下载到本地后，我们随便找个路径作为根目录，新建一个 tensorflow 文件夹和一个 pytorch 文件夹，分别保存两个模型的文件。我们将两个模型的模型文件保存成如下的文件夹结构，方便后续操作。

Tensorflow 模型大概长这样：

Pytorch 模型则是这样的：



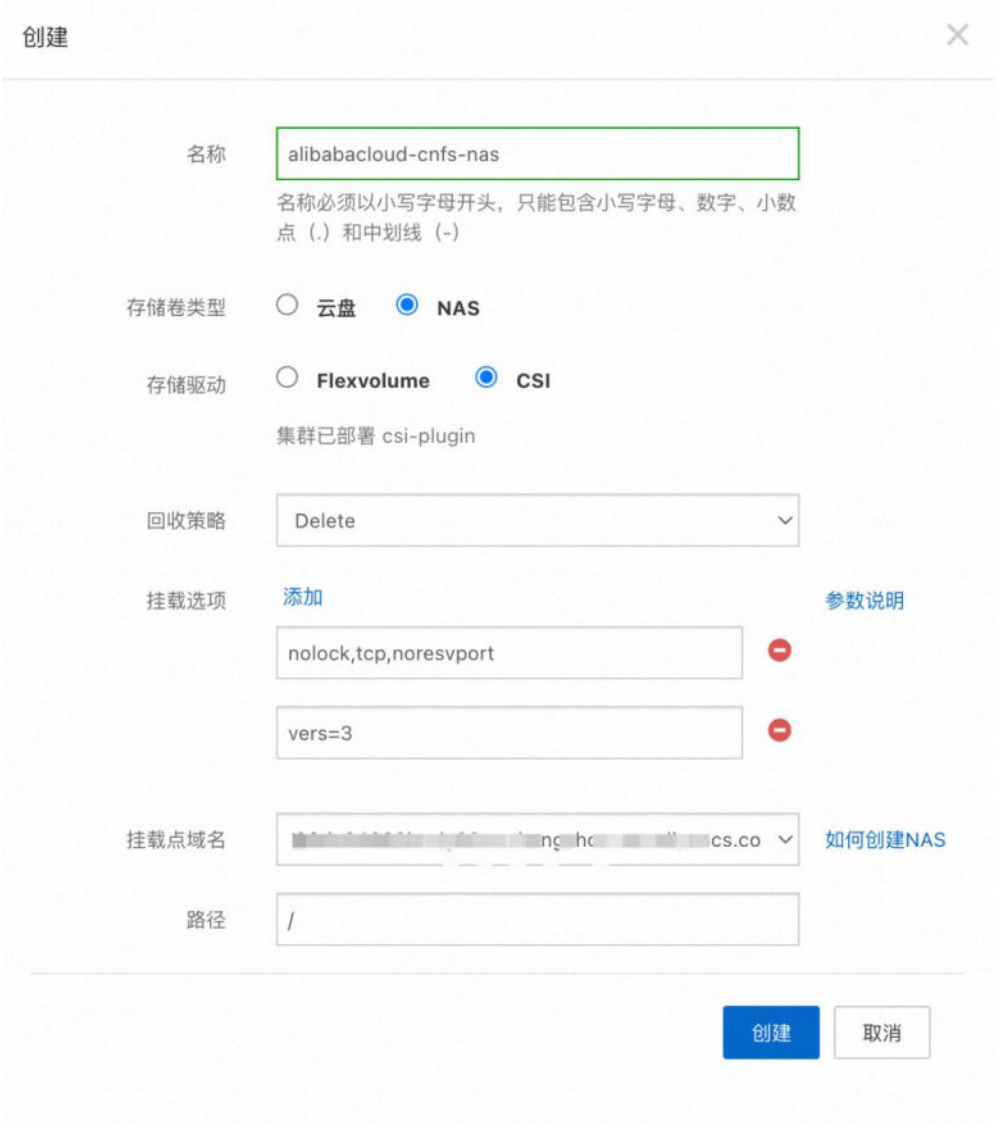
在根目录运行 ls -R 指令，可以看到如下的文件结构：

```
$ ls -R
pytorch    tensorflow

./pytorch:
```

```
style-transfer

./pytorch/style-transfer:
candy.onnx

./tensorflow:
style-transfer

./tensorflow/style-transfer:
saved_model.pb variables

./tensorflow/style-transfer/variables:
variables.data-00000-of-00002              variables.data-00001-of-00002
variables.index
```

b. 将模型文件加载到 PVC

首先创建一个存储类，前往容器服务控制台的 存储 > 存储类，创建一个存储类：



接着创建 PVC，前往容器服务控制台 存储 > 存储声明，用刚刚创建的存储类来创建一个存储声明 PVC，名字就叫 my-models-pvc。

c. 创建一个 pod 用来将模型文件拷贝到 PVC 里

前往容器服务控制台的工作负载 > 容器组，点击“使用 YAML 创建”，并在 YAML 框中输入以下内容，点击“创建”来创建一个 pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: "pvc-access"
  namespace: modelmesh-serving
spec:
  containers:
    - name: main
      image: ubuntu
      command: ["/bin/sh", "-ec", "sleep 10000"]
      volumeMounts:
        - name: "my-pvc"
          mountPath: "/mnt/models"
  volumes:
    - name: "my-pvc"
```

```
    persistentVolumeClaim:
      claimName: "my-models-pvc"
```

d. 使用 kubectl cp 将模型文件通过 pod 拷贝进 PVC

首先使用 kubectl 连接至 ACK 集群（参考获取集群 KubeConfig 并通过 kubectl 工具连接集群[10]）。接下来在刚才的模型文件根目录处，打开命令行，运行以下指令：

```
kubectl cp -n modelmesh-serving tensorflow pvc-access:/mnt/models/
kubectl cp -n modelmesh-serving pytorch pvc-access:/mnt/models/
```

接下来执行以下命令，确定拷贝已经成功：

```
kubectl exec -n modelmesh-serving pvc-access -- ls /mnt/models
```

预期得到以下内容，就说明模型文件已经被拷贝到 PVC 里了。

```
pytorch
tensorflow
```

e. 使用 InferenceService 自定义资源创建模型推理服务

使用以下内容，创建 isvc.yaml 文件

```
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: tf-style-transfer
  namespace: modelmesh-serving
  annotations:
    serving.kserve.io/deploymentMode: ModelMesh
    #serving.kserve.io/secretKey: myoss
spec:
```

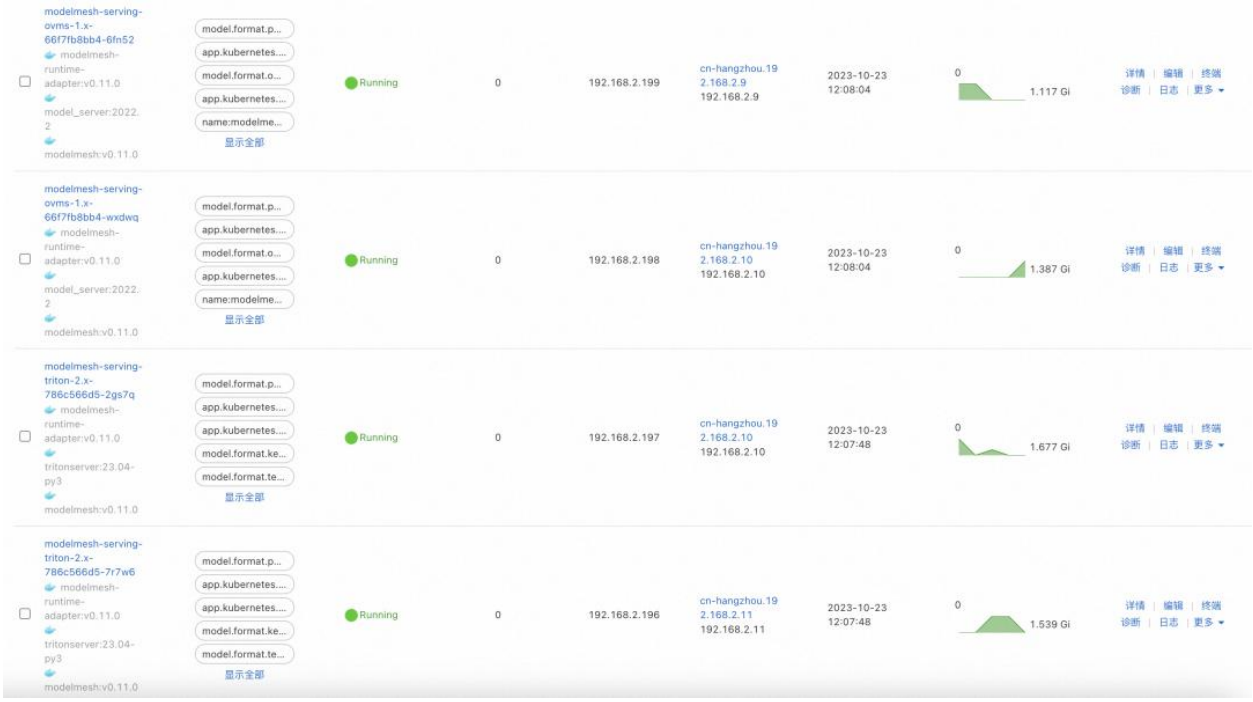
```
  predictor:
    model:
      modelFormat:
        name: tensorflow
      storage:
        parameters:
          type: pvc
          name: my-models-pvc
        path: tensorflow/style-transfer/
---
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: pt-style-transfer
  namespace: modelmesh-serving
  annotations:
    serving.kserve.io/deploymentMode: ModelMesh
spec:
  predictor:
    model:
      modelFormat:
        name: onnx
      storage:
        parameters:
          type: pvc
          name: my-models-pvc
        path: pytorch/style-transfer/
```

isvc.yaml 中声明了两个 InferenceService，分别对应 Tensorflow 和 Pytorch 模型的推理服务声明。

使用以下命令，在 ACK 集群中创建模型推理服务。

```
kubectl apply -f isvc.yaml
```

我们可以观察到在集群中，支持 Tensorflow 和 Pytorch 这两个模型的运行时工作负责 Pod 被动态扩容拉起，并开始加载对应支持格式的模型。在此 DEMO 示例中，我们用 InferenceService 分别声明了 Tensorflow 和 ONNX 格式的模型文件，因此，可以看到，对应拉起的运行时是 triton-2.x 运行时和 ovms-1.x 运行时。



当运行时启动与模型加载都完成后，使用 kubectl 获取 InferenceService，可以看到两个 InferenceService 也都对应处于就绪状态：

```
$ kubectl get isvc -n modelmesh-serving
NAME                URL                                                 READY   PREV
LATEST   PREVROLLEDOUTREVISION   LATESTREADYREVISION   AGE
pt-style-transfer        grpc://modelmesh-serving.modelmesh-serving:8033
True                                                                    11d
tf-style-transfer        grpc://modelmesh-serving.modelmesh-serving:8033
True                                                                    11d
```

3）在集群中部署业务服务

在模型推理服务的前面就是我们的业务服务了，分别是 style-transfer 业务服务和最前方

的 AI 应用服务，我们接下来就需要在集群中部署这些服务以及服务的工作负载。

a. 使用 kubectl 连接到 ACK 集群，并使用如下命令创建一个命名空间来部署应用

```
kubectl create namespace apsara-demo
```

b. 使用以下内容，创建 ai-apps.yaml 文件

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ai-backend
  namespace: apsara-demo
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: style-transfer
  namespace: apsara-demo
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: ai-backend
  name: ai-backend
  namespace: apsara-demo
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: ai-backend
  strategy:
    rollingUpdate:
```

```
    maxSurge: 25%
    maxUnavailable: 25%
  type: RollingUpdate
 template:
  metadata:
    labels:
      app: ai-backend
  spec:
    serviceAccountName: ai-backend
    containers:
      -                                                                image:
'registry.cn-hangzhou.aliyuncs.com/build-test/asm-apsara:g56a99cd1-aliy
un'
        imagePullPolicy: IfNotPresent
        name: ai-backend
        ports:
          - containerPort: 8000
            name: http
            protocol: TCP
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
---
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
   app: style-transfer
 name: style-transfer-tf
 namespace: apsara-demo
```

```
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: style-transfer
      model-format: tensorflow
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: style-transfer
        model-format: tensorflow
    spec:
      serviceAccountName: style-transfer
      containers:
        - image: >-

registry.cn-hangzhou.aliyuncs.com/build-test/style-transfer-tf:g78d00b1
c-aliyun
          imagePullPolicy: IfNotPresent
          name: style-transfer-tf
          env:
            - name: MODEL_SERVER
              value:
istio-ingressgateway.istio-system.svc.cluster.local:8008
            - name: MODEL_NAME
              value: tf-style-transfer
          ports:
            - containerPort: 8000
              name: http
              protocol: TCP
          resources:
```

```
          requests:
            cpu: 250m
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: style-transfer
  name: style-transfer-torch
  namespace: apsara-demo
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: style-transfer
      model-format: pytorch
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: style-transfer
        model-format: pytorch
    spec:
      serviceAccountName: style-transfer
```

```
    containers:
      - image: >-

registry.cn-hangzhou.aliyuncs.com/build-test/style-transfer-torch:g78d0
0b1c-aliyun
        imagePullPolicy: IfNotPresent
        name: style-transfer-torch
        env:
          - name: MODEL_SERVER
            value:
istio-ingressgateway.istio-system.svc.cluster.local:8008
          - name: MODEL_NAME
            value: pt-style-transfer
        ports:
          - containerPort: 8000
            name: http
            protocol: TCP
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: ai-backend
  name: ai-backend-svc
  namespace: apsara-demo
spec:
  internalTrafficPolicy: Cluster
```

```
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http
      port: 8000
      protocol: TCP
      targetPort: 8000
  selector:
    app: ai-backend
  type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: style-transfer
  name: style-transfer
  namespace: apsara-demo
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  ports:
    - name: http
      port: 8000
      protocol: TCP
      targetPort: 8000
  selector:
    app: style-transfer
  sessionAffinity: None
  type: ClusterIP
```

c. 使用 kubectl 执行以下命令来部署上方文件中声明的应用服务

```
kubectl apply -f ai-apps.yaml
```

4）创建 ASM 网关、waypoint 网格代理，并部署生效流量规则

部署的最后一部分都有关服务网格，具体来说有以下部分：

- ASM 入口网关。
- 网格 waypoint 代理，它是 Sidecarless 的服务网格能力载体。
- 服务网格流量规则，这些规则将生效到 ASM 网关和 waypoint 代理，保证流量路径按照我们的设计运行。

a. 部署 ASM 入口网关

我们可参考创建入口网关[11]，来创建 ASM 入口网关。我们需要创建两个 ASM 入口网关，其中一个叫 api-ingresgateway，服务类型为 LoadBalancer，网关上需要开启 80 端口；另一个叫 ingressgateway，服务类型为 ClusterIP，网关上需要开启 8008 端口。其余网关配置保持默认即可。

都创建完成后，我们应该可以在 ASM 入口网关页面看到这样的显示：

b. 开启 apsara-demo 命名空间的 Ambient Mesh 模式

- 登录 ASM 控制台[12]，在左侧导航栏，选择服务网格 > 网格管理。
- 在网格管理页面，单击目标实例名称，然后在左侧导航栏，选择网格实例 > 全局命名

空间。

- 在全局命名空间页面，单击从 Kubernetes 集群同步自动注入，选择数据面 ACK 集群后单击确定。
- 在全局命名空间页面的数据面模式列，单击 apsara-demo 命名空间对应的切换为 Ambient Mesh 模式，然后在确认对话框，单击确定。

c. 部署 waypoint 代理

使用 kubectl 连接到 ACK 集群，然后使用前提条件中安装的 istioctl 工具，执行以下指令：

```
istioctl x waypoint apply --service-account style-transfer -n apsara-demo
```

执行完成后，我们可以使用 kubectl 列出集群中的无状态工作负载。

```
kubectl get deploy -n apsara-demo
```

预期输出：

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
ai-backend                    1/1     1            1           13d
style-transfer-istio-waypoint   1/1     1            1           13d
style-transfer-tf             1/1     1            1           13d
style-transfer-torch          1/1     1            1           13d
```

可以看到集群中除了我们刚才部署的 AI 应用以及 style-transfer 应用的工作负载外，还

增加了一个名为 style-transfer-istio-waypoint 的工作负载，这就是服务网格的 waypoint 代理，它是以独立的工作负载方式部署在集群中的，所提供的所有能力也都是 Sidecarless 的。

d. 部署服务网格规则

① 使用以下内容，创建 modelsvc-routing.yaml 文件

```
# make sure voyage is 1.13.4.13 or higher
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: grpc-gateway
  namespace: modelmesh-serving
spec:
  selector:
    istio: ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: grpc
        number: 8008
        protocol: GRPC
    - hosts:
        - '*'
      port:
        name: http
        number: 80
        protocol: HTTP
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-modelmesh-serving-service
```

```
  namespace: modelmesh-serving
spec:
  gateways:
    - grpc-gateway
  hosts:
    - '*'
  http:
    - headerToDynamicSubsetKey:
        - header: x-model-format-tensorflow
          key: model.format.tensorflow
        - header: x-model-format-pytorch
          key: model.format.pytorch
      match:
        - port: 8008
      name: default
      route:
        - destination:
            host: modelmesh-serving
            port:
              number: 8033
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: dr-modelmesh-serving-service
  namespace: modelmesh-serving
spec:
  host: modelmesh-serving-service
  trafficPolicy:
    loadBalancer:
      dynamicSubset:
        subsetSelectors:
          - keys:
              - model.format.tensorflow
          - keys:
              - model.format.pytorch
---
apiVersion: istio.alibabacloud.com/v1beta1
```

```
kind: ASMGrpcJsonTranscoder
metadata:
  name: grpcjsontranscoder-for-kservepredictv2
  namespace: istio-system
spec:
  builtinProtoDescriptor: kserve_predict_v2
  isGateway: true
  portNumber: 8008
  workloadSelector:
    labels:
      istio: ingressgateway
---
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  labels:
    asm-system: 'true'
    provider: asm
  name: grpcjsontranscoder-increasebufferlimit
  namespace: istio-system
spec:
  configPatches:
    - applyTo: LISTENER
      match:
        context: GATEWAY
        listener:
          portNumber: 8008
        proxy:
          proxyVersion: ^1.*
      patch:
        operation: MERGE
        value:
          per_connection_buffer_limit_bytes: 100000000
  workloadSelector:
    labels:
      istio: ingressgateway
```

modelsvc-routing.yaml 中主要包含的是针对集群中的模型推理服务的流量规则。这主要包含两部分规则：

- 针对模型推理服务中不同运行时工作负载的动态子集路由能力高
- 针对 kserve v2 推理接口的 JSON/HTTP - gRPC 请求转码能力

我们将在下一个大章节介绍这些能力的细节。

② 使用 kubectl 连接 ASM 实例，执行以下命令，部署 modelsvc-routing 流量规则

```
kubectl apply -f modelsvc-routing.yaml
```

③ 使用以下内容，创建 app-routing.yaml 文件

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: ai-app-gateway
  namespace: apsara-demo
spec:
  selector:
    istio: api-ingressgateway
  servers:
    - hosts:
        - '*'
      port:
        name: http
        number: 8000
        protocol: HTTP
    - hosts:
        - '*'
      port:
        name: http-80
        number: 80
```

```
      protocol: HTTP
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: ai-app-vs
  namespace: apsara-demo
spec:
  gateways:
    - ai-app-gateway
  hosts:
    - '*'
  http:
    - route:
        - destination:
            host: ai-backend-svc
            port:
              number: 8000
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: style-transfer-vs
  namespace: apsara-demo
spec:
  hosts:
    - style-transfer.apsara-demo.svc.cluster.local
  http:
    - match:
        - headers:
            user_class:
              exact: premium
      route:
        - destination:
            host: style-transfer.apsara-demo.svc.cluster.local
            port:
              number: 8000
            subset: tensorflow
```

```
    - route:
        - destination:
            host: style-transfer.apsara-demo.svc.cluster.local
            port:
              number: 8000
            subset: pytorch
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: style-transfer-dr
  namespace: apsara-demo
spec:
  host: style-transfer.apsara-demo.svc.cluster.local
  subsets:
    - labels:
        model-format: tensorflow
      name: tensorflow
    - labels:
        model-format: pytorch
      name: pytorch
```

app-routing.yaml 中主要包含的是对 AI 应用服务和 style-transfer 服务的路由规则。其中包括一个对 style-transfer 不同工作负载进行根据用户身份分流的能力。

④ 使用 kubectl 连接 ASM 实例，执行以下命令，部署 app-routing 流量规则

```
kubectl apply -f app-routing.yaml
```

⑤ 将 ASM 网关对接阿里云 iDaas 应用身份服务，轻松实现单点登录

搭建整个应用的最后一步位于应用的总入口，也就是 ASM 入口网关。在这里，我们还需要将网关与阿里云 iDaas 的 OIDC 应用进行对接，对整个应用进行一个单点登录的配置。

我们可以参考这篇文档来进行对接的操作：ASM 集成阿里云 IDaaS 实现网格内应用单点

登录[13]。

值得注意的是，我们使用用户 jwt claim 中的额外字段 user_type 来完成用户身份的识别，这需要进行如下操作：

点击云身份服务的扩展字段，添加扩展字段（扩展字段名称和 OIDC 登陆后返回的字段名称均可以自定义，这里扩展字段定义为 user_type，OIDC 登陆后返回字段名称会在后面定义为 user_class）：

然后编辑用户信息，为指定用户设置该字段：

设置好该字段后，需要配置在 OIDC 登陆成功后，返回该字段。进入 OIDC 应用设置，点击登录访问 tab，点击“显示高级配置”。在这里设置新增一个 OIDC 登陆成功后返回的 key-value 对，key 是 user_type，value 是 user_class 的值。

我们披星戴月我们奋不顾身，终于！我们的 AI 应用搭好了！可以看到，从零开始搭建这样一套集成了模型推理的业务服务确实不能一步登天，不过服务网格 ASM 在这其中通过一些生态集成的能力，以及完善的 Web UI，将很多步骤进行了简化。

3）Try it out!

在 ASM 控制台的网格管理页面，我们可以直接看到 api-ingressgateway 的服务地址：

整个应用的访问入口就是 http://{ASM 网关服务地址}/home。用浏览器打开它，就可以开始玩我们的 AI 应用了~

2. 服务网格如何帮助我们

这个章节会简要介绍在这个 DEMO 中，服务网格 ASM 开启了怎样的一些能力，帮助我们做到更多。也就是我们在云栖大会中为大家介绍的内容。

1）针对模型服务运行时的动态子集路由

在 AI 应用的构建中，如何将训练好的模型转化为可靠的推理服务是工作的重心，因此我们首先介绍这个 DEMO 中的模型推理服务。

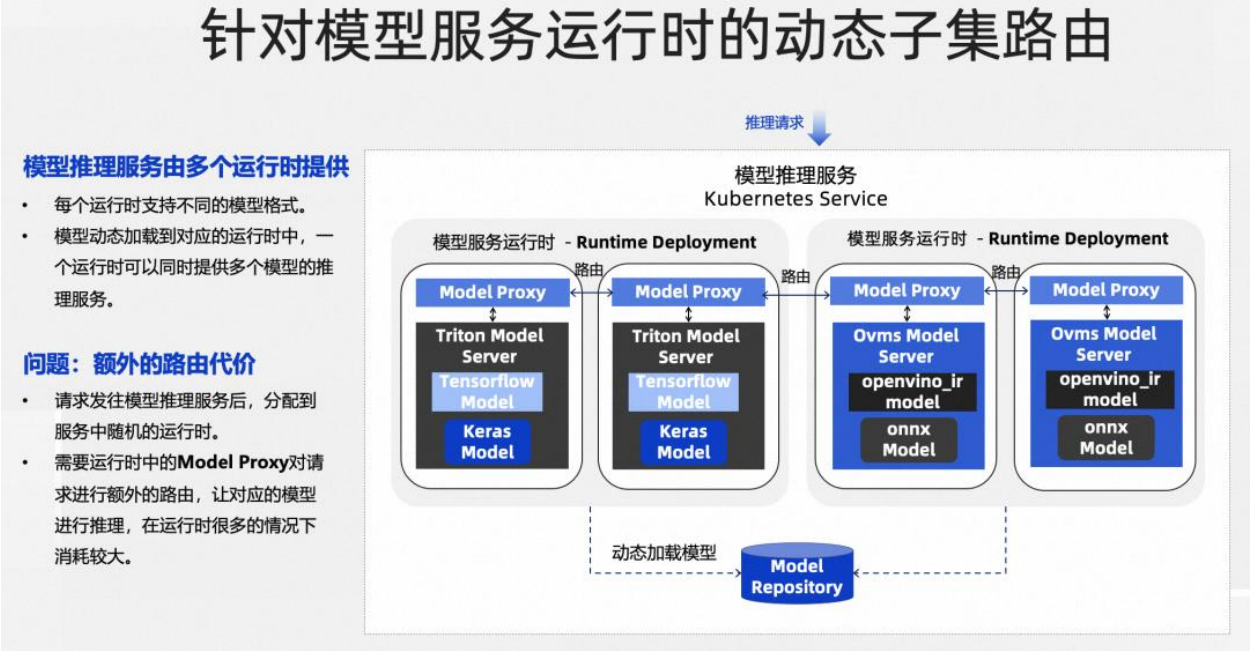
在模型推理服务的整体框架中，由一个整体的 k8s Service 对外提供所有模型的推理。然而，模型有很多格式种类、如何将类似 sklearn、tensorflow、pytorch 等等不同种类的模型统一成 API 相同的推理服务呢？这就要使用不同的运行时。

在统一的模型推理 Service 之下，不同的模型、其推理将交给多个运行时工作负载来完成。每个运行时支持不同的模型格式；并且可以同时提供多个模型的推理服务。当我们使用 InferenceService 资源定义一个模型后，模型文件将根据模型的格式、动态地加载到对应的运行时工作负载之中。一个运行时可以同时提供多个模型的推理服务。

通过这种方式，能够实现高弹性、高灵活性、低消耗的模型推理服务部署。

然而这种方式也存在问题，即存在额外的路由代价。由于 k8s Service 的机制，请求发往模型推理服务后，k8s 不会区分请求的模型格式、而是会随机将请求分发到不同的运行时工作负载，也就无法保证请求能够正确发往可提供服务的运行时。

这就需要在运行时中注入额外的 model-proxy，用来进行额外的路由操作、保证请求的正确响应，在运行时规模增大的情况下会造成消耗和性能问题。这也正是服务网格的重要价值所在。服务网格通过数据面的网格代理，能够动态识别模型推理服务内部、支持不同模型格式的运行时。并在推理请求发出时，根据请求元数据寻找匹配的运行时分组，保证请求能够直接发向正确的运行时，在不额外增加运维成本的同时降低系统路由消耗，这被称作动态子集路由能力。



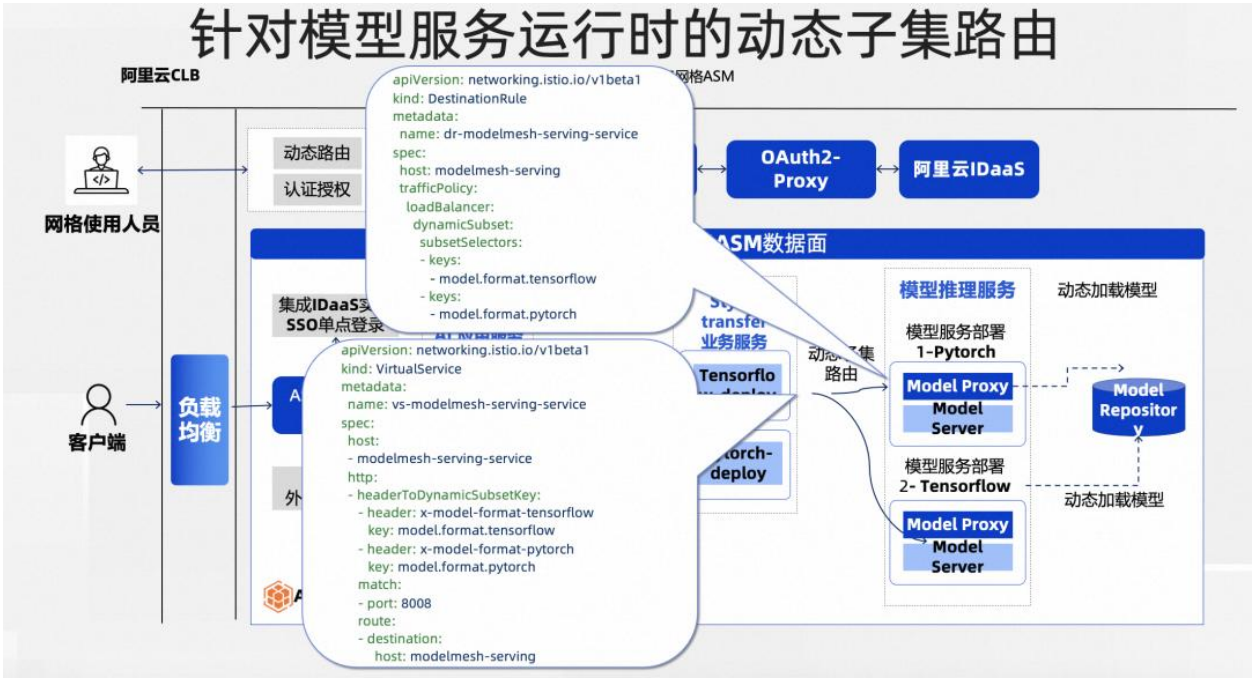
要实现动态子集路由能力，我们只需要使用针对服务配置的 DestinationRule 资源与 VirtualService 资源即可。

对运行时的识别主要通过工作负载的标签，声明一系列模型格式相关的标签，服务网格就

将以这些标签为依据、对运行时进行动态分组。在目标规则 DestinationRule 中，主要声明了一系列的标签信息，这些标签将成为工作负载的分组依据。

在下方的虚拟服务 VirtualService 中，我们可以看到基于标签动态分组的路由配置。具体来说，服务网格能够利用请求 header 信息生成请求元数据，元数据包含目标工作负载的标签信息，可以与工作负载的分组进行匹配。

在这个 DEMO 中，我们将请求中以 x-model-format 开头的 header 转换为请求元数据，并与 DestinationRule 中声明的工作负载分组进行匹配，找到请求应该发往的分组。



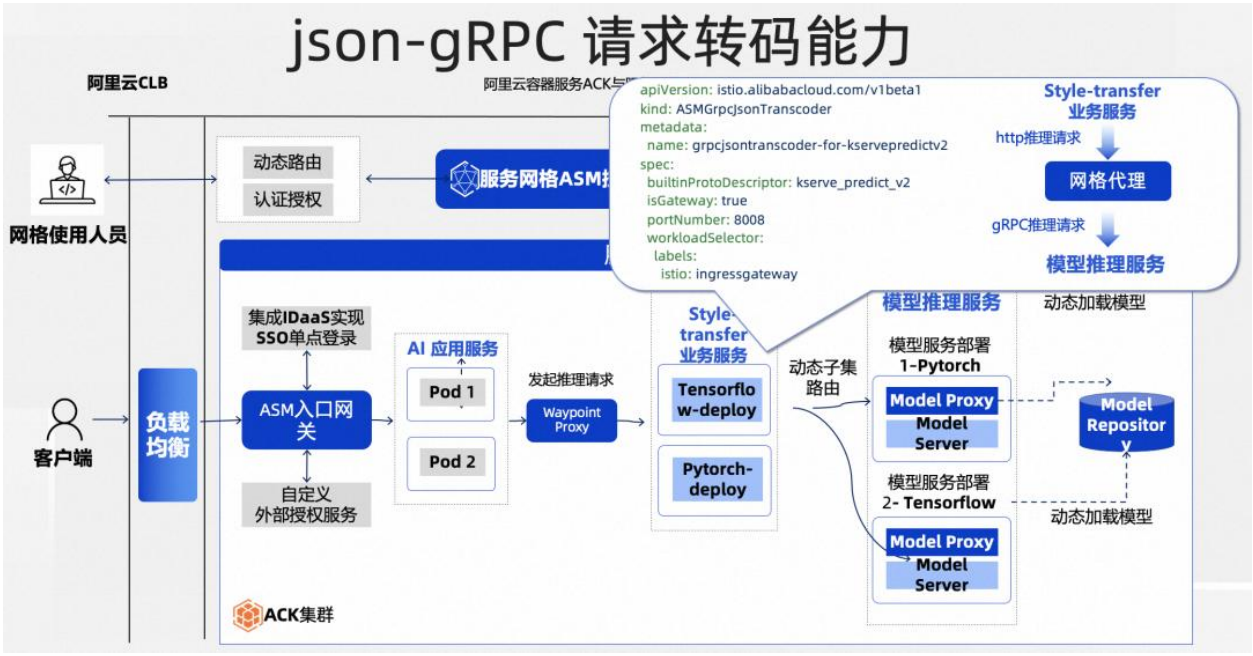
2）Json/http - gRPC 请求转码能力

在实现了动态子集路由的网格代理之上，我们还配置了 json to grpc 的转码能力。

当前，模型推理服务器大多都只实现了 gRPC 协议的服务，而对于依赖模型推理的业务服务来说，则可能是以 restful 等方式来实现服务之间的相互调用。因此，在业务服务调用模型推理服务时，可能存在协议不兼容、导致难以调用的情况。

通过在服务网格中配置 json to grpc 转码能力，原本只能通过 grpc 协议访问的模型推理服务、现在也可以通过 http 传输 json 数据的方式来访问。

如图所示，我们只需要声明 grpc 服务的 proto 描述，集群中的网格代理将替我们完成 restful 请求中 json 数据到 gRPC 请求体的动态转换，为集群中的服务调用增添更多的灵活性，解决调用协议的兼容问题。



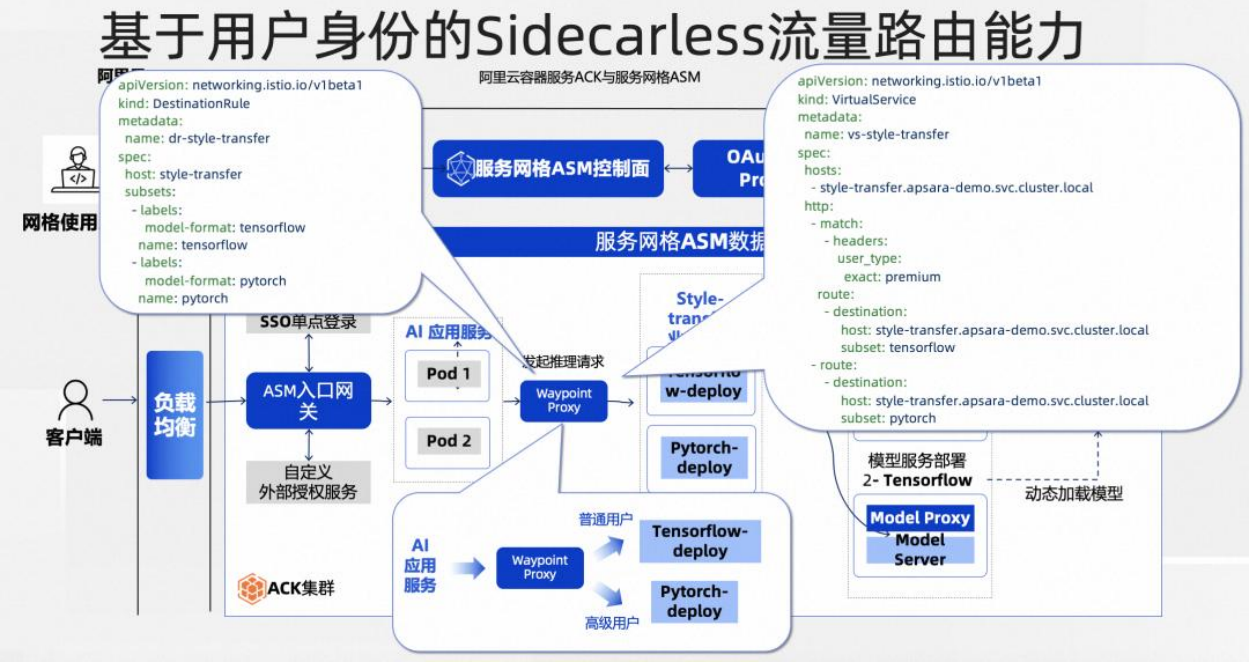
3）基于用户身份的 Sidecarless 流量路由能力

让我们将目光投向调用链路的前端，针对 AI 应用服务调用 style-transfer 业务服务的这一环，我们也发挥服务网格的能力，实现了基于用户身份的流量分流。

调用链路的上游是集群中的 style-transfer 业务服务，对于这个业务服务，我们针对 tensorflow 和 pytorch 两种模型，分别提供了名为 style-transfer-tf 和 style-transfer-torch 的不同工作负载，负责将下游应用传入的图片处理为模型可以接受的张量、并交给依赖的模型进行推理。而服务网格负责根据用户身份信息，将下游传输的数据交给不同的工作负载。

我们来看相关配置，首先，还是通过目标规则 DestinationRule 将中台业务服务下不同的工作负载进行分组。接着，虚拟服务 VirtualService 将根据请求中的用户信息，将流量发往不同的工作负载，用不同的模型对请求进行响应。其中请求的用户信息则是用户的 jwt claim，由 OIDC 应用提供。

在本 DEMO 中，服务网格的运用完全基于 Sidecarless 模式，上述能力是通过独立部署的网格代理 waypoint 实现的，也就是说，这些能力的实现不需要任何业务感知，能够大大提高服务网格的运维效率。



4）ASM 网关集成 OIDC 应用实现单点登录能力

最后，在整个调用链路的最前端就是作为流量入口的 ASM 网关。

DEMO 在 ASM 网关上实现了与 OIDC 应用的快速对接来配置单点登录。本次 DEMO 中使用阿里云 idaas 应用身份服务。通过将网关与 OIDC 应用进行对接，网关后的应用无需自己实现身份认证、即可对集群中的应用完成单点登录并拿到用户身份。

如图所示：在服务网格 ASM 中，通过一个完善的 Web 界面即可快速配置与已有 OIDC 应用的对接，这能够大大降低单点登录系统的实现与运维成本，提升运维效率。

5）小结

最后让我们简单总结一下。

在此次的 DEMO 应用中，服务网格 ASM 针对服务调用链路上不同服务的特性以及业务需求，能够灵活配置不同的流量路由以及流量处理规则，快捷地完成应用的各项运维工作；同时，这些能力的生效也是完全基于 Sidecarless 模式，对业务几乎无感知，服务网格进一步沉淀为应用的流量基础设施。作为业务入口的 ASM 入口网关，在满足基础的路由和安全能力之外，还提供丰富的生态集成、证书管理等增强能力，并都辅以完备的 Web 界面帮助用户进行快速配置。

大家可以根据自身需求，选择使用服务网格的相应能力，Let Service Mesh helps you to achieve more！有关更多的产品能力，欢迎参考官方文档[14]。

相关链接：

[1] 创建 ASM 实例

https://help.aliyun.com/zh/asm/user-guide/create-an-asm-instance#task-2370657

[2] Kubernetes 集群及配置要求

https://help.aliyun.com/zh/asm/user-guide/restrictions-on-use#rwA6T

[3] 创建 Kubernetes 专有版集群

https://help.aliyun.com/zh/ack/ack-managed-and-ack-dedicated/user-guide/create-an-ack-dedicated-cluster#steps-7hk-mqa-7wa

[4] 创建 Kubernetes 托管版集群

https://help.aliyun.com/zh/ack/ack-managed-and-ack-dedicated/user-guide/create-an-ack-managed-cluster-2

[5] 添加集群到 ASM 实例

https://help.aliyun.com/zh/asm/getting-started/add-a-cluster-to-an-asm-instance-1#task-2372122

[6] Istio

https://github.com/istio/istio/releases/tag/1.18.2

[7] ASM 集成云原生推理服务框架 KServe

https://help.aliyun.com/zh/asm/user-guide/integrate-the-cloud-native-inference-service-kserve-with-asm

[8] 管理全局命名空间

https://help.aliyun.com/zh/asm/user-guide/manage-global-namespaces

[9] 通过控制面 kubectl 访问 Istio 资源

https://help.aliyun.com/zh/asm/user-guide/use-kubectl-on-the-control-plane-to-access-istio-resources

[10] 获取集群 KubeConfig 并通过 kubectl 工具连接集群

https://help.aliyun.com/zh/ack/ack-managed-and-ack-dedicated/user-guide/obtain-the-kubeconfig-file-of-a-cluster-and-use-kubectl-to-connect-to-the-cluster

[11] 创建入口网关

https://help.aliyun.com/zh/asm/user-guide/create-an-ingress-gateway?spm=a2c4g.11186623.0.i1

[12] ASM 控制台

https://account.aliyun.com/login/login.htm?oauth_callback=https%3A%2F%2Fservicemesh.console.aliyun.com%2F&lang=zh

[13] ASM 集成阿里云 IDaaS 实现网格内应用单点登录

https://help.aliyun.com/zh/asm/user-guide/integrate-alibaba-cloud-idaas-with-asm-to-implement-single-sign-on

[14] 官方文档

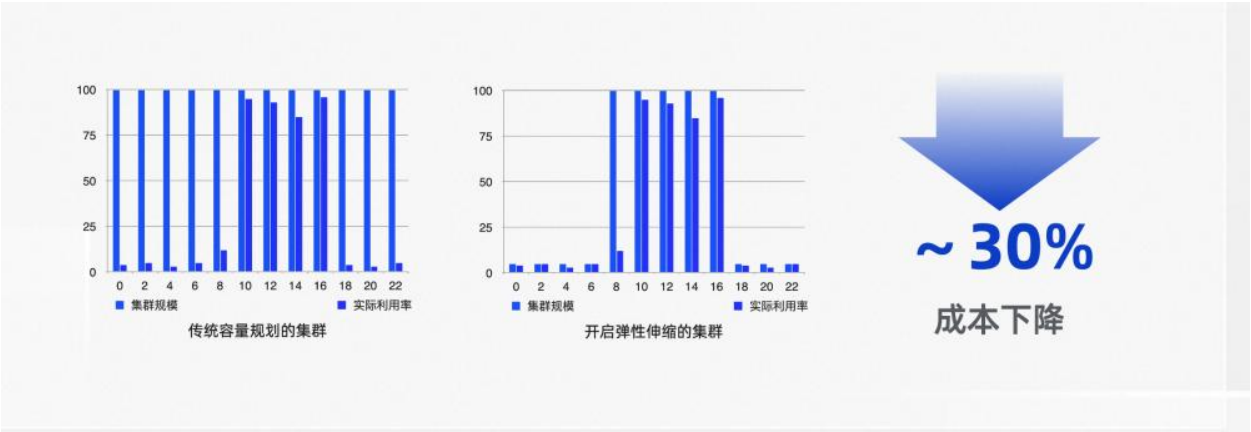
https://help.aliyun.com/zh/asm

阿里云云原生弹性方案：用弹性解决集群资源利用率难题

作者：郭赫曦，阿里云技术专家

随着上云的认知更加普遍，我们发现除了以往占大部分的互联网类型的客户，一些传统的企业，一些制造类的和工业型企业客户也都开始使用云原生的方式去做 IT 架构的转型，提高集群资源使用率也成为企业上云的一致共识。大家上云的同时，开始思考有没有云原生的方法能更好地实现提高集群资源使用率这个核心目标。

如何提升集群资源使用率呢，其实这个问题是要去解决规划容量和真实需求容量直接存在差异，这里边最重要的一个途径就是通过弹性来去解决一些问题，实现成本优化。传统的容量规划为了保证业务高峰稳定性，资源需要按照业务最高使用量进行常备，如左图所示，这种方案保证了稳定性，但是资源利用率很低，成本浪费比较大。使用弹性伸缩后，资源容量曲线和实际业务资源需求的曲线贴合度增高，也就是我们追求的资源利用率有了明显的提升，从而整体成本也得到下降。



1. 阿里云云原生弹性解决方案

明确了弹性的目标后，那我们下面具体看看阿里云容器服务各维度提供了哪些弹性解决方案。

弹性是分成不同维度的。如果我们把 IT 架构分成两个层次的话，最上面一层是应用层。应用层上面如果做弹性，按照伸缩的方向可以分为水平和纵向，另外还有精细化调度，解决应用层和调度器的一些策略问题的。使用最普遍的是容器水平伸缩，阿里云在这里也为大家提供了很丰富的弹性能力，以便大家能用最合适的组件来扩缩自己的业务 Pod。后面会为大家具体分析如何根据自己的业务特性来选择。

第二个层次是资源层，也就是云厂商主要负责维护和提供给客户（包括 K8s 平台和底层虚机、网络、存储等资源的供给）。从资源形态可以分为节点交付和无服务器资源交付两类。在资源层，我们主要做的就是怎么去满足这个应用平台的稳定性，以及怎么去解决容量规划和实际的业务使用上的这个差异。这个是在资源层弹性需要核心建设的能力。

接下来，我们会针对于上面的这一些能力，给大家做一些深度的解析，来看一下阿里云 ACK 在这个地方是怎么投入的，有哪些能力建设。

1）应用层弹性

首先我们先来看这个应用层的水平伸缩。先看触发源，从触发源的类型可以分成 3 大类，指标、资源和事件。基于指标 或者说资源阈值模型的就是咱们最常见的 HPA，HPA 适合业务高低峰能用指标来描述的业务，比如有的微服务业务高低峰会反映在 cpu、内存这类

资源指标上，再比如说网站或者 API 可以通过访问量来决定是否需要扩缩，这些都是比较适合 HPA 的场景。

有的服务和时间是强相关的，比如外卖业务，饭点附近高峰，其他时候低峰，还比如促销场景，在规划好的促销时间段内会盈来高峰，这类就很适合使用 CronHPA，基于时间来扩缩 Pods。还有一类是以上两种都不能很好覆盖的，就是事件类型的，比如消费消息的业务，需要根据 MQ 中消息的多少来判断是否需要扩缩，这种就可以考虑使用 KEDA 来满足。

再看看这其中使用最普遍的也是大家最熟悉的 HPA 上阿里云扩展了哪些能力。首先弹性指标方面是否丰富，应该是大家比较关心的，这个也是第一个阿里云容器服务重点建设的方向。像传统的 HPA，大家可能理解指标就是 cpu 、内存。这个是社区版本的 resource metrics 里唯一定义的两种的这个伸缩指标。对于不同的业务场景，对于不同的这个业务形态，简简单单的 cpu 跟内存是没有办法满足业务的诉求的。那阿里云在 HPA 弹性指标这个维度做的事情，就是找到合适不同业务场景的伸缩指标，来进行丰富，让这个链路使用起来更简单。

为此，阿里云 ACK 提供了 Metrics Adapter 这个组件来完成指标转化，支持用户自定义指标，比如 GPU 的利用率、Prometheus 的指标都是可以通过它让 HPA 感知到。除了自定义指标，阿里云产品常用指标也在其中默认支持，比如 Ingress/Istio/AHAS QPS，用户需要的话可以直接在 HPA 配置使用。

伸缩指标再往下是伸缩对象，我们知道 HPA 的伸缩对象是哪些呢？可以伸缩 Deployment，可以伸缩 Statefulset，实际上 HPA 不关心具体的伸缩对下是 Department 还是 Statefulset。他实际上感知的是一个泛化的 scale 的一个 subresource。也就是说只要资源对象实现了 subresource 这种资源对象，它就能够被 HPA 去管理，就能够被 HPA 伸缩。

所以在伸缩对象这个维度上，我们找到有哪些领域场景是需要去定义成一个伸缩对象的，然后进行支持。比如说阿里云 ACK 在 Spark 和 Presto Cluster 上面去做了对应的 CRD，并且这些 CRD 同样也是可以被 HPA 去接管的，满足这些领域场景的自动水平伸缩的这个需求。

2）资源层弹性

应用层的自动伸缩负责对 Pods 的自动伸缩控制。设想一个场景，某个服务在突增的业务高峰需要 100 个 Pods 扛住流量，集群常备的资源够跑 50 个 Pods，那会剩余 50 个 Pods 没有资源可以调度。这个就是资源层弹性要解决的问题了，资源层弹性负责集群有足够的资源调度 Pods，且在不需要那么多资源的时候，自动释放，不造成太大的浪费。

更细地拆分的话，有 5 个维度的核心问题是资源层弹性最关心的，也是直接影响我们资源层弹性方案选型的 xb;我们刚才介绍过，从资源类型维度来分，资源层弹性可以分为 ECS 还是 ECI，那我们就以他们各自的的典型组件 cluster-autoscaler 和 VK 来举例说说资源层弹性的 5 个维度。

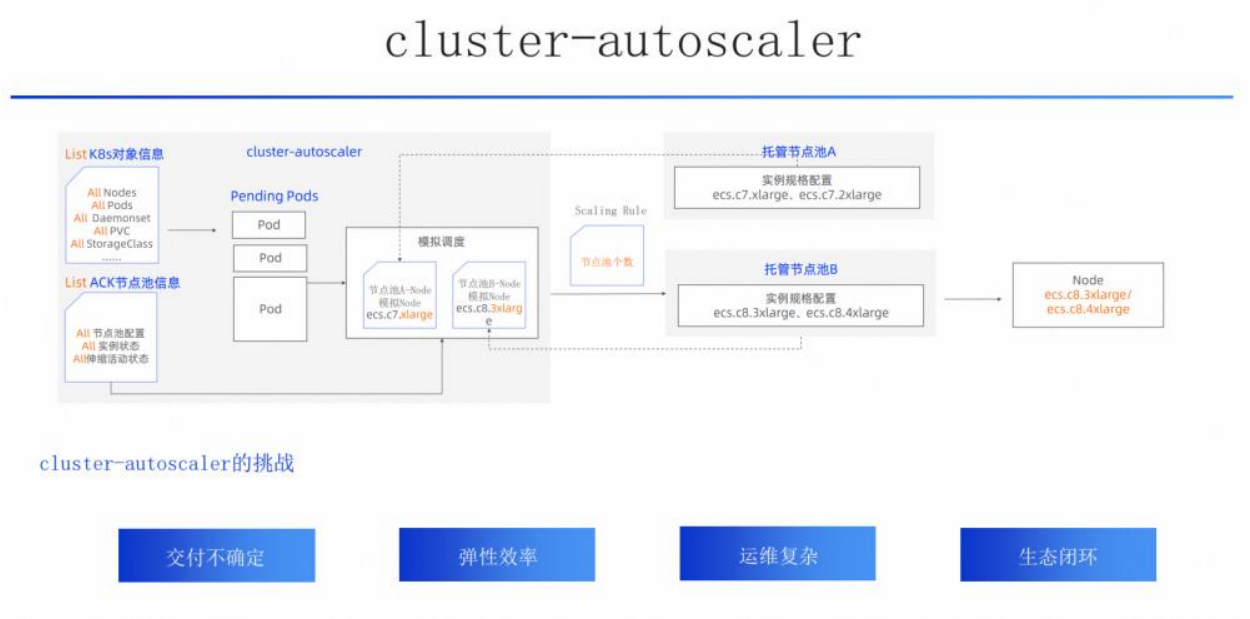
首先看成本，VK/cluster-autoscaler 的成本对比主要的区别在于超卖比，ECI 不支持超卖，通常离线作业的超卖比平均在 1:2-1:4 之间。然后看效率，cluster-autoscaler 是近分钟级别的（1 分钟多并且会随着节点池的个数，连续弹性的负载个数出现不稳定的交付）。ECI 是 1 分钟内的快速交付（在连续弹性等场景下，有绝对的优势）。

再看可以支持的多大的集群规模，VK 的规模化问题在于 One Pod One Node 模型对底层 API 与 APIServer 的冲击是直接的冲击，cluster-autoscaler 是集群/节点两级的模型，

相比而言，规模化场景下 cluster-autoscaler 的容量上限会更高。兼容性方面，cluster-autoscaler 是全兼容，ECI 对于需要内核参数、Daemonset 等场景支持存在差异。最后是运维难易，VK 为免运维，cluster-autoscaler 为强运维。

我们具体看下业界使用普遍的 cluster-autoscaler 和它面临的挑战。首先 cluster-autoscaler 是轮询模式，图上是一个 loop 的基本逻辑示意图。每个 loop 中，cluster-autoscaler 对全集群的状态机的维护，找到集群中不可调度的 Pods，然后把每个开启弹性的节点池抽象为一个 Virtual Node（以下简称为 One Nodepool One Virtual Node），判断是否可以部署（到节点池供给的资源上）后，增加对应节点池的节点个数，实现扩容。这就是大多数弹性客户在使用 cluster-autoscaler 的基本逻辑。

同时对很多应用和工作负载来说这套机制都运行的不错。但是随着更多的用户上云，ACK 也被更广泛地使用，用户正在讲将各种不同类型的工作负载转移到 ACK 上，也就对基于依赖全集群状态维护和 One Nodepool One Virtual Node 的 cluster-autoscaler 的这套机制引入了更多的挑战。主要集中在：交付不确定性、弹性效率：维护全局状态机、单节点池串行扩容（轮询模式对底层 API 的压力）、运维：节点池膨胀、问题排查难度大、生态闭环这几点。

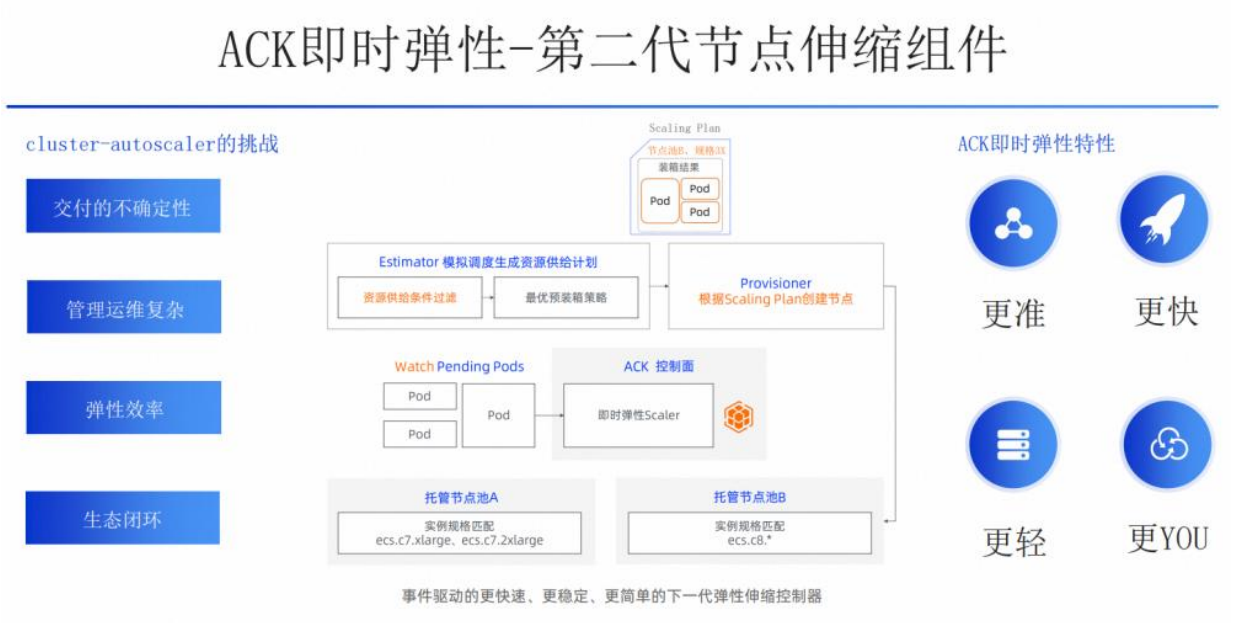


3）即时弹性

cluster-autoscaler 的交付的不确定性、使用复杂性以及不够快的弹性速度、相对闭环的生态是很多开发者对于生产环境开启弹性的 4 个最大的顾虑。为了解决这些顾虑我们研发了第二代节点伸缩产品——即时弹性。

即时弹性是一个基于事件驱动的节点伸缩控制器，兼容现有的弹性节点池语义与行为，支持所有类型的应用无感开启与使用。有以下特性：

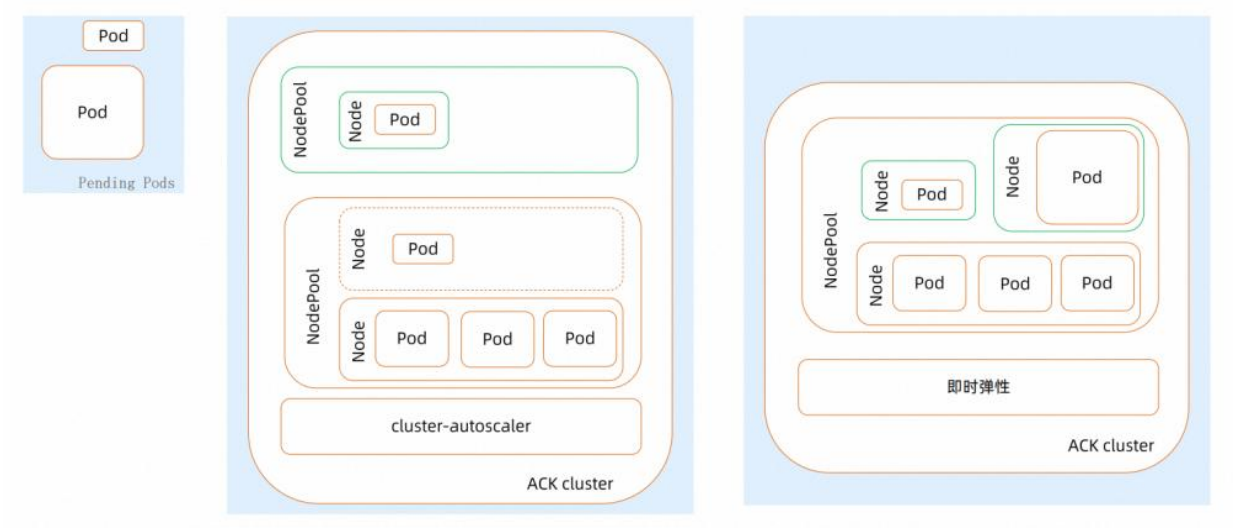
- 更准：即时弹性摒弃了第一代弹性组件的 One Nodepool One Virtual Node 的抽象方式和预调度装箱模式，在扩容时候决策从简单的只包含节点个数的 Scaling Rule，扩充到支持具体实例规格的 Scaling Plan，使得扩容更准。
- 更快：基于事件驱动和并行扩容让即时弹性更灵敏更快速。
- 更轻：即时弹性能自动选择实例规格，需要节点池数目更少，管理运维更轻 xb;，
- 更 YOU：在扩容和缩容阶段都支持了可扩展机制，让用户逻辑参与到弹性节点生命周期来。



下面我们针对这四个方面，用具体的业务 case 为大家对比两代节点伸缩组件的细节差异。

看个典型的扩容场景，集群内的节点资源已经被 3 个 Pods 占满 xb;这个时候出现了一个 Request 资源更小的 Pod，假设这个 Pod 可以调度到现有节点池的新节点上。如果是 cluster-autoscaler，会触发已有节点池弹一个相同大小节点，需求资源更小的 Pod 可以调度，但是节点资源利用率很低。

如果我们想更好地节点资源利用率，那只能再创建个新的节点池，配置上更合适的小资源的实例规格，但是这样又加重了后续运维负担。那如果是即时弹性的话，只需要在同一个节点池内配置好了多种规格的实例，那即时弹性会根据 Request 资源挑选合适的实例规格进行扩容。可以看出，即时弹性的交付确定性对资源利用率和运维这两方面都带了好处。

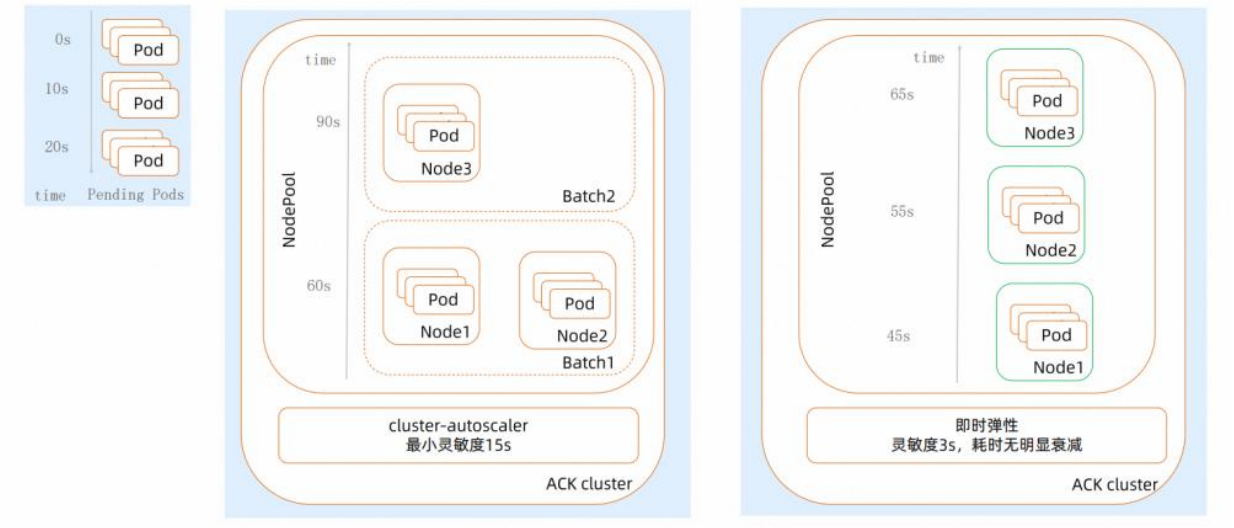


然后我们再看看弹性效率方面。

cluster-autoscaler 我们说过是轮询模式的，最小间隔 15s，他的扩缩处理是分批次的，比较适合单次交付这种模型，xb;但是如果遇到像流失计算、Workflow 这类分批或者一个时段连续出现的不可调度 pods 的场景，cluster-autoscaler 就会出现跨批次处理，批次之间还可能出现互相影响，导致弹性效率不稳定。

而即时弹性是基于事件的，Pods 出现不可调度的事件后即时弹性就会开始处理，并且支持同一节点池的并行扩容，这样就大大降低了触发扩容的耗时。

我们这个例子中可以看到有 3 批 Pods，间隔是 10s，在 cluster-autoscaler 因为最后这个 20s 的这批 pods 出现了跨批次，导致所有 pod 耗时 90s 才能全部调度；而即时弹性，Pods 没有因为批次而延迟的问题，每一批基本耗时都可以稳定在 45s 上下。



再来看看运维方面。

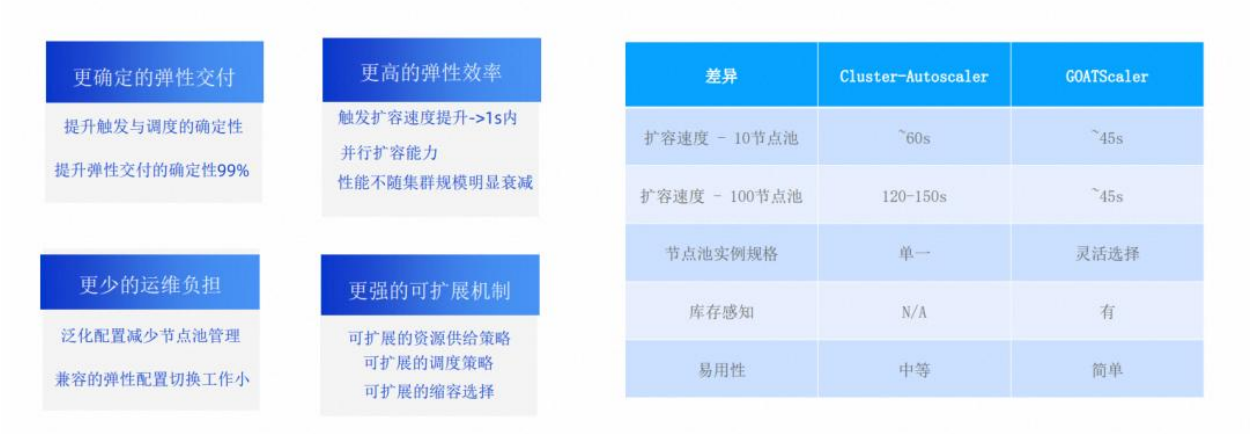
运维的成本其实是前期很容易忽视后期很容易头疼的点，将近一半的弹性用户都有对运维工作复杂的抱怨。复杂性一方面来自 cluster-autoscaler 交付不确定性带来的节点池数目膨胀。还有很重要的一方面来着 cluster-autoscaler 问题排查的复杂度，我们看一个经常用户会问到的问题，pod 预期是可以通过弹性扩容部署的，但是很长时间就是没有调度下去。这种问题在 cluster-autoscaler 中要排查的话，可能需要排查从业务层到基础设施层各层的日志都需要看，复杂度是指数级上升的。

那用户理想的方式是怎样的呢，咱们从产品全流程看，首先配置时候有提示，尽量前置的规避掉一些配置上容易出错的点。然后 Pod Pending 时候，弹性相关的需要用户关注的都能通过 Pod Events 透出，用户只需要 describe 一下 Pod 就可以看到。最后，如果 Pod 已经被删了，或者需要看一写统计方面的情况，能有大盘进行追溯，这些简化运维的能力也都会在即时弹性产品上推出。

最后看下扩展能力方面，cluster-autoscaler 因为要维持全局状态，生态是比较封闭的，用户没法参与到节点生命周期中来。即时弹性在扩容方面，会允许用户指定更细节的扩容要求，比如可用区、实例规格的优先级，以及对竞价实例的限定，最后的扩容计划会遵循用户这些策略。在缩容方面，也很开放，用户可以遵循即时弹性协议的 Policy，来按照自己的逻辑指定缩容的节点，即时弹性来为这些节点缩容。

总结下，相比传统的 cluster-autoscaler，即时弹性拥有更快速的弹性速度，更稳定的弹性效率，特别是在集群的规模变大或者弹性伸缩的频率变高的场景下，即时弹性相交传统的弹性模型，拥有 50% 以上的效率提升。此外，即时弹性还简化了节点池的使用复杂度，根据不同的应用选择合适的规格是运维人员非常困扰的问题，即时弹性通过自动模拟装箱策略，可以在开发者配置少量筛选规则的范围内，在阿里云超过 1500 款的机型中选择合

适业务的机型，进行弹性供给。不仅降低了开发者的使用成本，还提升了节点池弹性的成功率。



差异	Cluster-Autoscaler	GOATScaler
扩容速度 - 10节点池	~60s	~45s
扩容速度 - 100节点池	120-150s	~45s
节点池实例规格	单一	灵活选择
库存感知	N/A	有
易用性	中等	简单

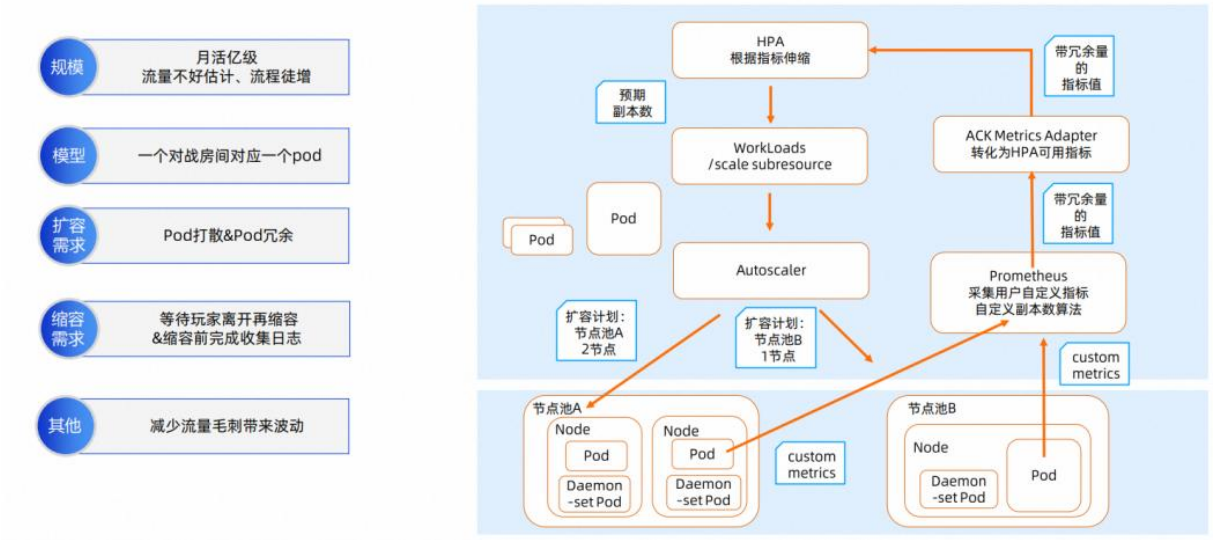
2. 企业云原生弹性案例与解析

我们用一个生产环境的实际案例来举例说明，上面提到的各层次弹性方案在具体情况下是如何被选择、运作和实际效果是怎样的。首先介绍这个用户的一些基本情况和特点：

- 规模：用户主要想把弹性运用在一个月活达到上亿级别的游戏的 AI 场景。月活很高但是流量却无法预测，可能因为客户涌入而有突增的流量。
- 模型抽象方面：用户将一个游戏对战房对应一个业务负载的 Pods，每个作战房的玩家数目有固定上限。
- 扩容需求：除了基本的扩容要求外，用户为了保证稳定性希望 Pods 可以基于节点打散。其次，用户一方面希望能随业务高低峰自动扩缩 Pods 数目，但是为了更好地应对突增流量，用户需要总是有固定数目的 Pods 冗余。
- 缩容需求：为了保证玩家体验，用户希望有玩家的 Pods 及其所在 Node 能等待玩家离场后再缩容，并且需要在节点缩容前去收集一些数据和日志，节点同时也需要在数据和日志收集完成后缩容，否则会导致数据或日志缺失。

1）应用层弹性方案

从用户业务场景分析，业务负载的变化和玩家数量是正相关的，其他维度上没有找到更好地规律，因此我们推荐用户通过 Aliyun Prometheus 采集玩家数目，然后安装 Metrics Adapter 组件将玩家数目转化为 HPA 的 custom metrics。为了满足最后算出的副本数总是带冗余量的，而侵入式地修改 HPA 的内部算法是一个维护成本较高的方案，因此我们建议用户在指标侧加入相应的冗余量，来保证 HPA 的输出副本数是带冗余数目的。



2）资源层弹性方案

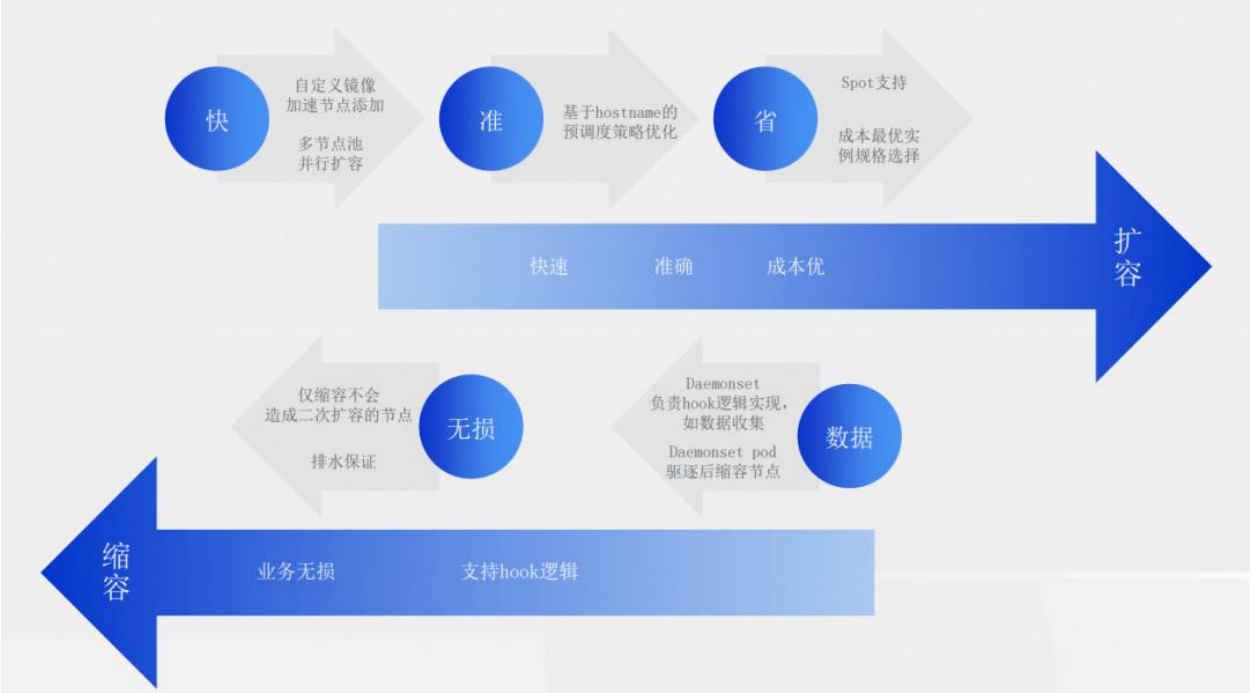
资源层弹性方案我们分扩容和缩容两方面来介绍。

扩容方面，虽然业务负载层实现了冗余 Pods，但是在较大量的突增流量的情况下，仍然需要快速地供给资源。阿里云提供了预先加载节点组件镜像和用户镜像的自定义镜像方案和工具，帮助节省新节点扩容和业务启动过程中镜像拉取的时间。同时，还提供了多节点池并行扩容，让分属于不同节点池的业务的资源供给不互相影响，可以同时进行。除了快用户还需要基于节点 HostName 的打散，我们优化了弹性组件中基于 HostName 的预调度调度策略，让弹性决策更加准确。

在稳定性得到保证的前提下，资源成本永远是用户最关心的问题之一。阿里云抢占式实例（Spot 实例）的市场价格会随供需变化而浮动，相对于按量付费实例能最高节约 90% 的

实例成本。集群使用抢占式实例是能够大幅节省成本的，但是抢占式实例随着动态库存和出价会不定时回收。

为尽可能降低抢占式实例的中断回收的影响，阿里云弹性提供了抢占式实例回收前提前主动排水、提前扩出新节点以备补偿等多种能力。这样对抢占式中断回收有一定容忍性地业务，比如离线数据类型业务，用户选择了使用抢占式实例作为弹性资源供给类型。对稳定性要求较高，我们也提供了成本最优策略，可以保证在按量实例的多种实例规格中，每次都可以扩容出成本最低的实例规格。



缩容方面，为了满足玩家退场再缩容的需求，阿里云弹性提供了自定义的排水等待时间，保证在用户指定时间内等待排水完成，不删节点资源。用户通过 K8s 的优雅退出来保证 Pods 被删前玩家都已经退场。这样两层保障即可让最终的缩容效果满足玩家退场节点再缩容的需求。

另外，用户缩容前数据日志收集的需求也是类似的，不同地方时这类针对每个节点都需要的服务一般是以 Daemonset 部署的，而一般缩容都是默认跳过 Daemonset Pods，并且用户场景中也只是需要对某一类或者某几类的 Daemonset Pods 进行排水并等待，因此我们在弹性缩容时添加了对指定 Daemonset Pods 进行缩容的支持，同时用户也可以

标记需要等待完成排水的 Daemonset Pods，以达到数据日志收集完全后，节点才被缩容的需求。

基于 ACK One 实现简单的跨云协同，让业务管理更高效

作者：庄宇，阿里云高级技术专家

2 年前的云栖大会，我们发布分布式云容器平台 ACK One，随着 2 年的发展，很高兴看到 ACK One 在混合云，分布式云领域帮助到越来越多的客户，今天给大家汇报下 ACK One 2 年来的发展，如何帮助客户解决分布式领域多云多集群管理的挑战。

首先我们会介绍下分布式云容器平台 ACK One 的背景与架构，之后详细介绍 ACK One 各个功能与应对的业务场景，包括注册集群、边缘集群、多集群舰队、全托管 Argo 工作流集群。

1. 分布式云容器平台 ACK One，解决企业多云多集群管理挑战

我看下 ACK One 这个产品出现的背景，Gartner 报告中预测到 2025 年，50% 的大型企业将通过分布式云实现业务转型，在 Forrester 报告中，未来 89% 的企业至少使用两个云，74% 的企业至少使用三个云。在 ACK 的用户中，我们也看到同样的趋势，越来越多的用户选择多云多集群架构，来应对业务发展带来的挑战。

可以说，分布式云已经成为新常态。同时 Gartner 报告中指出，安全，运维复杂性，财务复杂性是多云多集群架构的主要挑战。

为了应对分布式云带来的挑战，我们构建了分布式云容器平台 ACK One。

ACK One 提供多云多集群管理解决方案。

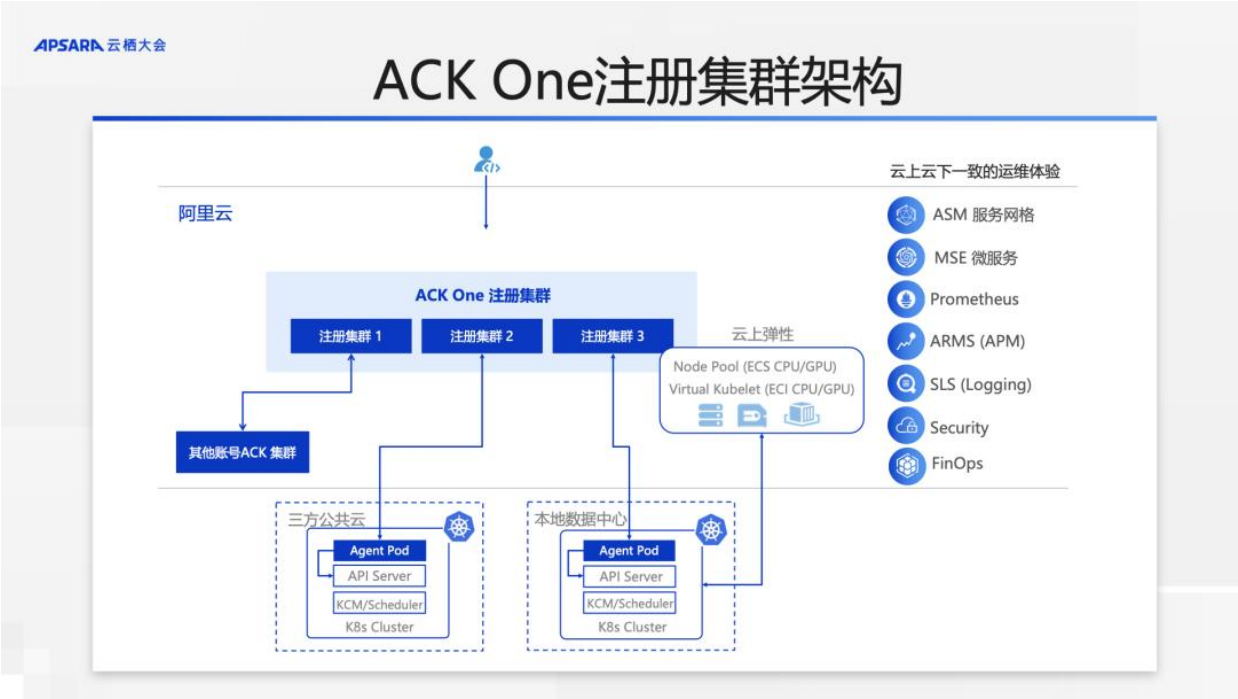
ACK One 可以统一接入不同形态\不同位置的 K8s 集群，例如：公共云 ACK 集群，边缘集群，本地数据中心中的集群，和第三方公共云集群。接入后，ACK One 提供云上云下一致的运维体验包括：监控日志、成本分析、安全加固、备份容灾、微服务治理等。极氪汽车使用 ACK One 统一管理多个 K8s 集群，提升了安全水位和业务连续性，减少 25% 的资源用量，运维效率提高 80%。

为了解决本地数据中心弹性不足，无法应对业务高峰的问题，ACK One 支持云上弹性，将云上无限算力接入到本地数据中心 K8s 集群中，并实现云上云下混合调度。智联招聘使用 ACK One 云上弹性能力，5 分钟实现业务数万核扩容，应对招聘季业务高峰。

同时，ACK One 支持多集群舰队管理，提供一个控制面管理多个 K8s 集群，提供统一应用分发、南北流量管理、统一运维管理等能力。

2. ACK One 注册集群：接入和管理不同形态 K8s 集群

下图展示的是 ACK One 注册集群的架构，我们可以将三方公共云中的 K8s 集群和本地数据中心的自建 K8s 集群接入到 ACK One 注册集群中。

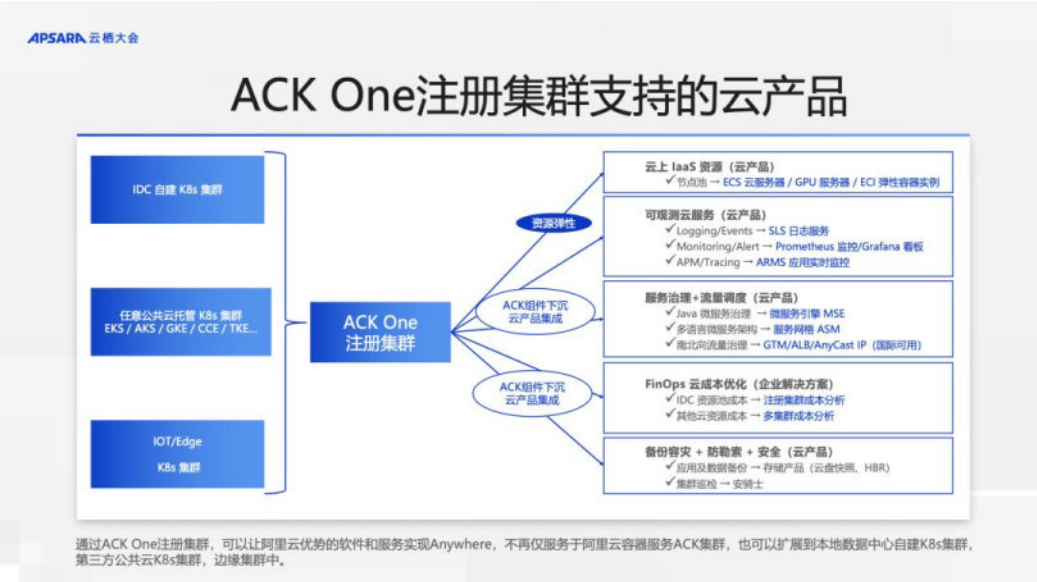


首先，您要有一个 K8s 集群，之后在 K8s 集群安装注册集群的 agent pod，Agent pod 会反向连接 ACK One 注册集群服务端并完成管控通道的建立，之后，ACK One 注册集群管控可以下发 K8s API 请求，并通过 Agent pod 的中转到自建 K8s 集群的 API Server 中。从而通过 ACK One 注册集群，可以获得云上云下一致的运维体验。

具体来说包括：ASM 服务网格，微服务引擎 MSE，Prometheus 监控，ARMS 应用性能监控，SLS 日志，安全，FinOps。以上所有云上 ACK 集群具备的能力，都可以通过注册集群在三方公共云 K8s 集群和自建 K8s 集群中落地实现。

另外，为了应对本地数据中心资源不足，无法应对业务流量增长的问题，ACK One 注册集群支持云上弹性，可以将云上的 ECS/ECI 计算资源，包括 CPU 和 GPU 资源，加入到自建 K8s 集群中，由自建 K8s 调度云上资源，应对业务流量增长。

可以说，您依然完全控制自建 K8s 集群，同时，获得了云上 ACK 集群才具有的强大扩展能力。



通过 ACK One 注册集群，可以让阿里云优势的软件和服务实现 Anywhere，不再仅服务于阿里云容器服务 ACK 集群，也可以扩展到本地数据中心自建 K8s 集群，第三方公共云 K8s 集群中。包括：ECS/ECI 等 IaaS 资源，可观测服务，服务治理，FinOps 成本管理，安全等。使用注册集群，您依然完全控制自建的 K8s 集群，但同时获得了阿里云云产品带来的扩展能力。

场景 1：云上弹性，通过云上弹性，我们可以将云上的计算资源加入到本地数据中自建 K8s 集群中，应对业务流量的增长。同时，注册集群调度器支持对不同计算资源设置不同的调度优先级，例如：可以优先调度本地数据中心中的节点，当本地数据中心资源用尽后，调度使用云上的按量计费节点池 ECS，或者使用云上 Virtual Node 弹性容器实例 ECI 处理业务流量。在充分使用本地计算资源后，再使用云上资源，节省成本。

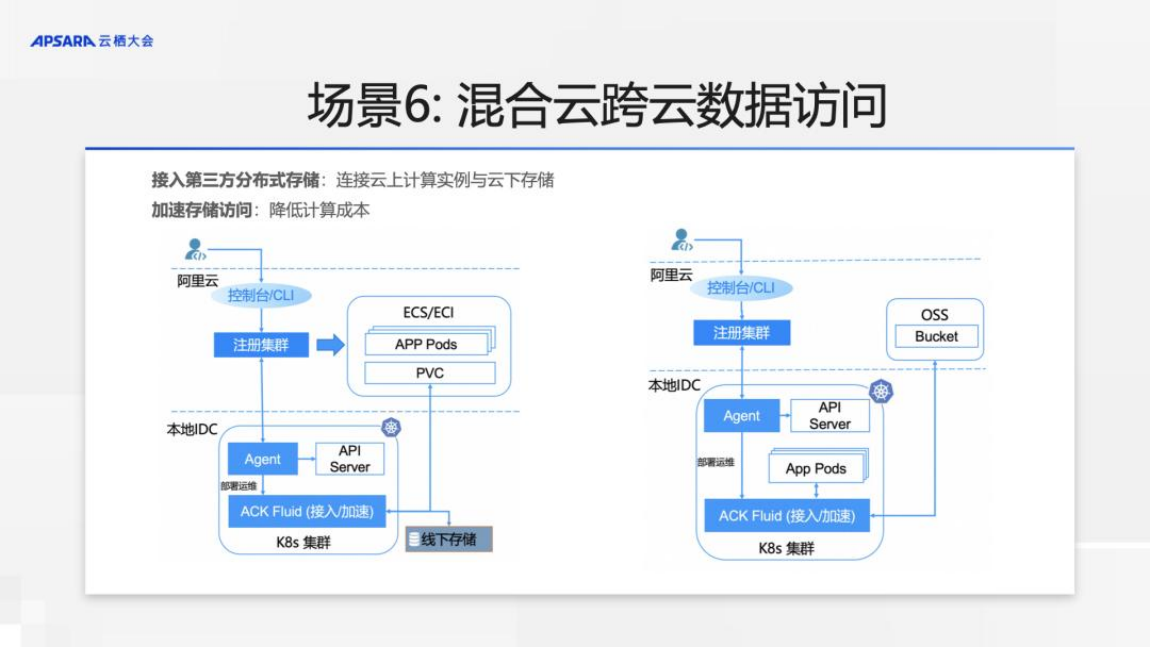
场景 2：可观测性。通过注册集群，可以将 ACK 集群的可观测能力，包括监控/日志/事件/告警，下沉到自建 K8s 集成中，一套可观测方案，同时用于容器服务 ACK 集群，自建 K8s 集群，三方公共云 K8s 集群，实现云上云下一致的运维体验。

容器服务 ACK 集群具有强大的安全管理能力，包括配置巡检、运行时安全检查和策略管理。通过注册集群，可以将 ACK 集群的安全能力，下沉到自建 K8s 集群和三方公共云 K8s 集群中。通过一个安全大盘，查看管理所有集群的安全风险，提高安全水位。

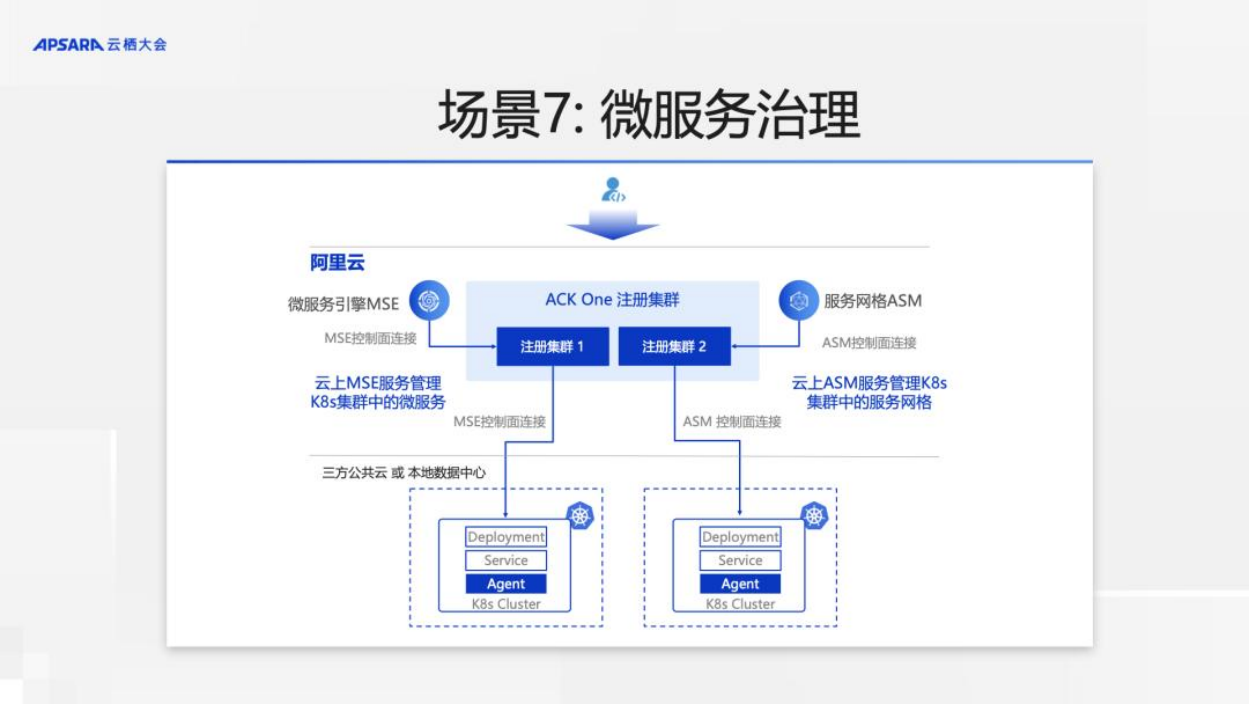
在成本治理方面，容器服务 ACK 提供完整的方案，支持以集群、命名空间、节点池、应

用多角度的成本洞察，基于洞察结果实施成本的治理，包括浪费发现、成本预测、预算告警等。通过注册集群，可以使用 ACK 集群的成本管理能力，分析和治理自建 K8s 集群和三方公共云 K8s 集群，降低成本。

场景 5：应用与数据的备份恢复，通过注册集群，可以使用云上数据备份与恢复能力，备份自建 K8s 集群和第三方公共云 K8s 集群中的应用与数据，并在 ACK 集群中恢复 ，实现跨地域的应用与数据容灾。



场景 6：混合云跨云数据访问。为了应对存算分离架构带来数据访问接口类型多和访问速度慢的问题，通过注册集群和 ACK Fluid 组件，可以通过标准 PV/PVC 接口实现云上计算实例访问云下自建存储资源，也可以通过 Fluid 分布式缓存加速云下计算资源访问云上 OSS 对象存储，缩短 GPU 等高成本计算资源的等待时间，提高了运行速度，降低计算成本。



阿里云提供强大的微服务治理能力，包括微服务引擎 MSE 和服务网格 ASM，通过注册集群的控制面连接，云上 MSE 和 ASM 可以管理三方公共云和本地数据中心 K8s 集群中的微服务。实现一套微服务方案，管理所有集群的微服务，降低运维复杂度，提高了稳定性。

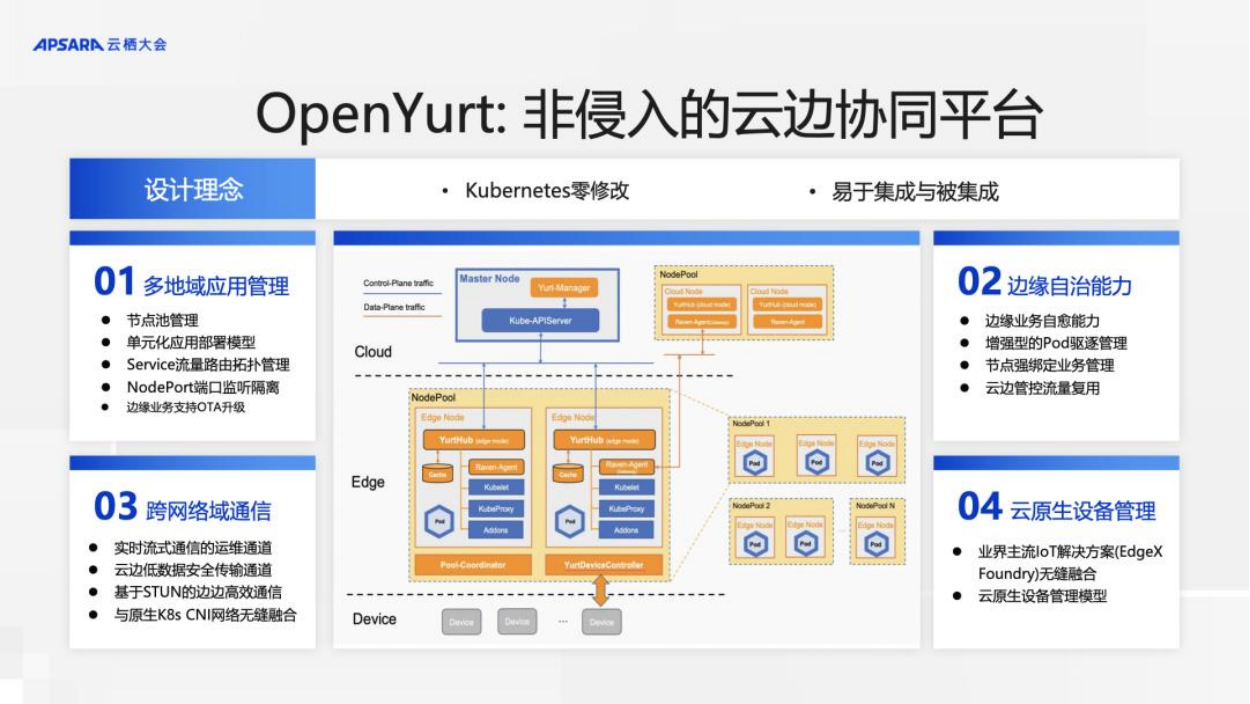
3. ACK Edge 边缘集群：接入和管理边缘计算资源

边缘集群 - 接入和管理边缘计算资源下面我们介绍一下边缘集群，边缘集群可以接入和管理边缘的计算资源，也就是机器和各种终端设备。您可以在云上创建边缘集群，并将边缘的机器和设备远程接入云上集群，实现通过标准 K8s 集群的方式调度使用这些机器和设备，从而省去在边缘自建集群的成本。

下面我们看下边缘集群的架构，上半部分是边缘集群 ACK Edge 的控制面，在阿里云上，复用 ACK Pro 的控制面（APIServer/ETCD 等），保证稳定性，集成开源 OpenYurt 提供边缘节点自治能力，最后融合了阿里云的日志监控安全能力。

下半部分是边缘部分，可以接入管理数据中心或者机房中的机器，包括 CDN、IDC、工厂、园区、楼宇等。也可以接入各种终端设备，包括车载、交通设备等。

淘麦郎使用边缘集群 ACK Edge，提供验票服务，人均验票时间缩短 70%，成功服务了北京冬奥会和杭州亚运会等大型赛事；影石使用边缘集群 ACK Edge，管理工厂中的计算设备，从云端直接下发指令到达工厂，极大提高了生产效率。



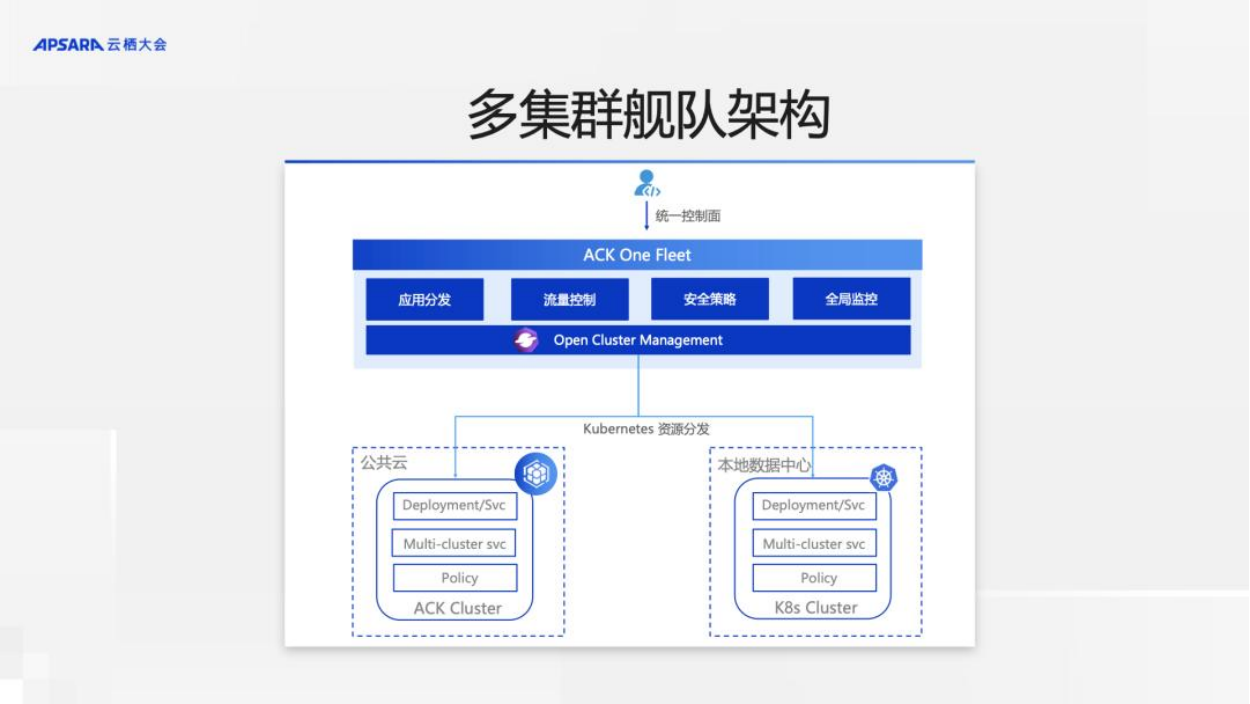
OpenYurt 是业界首个非侵入的云边协同的云原生平台，在中间的架构图中可以看到，master 组件部署在云端，可以方便的管理分散的边缘机器和设备。图中蓝色框是原生 K8s 组件，橙色框中表示的是 OpenYurt 组件。OpenYurt 没有对 K8s 做任何修改，保证 OpenYurt 很容易集成与被集成 。

OpenYurt 的核心能力主要 4 个方面，多地域应用管理，边缘自治，跨地域网络通信，以及云原生设备管理。

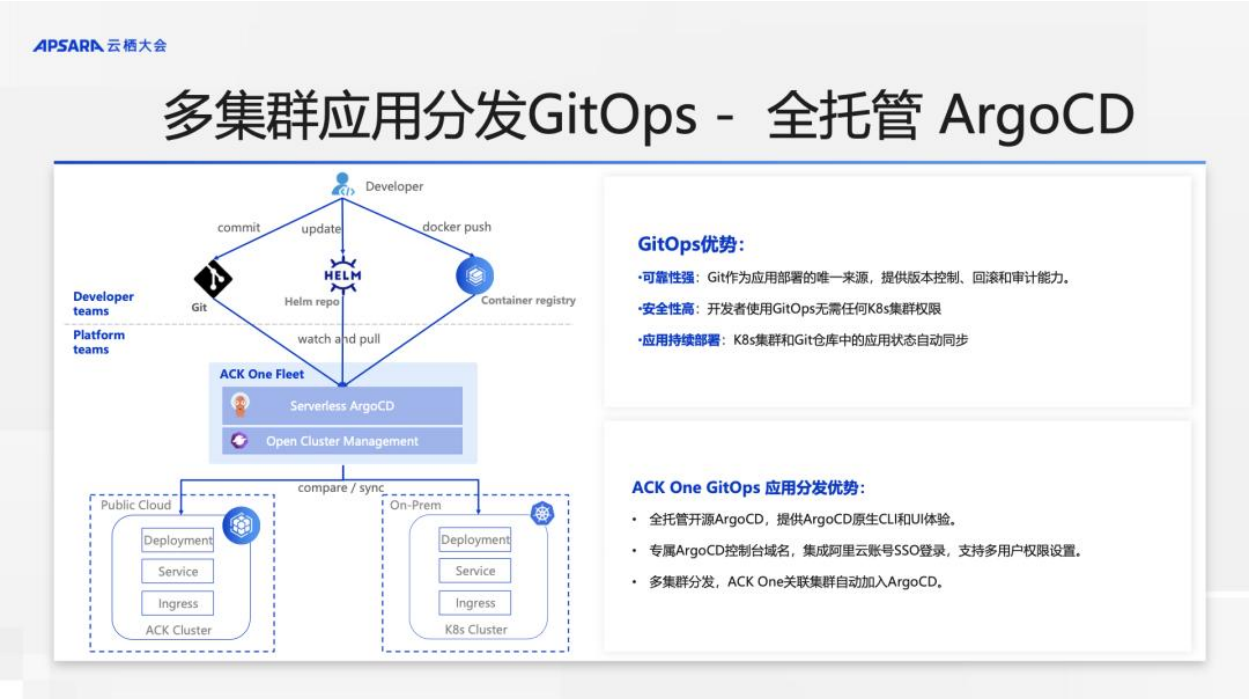
4. ACK One 多集群舰队：统一管理多个 K8s 集群

之前讲到的注册集群和边缘集群解决了各种形态位置的集群和边缘计算资源接入到云上的问题。

下面我们介绍，当我们有多个 K8s 集群的时候，如何统一管理，就要 ACK One 多集群舰队。



首先我们看下多集群舰队的架构，多集群舰队集成使用开源 Open Cluster Management，统一管理多个 K8s 集群，可以是公共云的 ACK 集群，本地数据中心 K8s 集群。多集群舰队对用户提供统一的控制面，提供应用分发，流量控制，安全策略，全局监控等多集群管理能力 。

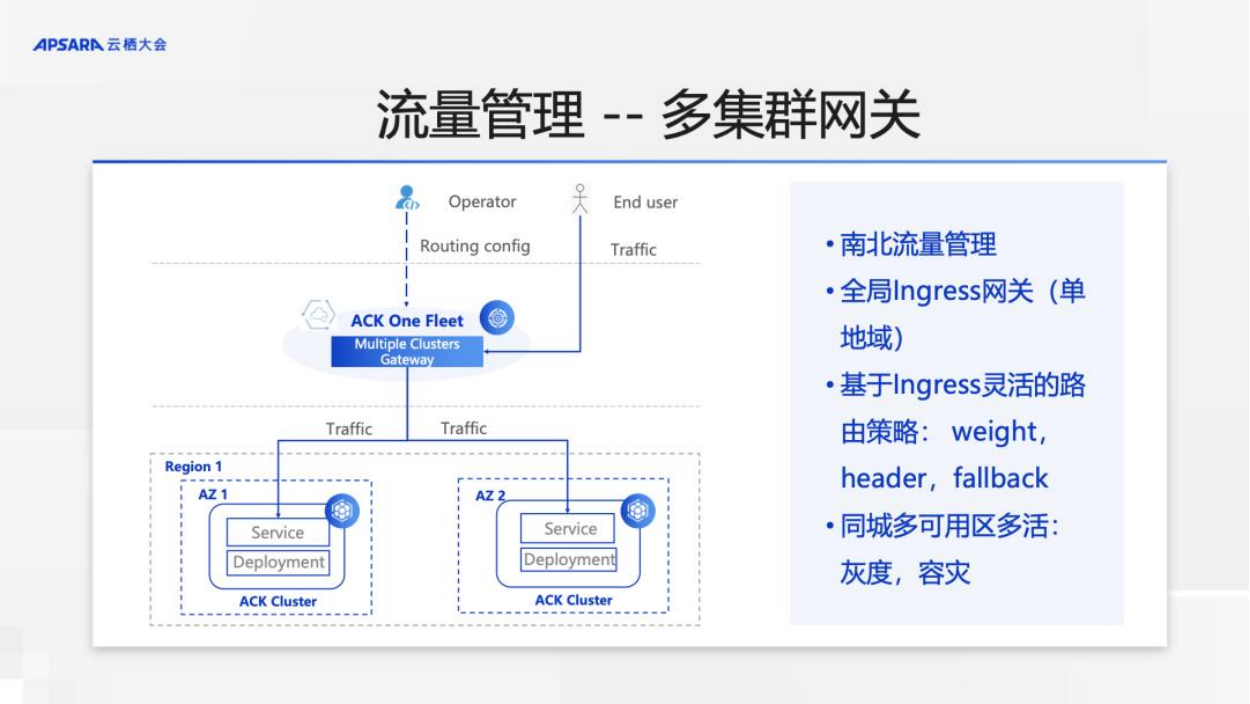


我们来看多集群的应用分发，多集群舰队托管了开源 ArgoCD 提供 GitOps 应用分发能力，开发团队可以通过在 Git 中定义应用部署 yaml 文件，实现应用的多集群部署。

使用 GitOps 具有强可靠性。Git 作为部署的唯一来源，提供了版本控制、回滚和审计能力。同时开发人员也不需要任何 K8s 集群的权限，安全性高。最后，GitOps 提供持续部署的能力，保证集群中应用的状态和 Git 仓库中定义的状态一致，避免误操作，保证稳定性。

ACK One 多集群 GitOps 在此基础上，提供了原生 ArgoCD 的 CLI 和 UI 体验，并提供专属的 ArgoCD 控制台域名，集成阿里云账号 SSO，支持多用户权限等。

基于 ACK One GitOps 实现多集群 CICD 流程。在流程图中，我们首先向业务代码仓库提交代码修改，之后由 ACR EE 或者全托管 Argo 工作流集群构建并推送镜像到 ACR EE，ACK One GtiOps 检查 ACR EE 中镜像版本的变更，并回写到应用部署仓库，也就是修改部署 yaml，ACK One GtiOps 自动向 Dev 集群部署新的镜像版本，在验证通过后，在维护窗口可以手工向生产集群部署。实现了从 CI 到多集群部署的完整 CICD 流程。



下面我们来看多集群南北流量管理，为了保证应用的容灾，我们一般会在不同可用区的 2 个集群部署相同的应用，并在前端部署负载均衡，传统的负载均衡一般会采用智能 DNS 的方案或者 4 层 LoadBalancer 方案，但 DNS 方案的问题在于客户端缓存，导致在问题发生时，切换不及时，造成业务损失。4 层 LoadBalancer 无法实现基于 7 层的路由转发，灵活性差。

为了解决这个问题，我们推出多集群网关，提供全局 Ingress，实现七层的负载均衡，多集群网关本身也支持跨可用区的高可用。因为是七层的网关，我们可以灵活的定制路由，实现基于权重、http header、自动 fallback 的路由策略。

通过多集群网关，我们可以实现应用同城多可用区多活容灾。在应用完成多集群部署后，我们还需要对应用的状态进行监控，ACK One 舰队管理提供多集群全局监控功能，汇总多个集群的监控指标，提供多集群视角的全局监控大盘。在截图中可以看到，一个大盘可以查看集群 1 和集群 2 的监控指标。

5. 全托管 Argo 工作流集群全新发布

最后一部分，我来介绍下全托管 Argo 工作流集群。

随着科技的发展，仿真计算、科学计算等离线作业在企业中的应用越来越多，在云原生领域，如何编排以及高效、低成本运行这些离线作业，是必须回答的问题。

ACK One 推出全托管跨地域 Argo 工作流集群，托管 CNCF 毕业项目 Argo Workflow，使用弹性容器实例 ECI，以无服务器方式在多可用区/多地域调度运行工作流，支持大规模工作流运行的同时降低计算成本。

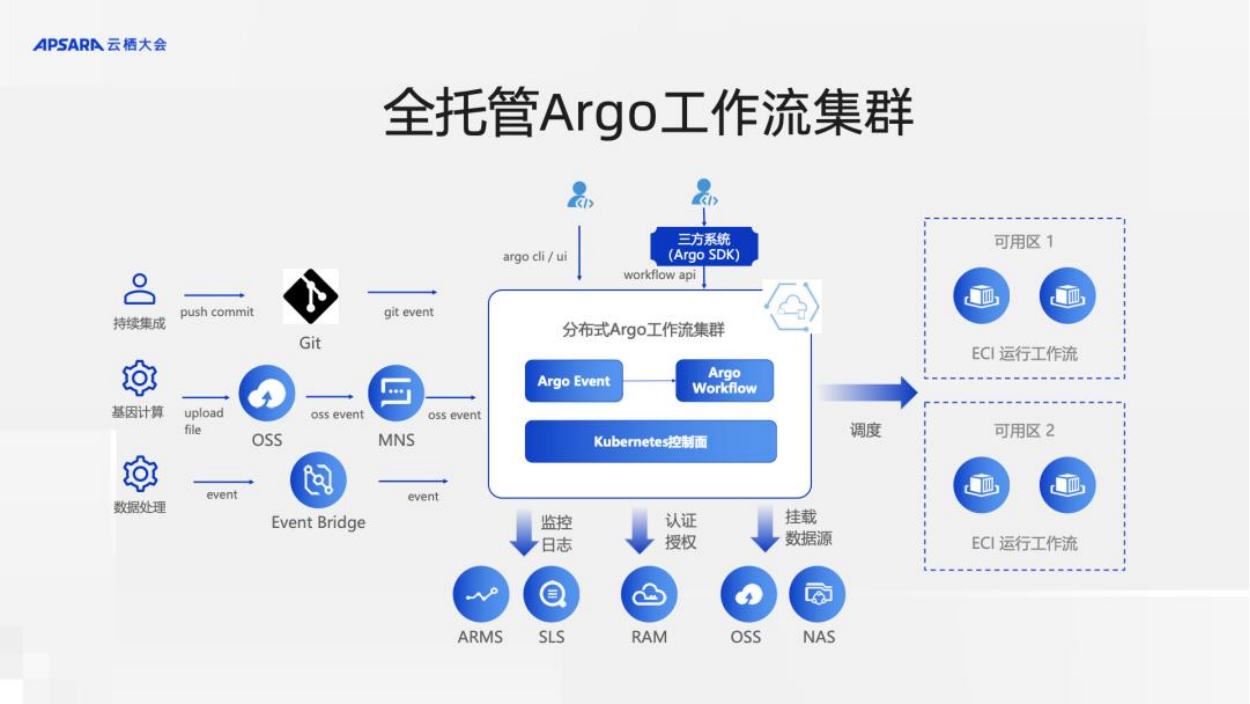
为了应对 AI 等数据密集性业务和存算分离架构，Argo 工作流集群支持分布式数据缓存，加速文件访问，减少运行时间，降低成本。

此外：全托管 Argo 工作流集群支持：

- 资源预测 – 通过负载感知自动调整资源规格
- 规模化计算 – 控制面进行了性能优化支持千级别并发工作流，数万 Pod 并行计算任务
- 事件驱动 - 支持 Git、阿里云 MNS、OSS 等事件源触发工作流自动运行

相比开源自建 Argo 工作流系统，利用 Serverless 技术，全托管 Argo 工作流集群可实现 30% 的资源成本节省；利用分布式数据缓存，数据读性能提高 15 倍；通过控制面优化和云上极致弹性，最大并行计算规模提升 10 倍。

泛生子使用全托管 Argo 工作流集群在 12 小时内完成处理数千例样本的处理，速度提升 50%，成本下降 30%。



看下分布式 Argo 工作流集群的架构，它托管开源 Argo workflow，提供开源原生的 CLI、UI、API、SDK 等接口，保证现有开源自建 Argo 工作流系统可无缝迁移。同时，可以使用工作流集群的 API SDK 接口，包装工作流集群，构建自己的工作流提交系统，以适应具体的业务场景。我们很多仿真和科学计算客户就是将工作流集群作为后端引擎使用。

工作流集群托管 Argo event 项目，支持事件驱动编程模型，支持 Git、阿里云 OSS、MNS、EventBridge 等事件源触发工作流自动运行，自动完成 CI、数据处理等离线任务。

分布式 Argo 工作流集群使用 Serverless 方式运行工作流，无需运维 worker 节点，利用云上极致算力资源，可实现同时运行数万 Pod 的大规模工作流， 同时提供监控日志等开箱即用的运维体验。

最后，我们简单总结下，分布式云面对的场景多种多样，挑战很多，为此，ACK One 提供注册集群能力，可以将非 ACK 集群接入 ACK One，从而实现云上云下集群统一管理。ACK One 提供边缘集群，实现云上集群管理边缘机器与设备，同时实现边缘自治。

多集群舰队提供多集群 GitOps 和全局 Ingress 实现应用的同城多活容灾。全托管 Argo 工作流集群，实现工作流的编排，以 Serverless 方式运行工作流，降低成本，应对持续集成、科学计算、仿真计算、数据处理等场景。

大家可以根据具体业务场景选择相应的能力，最终实现简单的跨云协同，让业务管理更加高效。